Lab 03 – Exploring and visualising data

ENVX1002 Handbook

Semester 1, 2025

Learning outcomes

At the end of this computer practical, students should be able to:

- ☐ Import and prepare data for visualisation
- ☐ Create basic plot types using ggplot2 (histograms, boxplots, bar plots, scatterplots)
- ☐ Customise plots with appropriate labels, colours, and themes
- ☐ Identify and visualise distribution properties (normality, skewness, kurtosis)
- ☐ Interpret visualisations to draw meaningful conclusions
- ☐ Export and save plots for reports

Before we begin

For this lab, you will need to:

1. Create a new Quarto document in your **project folder** (or create a new RStudio project) to simplify file management and reproducibility
2. Download the data files from the links provided in Exercise 1
3. Make sure you have the following packages installed and loaded:
   - `tidyverse`: For data manipulation and visualization (includes `ggplot2`)
   - `moments`: For calculating skewness and kurtosis
   - `patchwork`: For combining multiple plots

You can install packages using `install.packages("package_name")` if needed, and load them with `library(package_name)`.

Exercise 1: Dataset exploration

Exploring the `pie_crab` dataset

In this lab, we'll work with two environmental datasets. They can be downloaded from the links below:

- pie_crab.csv: Crab size measurements across different sites and latitudes
- hbr_maples.csv: Maple seedling measurements from different watersheds

By now, you should know how to use `read_csv()` to read these datasets into R. If you need a refresher, refer to the previous labs or ask for help from a demonstrator (if available).

Let's start by exploring the `pie_crab` dataset. We've done this before, but it's always good to refresh our memory as these functions are extremely common to use.

- The `str()` function shows us the structure of the dataset, including variable names, types, and a preview of the values.
- The `head()` function shows the first six rows of the dataset, giving us a quick look at the data.
- The `summary()` function provides descriptive statistics for each variable, including minimum, maximum, mean, and quartiles for numeric variables.

```
str(pie_crab)
head(pie_crab)
summary(pie_crab)
```

> Question 1
>
> What variables are in the dataset? What are the data types of each variable? Are there any missing values? What are the ranges of the numeric variables?

From our exploration, we can see that the pie_crab dataset contains information about:

- `date`: When the crabs were measured
- `latitude`: The latitude where the crabs were collected
- `site`: The specific collection site
- `size`: The size of the crabs in millimeters
- `air_temp`: Air temperature in degrees Celsius
- `water_temp`: Water temperature in degrees Celsius

Identifying factors in the data

One of the first steps in data exploration is to determine if your data types are recognised correctly by R. Looking at the output of `str()` can help you identify variables that might be better represented as different data types.

In our dataset, `site` and `name` are character variables with repeating values. When categorical variables like these have a limited number of possible values that repeat throughout the dataset, they're good candidates to be converted to factors. Factors are R's way of representing categorical data efficiently.

We can check how many unique values these variables have using the `unique()` function:

```
unique(pie_crab$site)
unique(pie_crab$name)
```

For a dataset with 392 observations, having only a few unique sites and names suggests that these variables should be factors. Let's count exactly how many unique values we have:

```
length(unique(pie_crab$site))
length(unique(pie_crab$name))
```

We can convert these character variables to factors using the `factor()` or `as.factor()` functions:

```
# Convert site and name to factors
pie_crab$site <- factor(pie_crab$site)
pie_crab$name <- factor(pie_crab$name)

# Alternatively, use the as.factor() function:
# pie_crab$site <- as.factor(pie_crab$site)
# pie_crab$name <- as.factor(pie_crab$name)
# We have intentionally commented out (#) the code to avoid running it

# Check the structure again to confirm the conversion
str(pie_crab)
```

Notice how we've converted the specific variables to factors and used the assignment operator `<-` to update the dataset. This is a common pattern in R, where we update the dataset in place. The output of `str()` now shows these variables as factors with their levels (unique values) listed.

Converting to factors is important because:

1. It makes R treat the data appropriately in statistical analyses
2. It can make visualisations more informative
3. It improves memory efficiency for large datasets

Exercise 2: Building visualizations with `ggplot2`

The grammar of graphics

The ggplot2 package is based on the "grammar of graphics," a framework that breaks visualisations into components, similar to how grammar breaks language into parts of speech. This approach makes it possible to create complex visualisations by combining simple elements.

The key components (or layers) include:

- **Data**: The dataset being visualised
- **Aesthetics**: Mappings from data variables to visual properties
- **Geometries**: The shapes used to represent the data
- **Facets**: Subplots that show different subsets of the data
- **Statistics**: Transformations of the data (e.g., counts, means)
- **Coordinates**: The space in which the data is plotted
- **Themes**: Visual styling of non-data elements

Let's learn how to create visualisations using this approach by building a plot step by step, explaining each component along the way. **As you follow along, you are improving the code that produces the plot – not re-writing it – at each step.**

Step 1: The canvas
Every ggplot2 visualisation starts with a blank canvas:

```
# Start with an empty canvas
ggplot()
```

This creates an empty plotting space. It doesn't show anything yet because we haven't specified any data or how to visualise it.

Step 2: Adding data
Next, we tell ggplot2 what data to use. Ideally the data is a `data.frame` or `tibble` object (which you will know by using `str()` previously):

```
# Add data to the plot
ggplot(data = pie_crab)
```

We've now told ggplot2 to use the `pie_crab` dataset, but we still cannot see anything because we have not specified which variables to plot – or how to represent them.

Step 3: Mapping aesthetics
Aesthetics map variables in your data to visual properties in the plot. They are a function `aes()` on their own. Think of aesthetics as how you woul define `x`, `y` and other dimensions in the plot.

```
# Map variables to visual properties
ggplot(data = pie_crab, mapping = aes(x = size))
```

Here, we've mapped the `size` variable to the x-axis. This variable is something that exists in the `pie_crab` data frame. We still cannot see any data points because we haven't specified how to represent the data (e.g., as points, bars, or lines).

Step 4: Adding a Geometry
Geometries determine how the data is represented visually. They are functions that are prefixed by `geom_*` so that users know what they are meant to do. In most cases it is clear what type of plot is being create by reading the name of the geometry function(s) in the plot code.

```
# Add a histogram geometry
ggplot(data = pie_crab, mapping = aes(x = size)) +
  geom_histogram()
```

Now we can see the data! We've added a histogram geometry (`geom_histogram()`), which counts the number of observations falling into bins along the x-axis. The `+` operator adds layers to the plot.

Notice the message about the default bin width. ggplot2 automatically chose 30 bins, but we can adjust this.

Step 5: Customizing the Geometry

Let's customize our histogram:

```
# Customize the histogram
ggplot(data = pie_crab, mapping = aes(x = size)) +
  geom_histogram(bins = 20, fill = "skyblue", color = "black")
```

We've made several changes:

- `bins = 20`: Changed the number of bins to 20
- `fill = "skyblue"`: Set the fill color of the bars to sky blue
- `color = "black"`: Set the outline color of the bars to black

These are fixed properties applied to all bars, not mappings from data variables. For colours you may search "r colours" in your web browser to see what available colours you can use (it's a *lot*).

Step 6: Adding Labels and Titles

Good visualisations have clear labels. Titles are optional – but the option is available if you need it. We can add all of these in the next layer using the `labs()` function:

```
# Add informative labels
ggplot(data = pie_crab, mapping = aes(x = size)) +
  geom_histogram(bins = 20, fill = "skyblue", color = "black") +
  labs(
    title = "Distribution of Crab Sizes",
    x = "Size (mm)",
    y = "Count"
  )
```

The `labs()` function adds various text elements to the plot:

- `title`: The main title of the plot
- `x`: The x-axis label
- `y`: The y-axis label

Step 7: Applying a Theme

Themes control the overall appearance of the plot:

```
# Add a theme for consistent styling
ggplot(data = pie_crab, mapping = aes(x = size)) +
```

```
  geom_histogram(bins = 20, fill = "skyblue", color = "black") +
  labs(
    title = "Distribution of Crab Sizes",
    x = "Size (mm)",
    y = "Count"
  ) +
  theme_minimal()
```

The `theme_minimal()` function applies a minimalist theme with a white background and subtle grid lines. Other common themes include:

- `theme_classic()`: No grid lines, simple axes
- `theme_light()`: Light background with subtle grid lines
- `theme_dark()`: Dark background for presentations

Bonus: Adding multiple geometries

One of the powerful features of ggplot2 is the ability to layer multiple geometries:

```
# Add a density curve on top of the histogram
ggplot(data = pie_crab, mapping = aes(x = size)) +
  geom_histogram(aes(y = after_stat(density)),
    bins = 20,
    fill = "skyblue", color = "black"
  ) +
  geom_density(color = "red", linewidth = 1) +
  labs(
    title = "Distribution of Crab Sizes",
    x = "Size (mm)",
    y = "Density"
  ) +
  theme_minimal()
```

In this plot:

- We've changed the y-axis of the histogram to show density instead of count using `aes(y = after_stat(density))`
- We've added a density curve with `geom_density()`
- We've set the density curve color to red and increased its line width

Question 2

1. What's the difference between setting a fixed property (like `fill = "blue"`) and mapping a variable to an aesthetic (like `aes(fill = site)`)?
2. How would you modify the histogram to have more or fewer bins? Use `?geom_histogram` to help you think of an answer.
3. What would happen if you changed the order of the `geom_histogram()` and `geom_density()` layers?

These questions can typically be answered by making changes to the code to view the differences.

Exercise 3: Analysing environmental variables

Now that we understand the grammar of graphics approach, let's analyse a different variable in our dataset.

Examining water temperature distribution

Let's examine the distribution of water temperatures across our sampling sites:

```
# Create a basic histogram of water temperatures
ggplot(pie_crab, aes(x = water_temp)) +
  geom_histogram(bins = 15) +
  labs(
    title = "Distribution of Water Temperatures",
    x = "Water Temperature (°C)",
    y = "Count"
  )
```

The histogram shows us the frequency distribution of water temperatures. We can see the shape of the distribution, including any skewness or unusual patterns.

```
# Add a density curve
ggplot(pie_crab, aes(x = water_temp)) +
  geom_histogram(aes(y = after_stat(density)),
    bins = 15,
    fill = "lightblue", colour = "black"
  ) +
  geom_density(colour = "red") +
  labs(
    title = "Distribution of Water Temperatures",
    x = "Water Temperature (°C)",
    y = "Density"
  )
```

Adding a density curve helps us see the overall shape of the distribution more clearly.

Question 3

What is the shape of the distribution of water temperatures? Does the distribution appear to be normal? Are there any outliers?

Skewness and Kurtosis

To quantify the shape of the water temperature distribution, we can calculate skewness and kurtosis:

```
# Calculate skewness and kurtosis for water temperature
skewness_value <- skewness(pie_crab$water_temp)
kurtosis_value <- kurtosis(pie_crab$water_temp)
skewness_value
kurtosis_value
```

Interpreting these values:

- **Skewness** measures the asymmetry of the distribution:
  - 0 = symmetric (like a normal distribution)
  - More than, > 0 = right-skewed (tail extends to the right)
  - Less than, < 0 = left-skewed (tail extends to the left)
- **Kurtosis** measures the "tailedness" of the distribution:
  - 3 = normal distribution (in the `moments` package, this is sometimes normalised to 0)
  - More than, > 3 = leptokurtic (heavy-tailed, more outliers)
  - Less than, < 3 = platykurtic (light-tailed, fewer outliers)

The skewness value of approximately 0.5 confirms our visual observation that the water temperature distribution is moderately right-skewed. The kurtosis value of approximately 2.5 indicates the distribution has slightly lighter tails than a normal distribution.

These numerical measures help us quantify what we observe visually in the histograms and density plots. Now that we understand the overall distribution of our data, let's explore how it varies across different groups.

Exercise 4: Comparing groups

Now that we've examined the overall distribution of crab sizes, let's compare sizes across different groups.

Creating boxplots to compare sites

Boxplots are excellent for comparing distributions between two or more groups:

```
# Create boxplots of crab sizes by site
ggplot(pie_crab, aes(x = site, y = size)) +
```

```
  geom_boxplot(fill = "skyblue") +
  labs(
    title = "Crab Sizes by Site",
    x = "Site",
    y = "Size (mm)"
  )
```

A boxplot shows:

- The median (middle line)
- The interquartile range (IQR) from the 25th to 75th percentile (the box)
- The whiskers (typically extend to 1.5 × IQR)
- Outliers (points beyond the whiskers)

To see the actual data points alongside the boxplots we can add another geometric layer. You can see how it may or may not be useful to readers. In most cases adding the data points is **not** necessary, but in some cases it could be useful in the exploration phase (i.e. not the final plot for publication).

```
# Add points to see the actual data
ggplot(pie_crab, aes(x = site, y = size)) +
  geom_boxplot(fill = "skyblue", alpha = 0.5) +
  geom_jitter(width = 0.2, alpha = 0.5) +
  labs(
    title = "Crab Sizes by Site",
    x = "Site",
    y = "Size (mm)"
  )
```

We've added:

- `geom_jitter()` to add individual data points with a slight horizontal jitter to avoid overplotting
- `alpha = 0.5` to make both the boxplots and points semi-transparent
- `width = 0.2` to control the amount of horizontal jittering

> Question 4
>
> How do crab sizes vary across different sites? Which site has the largest median crab size? Which site shows the most variability in crab sizes? Are there any outliers at specific sites? Answer these questions in a descriptive manner using the plot.

Here we are exercising our ability to see patterns from data visualisations and using them to make certain observations about the data.

Exploring the relationship between latitude and size

Let's examine if there's a relationship between latitude and crab size:

```r
# Create a scatterplot of size vs. latitude
ggplot(pie_crab, aes(x = latitude, y = size)) +
  geom_point(alpha = 0.5) +
  labs(
    title = "Crab Size vs. Latitude",
    x = "Latitude",
    y = "Size (mm)"
  )
```

Scatterplots show the relationship between two continuous variables. Each point represents a single observation.

To help visualise the trend, we can add a trend line (something that we will cover in greater detail once we look at linear models in Week 7):

```r
# Add a trend line
ggplot(pie_crab, aes(x = latitude, y = size)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", se = TRUE) +
  labs(
    title = "Crab Size vs. Latitude",
    x = "Latitude",
    y = "Size (mm)"
  )
```

We've added:

- `geom_smooth(method = "lm")` to add a linear regression line
- `se = TRUE` to include the standard error as a shaded confidence band

> Question 5
>
> Is there a relationship between latitude and crab size?

Exercise 5: Faceting and grouping

So far, we've created separate plots for different analyses. Now, let's explore techniques for comparing multiple groups or variables within a single plot.

Exploring the `hbr_maples` dataset

Let's switch to our second dataset, which contains measurements of maple seedlings from different watersheds:

```r
# Examine the structure of the maples dataset
str(hbr_maples)
```

```r
# View the first few rows
head(hbr_maples)
```

```r
# Get a summary of the variables
summary(hbr_maples)
```

Now, let's create histograms of stem length by watershed using faceting:

```r
# Create histograms of stem length by watershed
ggplot(hbr_maples, aes(x = stem_length)) +
  geom_histogram(bins = 20, fill = "forestgreen", colour = "black") +
  facet_wrap(~watershed) +
  labs(
    title = "Distribution of Stem Lengths by Watershed",
    x = "Stem Length (mm)",
    y = "Count"
  )
```

```r
hbr_maples |>
  select(watershed, stem_length) |>
  group_by(watershed) |>
  summarise(median = median(stem_length, na.rm = TRUE))
```

The `facet_wrap()` function creates separate panels for each value of the specified variable. This allows us to compare distributions across groups.

> Question 6
>
> How do stem lengths differ between watersheds? Which watershed shows more variability in stem lengths? Are the distributions similarly shaped? Use only the visualisations to answer these questions.

Comparing leaf area and stem length

Let's examine the relationship between leaf area and stem length, comparing across watersheds:

```r
# Create a scatterplot of leaf area vs. stem length, coloured by watershed
ggplot(hbr_maples, aes(x = stem_length, y = corrected_leaf_area, colour =
watershed)) +
  geom_point(alpha = 0.7) +
  labs(
    title = "Leaf Area vs. Stem Length",
    x = "Stem Length (mm)",
    y = "Corrected Leaf Area (cm²)",
```

```
    colour = "Watershed"
  )
```

Here, we've mapped the `watershed` variable to the `colour` aesthetic, which automatically creates a color-coded legend.

Let's add separate trend lines for each watershed:

```
# Add separate trend lines for each watershed
ggplot(hbr_maples, aes(x = stem_length, y = corrected_leaf_area, colour =
watershed)) +
  geom_point(alpha = 0.7) +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    title = "Leaf Area vs. Stem Length",
    x = "Stem Length (mm)",
    y = "Corrected Leaf Area (cm²)",
    colour = "Watershed"
  )
```

When we include `colour = watershed` in the global aesthetics, ggplot2 automatically applies this grouping to all geometries, including `geom_smooth()`. This creates separate trend lines for each watershed.

> Question 7
>
> Is there a relationship between stem length and leaf area? Does this relationship differ between watersheds? What might explain these differences from an ecological perspective?

Exercise 6: Bonus take-home

These exercises are designed for you to practice the visualisation techniques we've covered in this lab. You can complete them during the lab if you finish early, or at home for additional practice.

Basic visualisation practice

1. Create a histogram of the `air_temp` variable in the crabs dataset. Calculate and interpret its skewness and kurtosis.
2. Create boxplots comparing the `leaf_dry_mass` between watersheds in the maples dataset. What do you observe?
3. Create a scatterplot examining the relationship between `stem_dry_mass` and `leaf_dry_mass` in the maples dataset, with points coloured by watershed.

Advanced challenge: `patchwork`

The patchwork package allows you to combine multiple plots into a single figure. This is useful for creating complex visualisations that tell a story about your data.

```r
# Load the patchwork package
library(patchwork)
```

Let's create a few plots and then combine them:

```r
# Create multiple plots
p1 <- ggplot(pie_crab, aes(x = size)) +
  geom_histogram(fill = "skyblue", colour = "black") +
  labs(title = "Distribution of Crab Sizes")

p2 <- ggplot(pie_crab, aes(x = latitude, y = size)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Size vs. Latitude")

p3 <- ggplot(pie_crab, aes(x = site, y = size)) +
  geom_boxplot(fill = "skyblue") +
  labs(title = "Size by Site")

# Combine plots (this is patchwork in action!)
p1 / (p2 + p3)
```

The patchwork syntax is intuitive:

- `/` arranges plots vertically (one above the other)
- `+` arranges plots horizontally (side by side)
- You can use parentheses to control the layout

Now, try these exercises:

4. Create a combined plot using patchwork that shows:
   - A histogram of stem lengths
   - A scatterplot of stem length vs. leaf area
   - Boxplots of stem lengths by watershed
   - Arrange these plots in a 2x2 grid
5. Create a combined plot that tells a story about the relationship between temperature and crab size:
   - A scatterplot of air temperature vs. crab size
   - A scatterplot of water temperature vs. crab size
   - A boxplot of crab sizes by site
   - Arrange the scatterplots side by side and the boxplot below them

Summary

In this lab, we've explored how to create and customize various types of visualisations using ggplot2. We've learned:

1. The grammar of graphics approach to building visualisations layer by layer

2. How to create and interpret histograms, density plots, boxplots, and scatterplots
3. How to quantify and interpret distribution properties like skewness and kurtosis
4. How to compare groups using boxplots and faceting
5. How to examine relationships between variables using scatterplots and trend lines
6. How to combine multiple plots using the patchwork package

These skills will be valuable for exploring and presenting data in future labs and assignments.

Additional Resources

- R for Data Science - Data Visualisation chapter
- ggplot2 documentation
- patchwork package documentation
- R Graph Gallery - Examples of various visualisations in R
- Cookbook for R - Graphs - Recipes for common visualisation tasks