**VNUHCM-UNIVERSITY OF SCIENCE**
**FACULTY OF INFORMATION TECHNOLOGY**


# REPORT

## INTRODUCTION TO BIG DATA

**< Seminar – RStudio & ggplot2 >**

**Student:** 21127104 - Doan Ngoc Mai
21127129 - Le Nguyen Kieu Oanh
21127229 - Duong Truong Binh
21127616 - Le Phuoc Quang Huy

**Lecturer:** Le Ngoc Thanh
Nguyen Ngoc Thao

**Teaching Assistant:** Do Trong Le
Bui Huynh Trung Nam

**Class:** 21KHDL

# Table of Contents

# Team Contributions

| Student ID | Member | Responsibilities | Completion |
|---|---|---|---|
| 21127104 | Đoàn Ngọc Mai | • Content preparation for Scale section<br>• Presentation of Scale section | 100% |
| 21127129 | Lê Nguyễn Kiều Oanh | • Content preparation for Grammar section<br>• Presentation of Grammar section | 100% |
| 21127229 | Dương Trường Bình | • Content preparation for RStudio section<br>• Presentation of RStudio section | 100% |
| 21127616 | Lê Phước Quang Huy | • Content preparation for Layer section<br>• Presentation of Layer section | 100% |

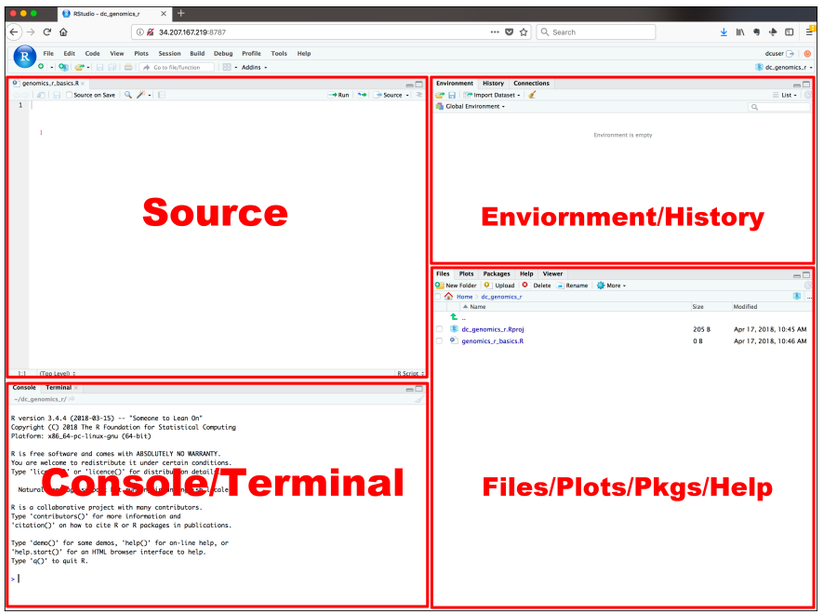
# 1 RStudio

## 1.1 Introduction

RStudio is a powerful integrated development environment (IDE) designed specifically for R, a programming language widely used in statistical computing and data analysis. Developed by RStudio, Inc. (now part of Posit), it has become the go-to IDE for R programmers since its initial release in 2011. RStudio's user-friendly interface and comprehensive feature set cater to both beginners and experienced R users, significantly enhancing productivity in data analysis and statistical computing tasks.

## 1.2 RStudio User Interface

The RStudio interface is carefully designed to provide intuitive access to all necessary tools for R programming and data analysis. Understanding this interface is crucial for efficient use of the IDE.




### a Main Window Structure

RStudio's layout consists of four main panes, each serving a specific purpose:

- **Source Editor**: This is where you write and edit R scripts. It supports syntax highlighting, code completion, and code folding, making it easier to write and manage complex scripts.
- **Console**: This pane allows direct interaction with the R interpreter. You can execute R commands here and see their output immediately.

- **Environment/History**: The Environment tab displays all objects in the current R session, while the History tab shows previously executed commands.
- **Files/Plots/Packages/Help**: This multi-purpose pane includes file management, plot viewing, package management, and access to R documentation.

This quadrant layout allows users to simultaneously view their code, execute commands, manage data objects, and access additional resources, creating a seamless workflow.

## 1.3 Key Features

RStudio offers a range of features that significantly enhance R programming and data analysis workflows. The following are some of the most important:

### a R Code Editing and Execution

RStudio's code editor is feature-rich and designed to boost productivity:

- **Syntax highlighting**: Automatically colors code elements for better readability and error detection.
- **Code completion**: Suggests function names, arguments, and object names as you type, speeding up coding and reducing errors.
- **Code folding**: Allows collapsing of code sections for better organization of large scripts.
- **Multiple file tabs**: Enables working on several scripts simultaneously, facilitating complex projects.
- **Easy code execution**: Use keyboard shortcuts (e.g., Ctrl+Enter) to run selected lines or entire scripts quickly.

### b Project and Workspace Management

RStudio's project management features help organize work effectively:

- **Project-based workflows**: Keep related files, data, and output together in a single project.
- **Session management**: Save and restore work environments.
- **Version control integration**: Built-in support for Git and SVN, with a GUI for common operations like commit, push, and pull.

## 1.4 Data Analysis Tools

RStudio provides robust tools for data analysis, a core strength of the IDE:

- **Data Import and Export**
  - Support for various file formats (CSV, Excel, JSON, etc.)
  - GUI import wizard for easy data loading, especially useful for beginners
  - Direct database connections for working with large datasets
- **Data Visualization**:
  - Integration with popular R plotting libraries (ggplot2, plotly)
  - Interactive plot viewer with zoom, pan, and export options
  - Support for dynamic and interactive visualizations

## 1.5 Reporting and Documentation

RStudio excels in creating reproducible research and professional reports:

- **R Markdown**: a powerful feature that allows the integration of R code, results, and narrative text
  - Create dynamic documents that combine code, output, and explanatory text
  - Supports multiple output formats (HTML, PDF, Word)
  - Interactive notebooks for exploratory data analysis and sharing insights
- **Shiny Apps**: a web application framework for R, deeply integrated with RStudio
  - Create interactive web applications directly from R code
  - Built-in templates and examples to get started quickly
  - Easy deployment options for sharing apps with collaborators or the public

## 1.6 Comparison with Other IDEs

While RStudio is the most popular IDE for R, it's worth comparing it to alternatives. The following table provides a detailed comparison:

RStudio stands out for its R-centric design, comprehensive feature set, and user-friendly interface, making it the preferred choice for many R users, from beginners to experts. However, each IDE has its strengths, and the choice often depends on specific user needs and preferences.

| **Feature** | **RStudio** | **Jupyter Notebooks** | **VS Code with R extensions** | **R GUI** | **Emacs + ESS** |
|---|---|---|---|---|---|
| Language Support | R-centric, supports others | Multi-language | Multi-language | R only | Multi-language |
| User Interface | User-friendly, integrated | Web-based, interactive | Customizable, modular | Basic, minimal | Highly customizable, text-based |
| R-specific Features | Comprehensive | Limited | Good with extensions | Basic | Extensive with ESS |
| Data Visualization | Integrated, interactive | Interactive, inline | Through extensions | Basic | Through add-ons |
| Version Control | Git integration | Limited | Excellent Git integration | None | Git support available |
| Extensibility | Addins, packages | Extensions, kernels | Large marketplace | Limited | Highly extensible |
| Learning Curve | Moderate | Low to moderate | Moderate | Low | Steep |
| Reproducibility | R Markdown, projects | Notebooks | Various tools | Manual | Org-mode, reproducible research tools |

Table 1: Comparison of RStudio with other IDEs

# 2 The Grammar in ggplot2

## 2.1 Overview

**The "ggplot2" package is a powerful R library for creating graphs and charts**. First appearing in 2007 as a successor to the less successful "ggplot," "ggplot2" was redeveloped by Hadley Wickham and has become one of the most popular R packages. The term "gg" in "ggplot2" comes from *The Grammar of Graphics* (1999), which introduced a systematic approach to representing charts and graphs.

The "Grammar of Graphics" concept stems from the question, "What is a graph?" and breaks it down into core elements:

- **Data:** The dataset used for creating the chart.
- **Aesthetic:** Selection of variables and their placement on the graph.
- **Geometry:** The type of chart (e.g., bar, line, scatter plot).
- **Facet:** Creation of smaller charts to compare different data aspects.
- **Scale:** Adjustment of axis lengths to fit the data.

- **Theme:** Editing of chart details (titles, labels, legends, fonts).

These elements form the foundation of the "Grammar of Graphics," allowing users to create various charts efficiently. The first three components (**Data**, **Aesthetic**, and **Geometry**) are essential and must be specified for data visualization.

## 2.2 Data

The first step in using `ggplot2` is to provide the data:

```
ggplot(data = house_df)
```

This alone doesn't produce a chart; it merely sets the data source.

## 2.3 Aesthetic

Aesthetic Mapping defines how variables map to chart components:

```
ggplot(data = house_df, aes(x = HuongCua, y = Gia.m2))
```

This maps `HuongCua` to the x-axis and `Gia.m2` to the y-axis. It creates a blank canvas for the chart. To complete the visualization, the Geometry component is required to specify the chart type.

## 2.4 Geometry

Geometry in `ggplot2` specifies the chart type. For example:

```
ggplot(data = house_df, aes(x = HuongCua, y = Gia.m2)) + geom_boxplot()
```

This creates a boxplot. Geometric functions follow the format `geom_GeometryName()`, such as `geom_boxplot()`, `geom_violin()`, or `geom_jitter()`.

`ggplot2` uses a layering system, with layers added using the plus sign (+).

By combining the three core elements—**Data**, **Aesthetic**, and **Geometry**—users can create various charts and graphs with different variables.

Figures 2.4.1, 2.4.2, and 2.4.3 illustrate the process of adding components in `ggplot2`:

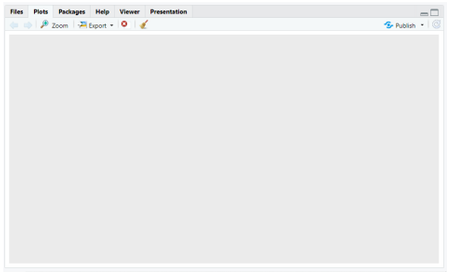


Figure 2.4.1: Data layer only

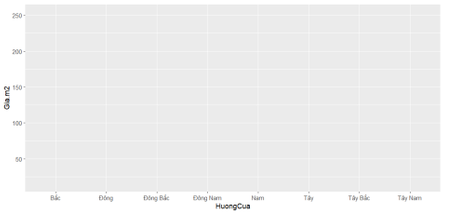



Figure 2.4.2: Data + Aesthetic layers

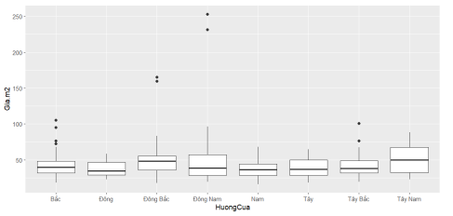



Figure 2.4.3: Data + Aesthetic + Geometry layers

## 2.5 Adding Colour

Adding colour enhances chart readability and visual appeal. There are several methods:

1. Direct color application:
   - Outline color: `geom_boxplot(colour = "red")` (Figure 2.5.4)
   - Fill color: `geom_boxplot(fill = "red")` (Figure 2.5.5)
   - Combined: `geom_boxplot(colour = "red", fill = "blue")` (Figure 2.5.6)
2. Dynamic coloring based on data: To color boxes by a variable (e.g., `HuongCua`), use:

```
ggplot(data = house_df, aes(x = HuongCua, y = GiaBan,
       fill = HuongCua)) + geom_boxplot()
```

   This assigns different colors to each unique value in `HuongCua` (Figure 2.5.7).

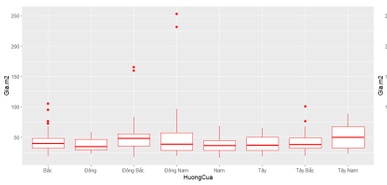


Figure 2.5.4: colour

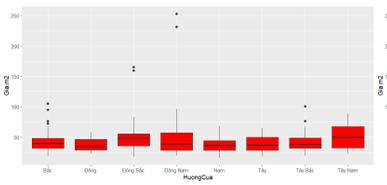


Figure 2.5.5: fill

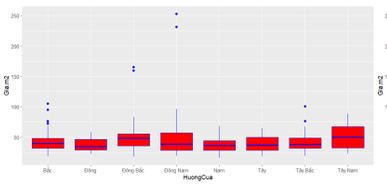


Figure 2.5.6: fill + colour

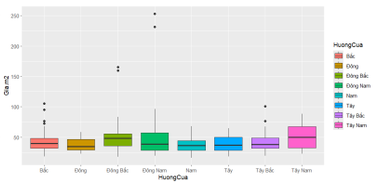



Figure 2.5.7: fill in aes

## 2.6 Scale

In `ggplot2`, **Scale** functions adjust the chart's aesthetic attributes. They can modify colors, fills, and other visual elements. For example:

- `scale_fill_brewer(palette = "Dark2")` changes box colors using a predefined palette.

- `scale_y_log10()` transforms the y-axis to a logarithmic scale, useful for visualizing unevenly distributed data.

## 2.7 Facet

**Facet** functions create multiple small charts from data subsets, facilitating comparison and analysis. The main types are:

- `facet_null()`: Default, creates a single chart.
- `facet_wrap()`: Arranges panels based on one variable.
- `facet_grid()`: Creates a panel grid based on two variables.

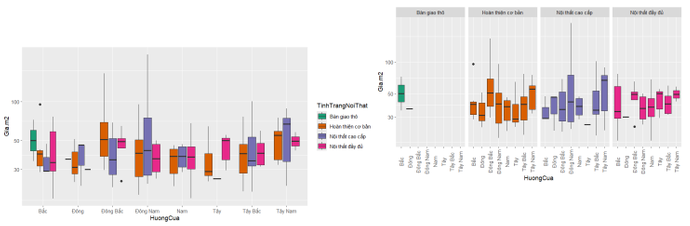


Figure 2.7.8: Base plot



Figure 2.7.9: `facet_grid(~var)`

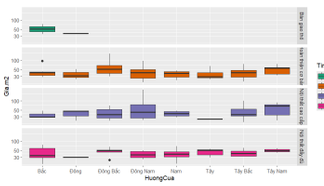


Figure 2.7.10: `facet_grid(var~)`

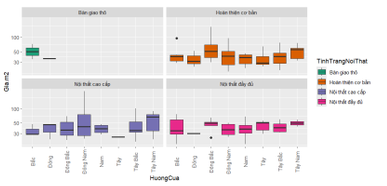


Figure 2.7.11: `facet_wrap(~var)`

## 2.8 Theme

**Theme** functions in `ggplot2` customize chart appearance:

- `labs()`: Sets or edits chart labels (title, axes, legend).
- `theme()`: Customizes chart style (text, color, size, layout of components).

# 3 Layers

- According to the definition, a layer is a collection of geometric components and statistical transformations. Geometric components, abbreviated as geom, represent what you actually see in the plot: points, lines, polygons, etc. Statistical transformations, abbreviated as stats, summarize the data. For example, categorizing and counting observations to create a plot or fitting a linear model.
- Generally, a layer serves three purposes:
  - To display the data.
  - To display a statistical summary of the data.

  - To add additional metadata: context, annotations, and references.

## 3.1 Individual Geoms

Geoms are the basic building blocks of ggplot2:

- They define the type of plot and coordinate axes.
- Without a geom, ggplot() only shows a blank background.
- Common geoms include:
  - geom_bar() for bar plots
  - geom_point() for scatter plots
  - Others like geom_title(), geom_text(), geom_line()

## 3.2 Collective Geoms

- In Individual Geoms, a separate graphical object is drawn for each observation (row) in the dataset. On the other hand, Collective Geoms display multiple observations with a single graphical object. This could result from a statistical summary (like a boxplot), or it could be a fundamental part of the geom's display (like a polygon).
- There are four main types of Collective Geoms:
  - Multiple groups, one aesthetic: Shows different groups using one property (e.g., color).
  - Different groups on different layers: Each group has its own layer.
  - Overriding default grouping: Organizes data in more meaningful ways.
  - Matching aesthetics to objects: Customizes how objects look (color, size, shape).

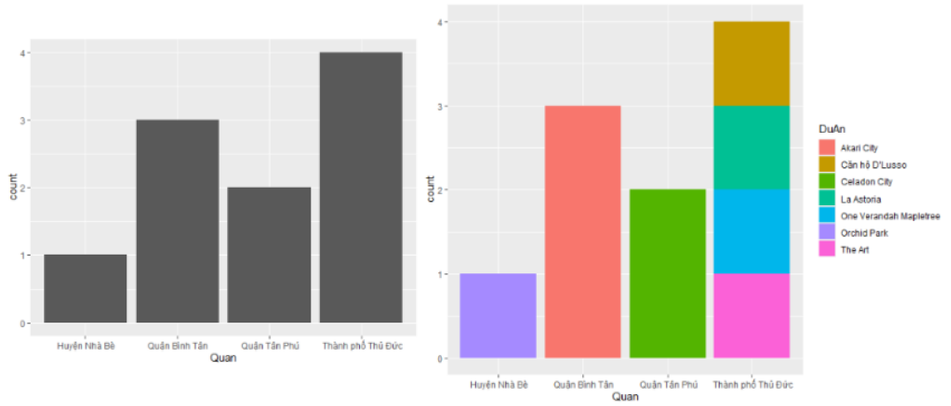



Figure 3.2.12: Examples of Collective Geoms

## 3.3 Statistical Summaries

Statistical summaries in ggplot2 help visualize data patterns, uncertainties, and complex relationships through various techniques and plot types.

- **Showing uncertainty:** Use confidence intervals or standard errors for experimental data.
- **Weighted data:** Adjust displays for data with varying importance or frequency.
- **Displaying distributions:** Use histograms, density plots, or boxplots for different data types.

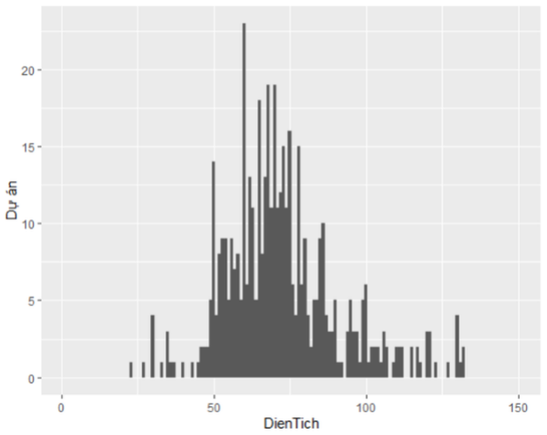



Figure 3.3.13: Examples of Histograms

- **Overplotting:** Address issues in scatter plots with large datasets where points overlap.
- **Summary statistics:** Show mean, median, or quartiles using various ggplot2 functions.
- **3D surfaces:** Use contours, color gradients, or bubble plots to represent 3D data in 2D.

## 3.4 Maps

- **Polygon maps:** In ggplot2, creating a Polygon Map begins with preparing geographic data, often in the form of shapefiles or GeoJSON.
- **Simple features maps:** The ggplot2 package supports overlaying one map on top of another by allowing multiple geom_sf() layers to be added to a plot. You can then add labels or text to identify different geographic features on the map.
- **Map projections:** The process of converting geographic data from a spherical (Earth) to a planar (map) representation, or vice versa, is the task of map projections.
- **Raster maps:** Raster data often consists of satellite images or environmental data like temperature or elevation.

## 3.5 Networks

Networks represent relationships between objects, where objects are called "nodes" and relationships are called "edges". Visualizing a network involves:

- **Setup:** Prepare data and configure ggplot for network drawing
- **Drawing nodes:** Use geom_node_ functions, typically geom_node_point()
- **Drawing edges:** Various options available to connect nodes
- **Faceting:** Useful for:
  - Observing network evolution over time
  - Quickly assessing grouping results using tidygraph algorithms

## 3.6 Annotations

Annotations provide metadata for plots, offering additional information about the displayed data. To represent annotations, the following components are involved:

- **Titles:** Plot and axis titles
- **Text labels:** Placed on specific data points or areas
- **Custom annotations:** Shapes, icons, or non-standard text for emphasizing elements
- **Direct labelling:** Labels attached directly to data points

- **Facet annotations:** Consistent labelling across faceted plots

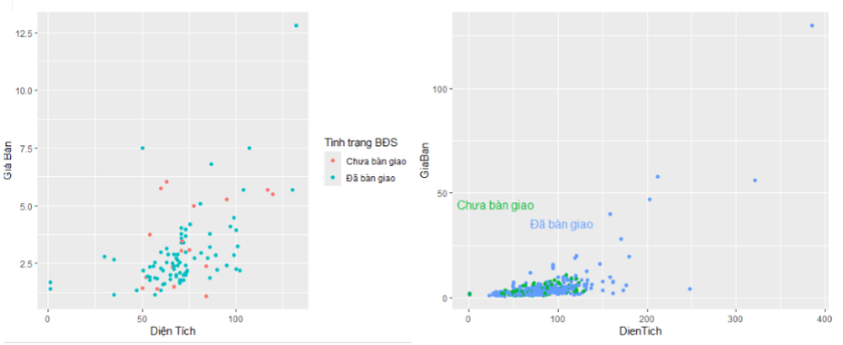



Figure 3.6.14: Examples of Annotations

## 3.7 Arranging Plots

- **Laying out plots side by side:** Arranging plots side by side prevents overlap and facilitates visual comparison between different datasets or different aspects of the same dataset.
- **Arranging plots on top of each other:** Stacking plots on top of each other allows for direct comparison of different data layers on the same coordinate axes.

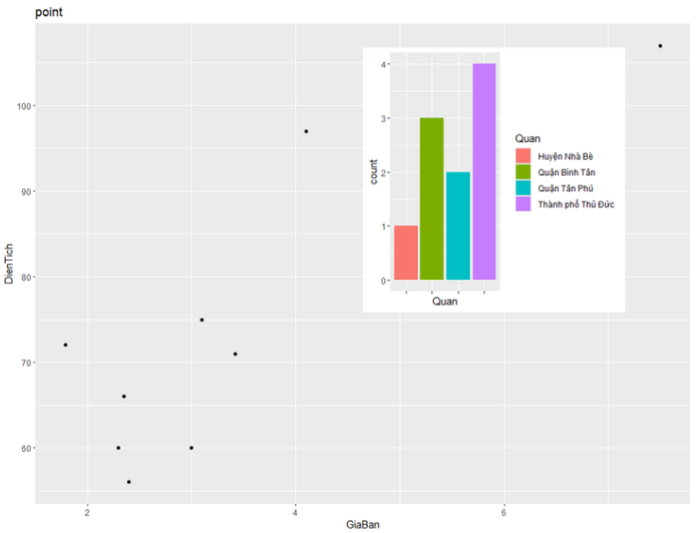



Figure 3.7.15: Examples of Arranging Plots

# 4 Scales in ggplot2

## 4.1 Overview

In ggplot2, scales are tools that control the mapping of data to aesthetic elements such as size, color, position, or shape on a plot. Scales not only transform data into visual elements but also help users understand and interpret the plot through axes and legends.

## 4.2 Types of Scales

Scales in ggplot2 are divided into three main groups:

- **Position and Axis Scales**: Control the positioning of objects on the plot, including:
  - **Numeric (Continuous) Position Scales**: Used with continuous numeric data.
  - **Date/Time Scales**: Used with time data.
  - **Discrete Position Scales**: Used with categorical or discrete data.
  - **Binned Position Scales**: Used when data is divided into intervals.
- **Color and Legend Scales**: Manage the mapping of data to color and adjust legends.
- **Scales for Other Aesthetic Elements**: Include size, shape, line width, line type, and transparency.

## 4.3 Position and Axis Scales

## a Numeric (Continuous) Position Scales

- **Limits:** Set the range of aesthetic values that the scales apply. Limits can be manually set to ensure consistency between plots.
- **Zooming In:** Reduce the default scale limits to zoom in on a part of the plot using `coord_cartesian()`.
- **Visual Range Expansion:** Default scales expand the range of the axes to prevent data from overlapping the axes, adjustable with `expand`.
- **Breaks and Minor Breaks:** Define the positions of tick marks on the axis and minor grid lines. Breaks can be manually set or removed.
- **Labels:** Labels can be changed using the `labels` argument.
- **Transformations:** Apply transformations like log, reverse, sqrt, etc., using the `trans` parameter.

Example:

```
base <- ggplot(data = house, aes(DienTich, GiaBan)) + geom_point()
base + scale_y_continuous(trans = "log10")
```

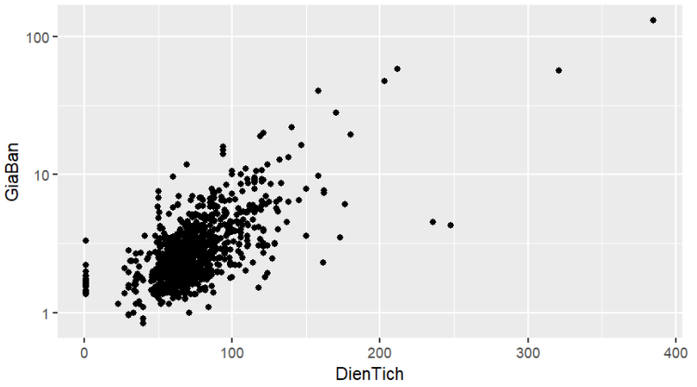



Figure 4.3.16: Example: Applying a logarithmic base-10 transformation to the y-axis.

## b Date/Time Scales

- Breaks for Date/Time `date_breaks` sets breaks according to time units such as years, months, weeks.
- Minor Breaks for Date/Time `date_minor_breaks` specifies minor breaks using time units.
- Labels for Date/Time Customize the label format using `date_labels`.

Example:

```
base <- ggplot(data, aes(x = NgayCapNhat, y = GiaBan))
+ geom_line(na.rm = TRUE) + labs(x = NULL, y = NULL)
lim <- as.Date(c("2023-01-01", "2023-06-31"))
base + scale_x_date(limits = lim, date_breaks = "3 month",
date_labels = "%m")
```

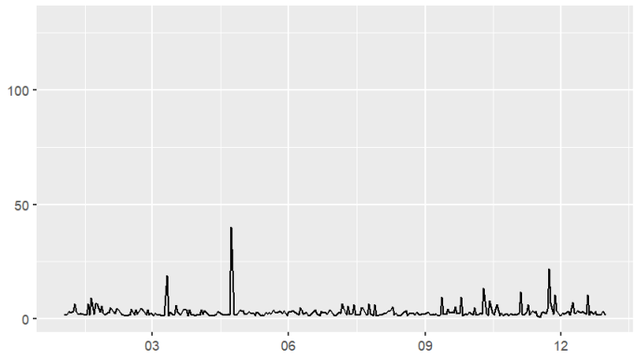



Figure 4.3.17: Example: Breaks and Labels for Date/Time Scales.

## c Discrete Scales

**Limits, Breaks, and Labels**: The limits of a discrete scale define the set of possible values, which can be set manually or automatically.

## d Binned Scales

Binned scales divide continuous data into intervals and transform them into discrete variables.

# 4.4 Color Scales and Legends

## a Continuous Color Scales

Map continuous data values to colors.

Examples: `scale_fill_viridis_c()` and `scale_fill_distiller()`.

`Example:`

```
df <- house %>% filter(!is.na(DienTich) & !is.na(Gia.m2))

df <- house %>%
  mutate(DienTich_bin = cut(DienTich, breaks = 10),
         GiaBan_bin = cut(GiaBan, breaks = 10))

ggplot(df, aes(x = DienTich_bin, y = GiaBan_bin)) +
  geom_tile(aes(fill = Gia.m2)) +
  scale_fill_gradient(low = "lightblue", high = "darkblue") +
  labs(title = "Heatmap of Price per Square Meter vs Area",
```

```
      x = "Area (Binned DienTich)",
      y = "Sale Price (Binned GiaBan)",
      fill = "Price per m² (Gia/m2)") +
theme_minimal() +
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

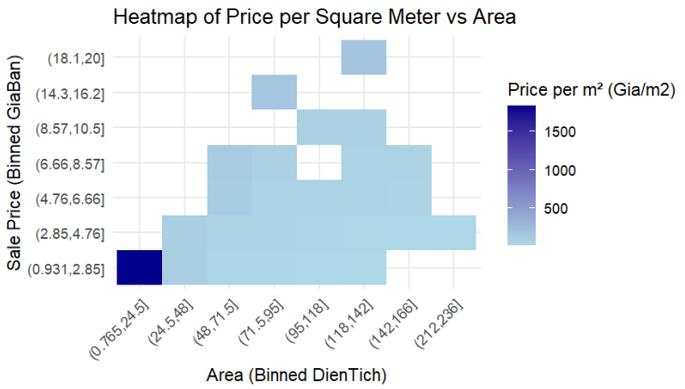



Figure 4.4.18: Example: Continuous Color Scale.

## b Discrete Color Scales

Map discrete data values to different colors.

Examples: `scale_fill_discrete()` or `scale_fill_hue()`.

## c Binned Color Scales

Divide data into intervals and assign each interval a specific color.

## 4.5 Other Scales

Besides position and color, ggplot2 also provides other scales:

- **Size scales**: Map data to point or text size on the plot.
- **Shape scales**: Map discrete values to different shapes.
- **Line Width scales**: Adjust the thickness of lines.
- **Line Type scales**: Map discrete values to different line types.
- **Manual scales**: Define values for each aesthetic element manually.
- **Identity scales**: Used when data values and aesthetic elements are mapped in the same system.

# Bibliography


[1] Wickham, H., Navarro, D., & Pedersen, T. L. (2023). ggplot2: Elegant Graphics for Data Analysis (3rd ed.). Springer-Verlag New York. https://ggplot2-book.org

[2] Data visualization with ggplot2:: Cheat Sheet. (n.d.). https://rstudio.github.io/cheatsheets/html/data-visualization.html

[3] Introduction to ggplot2. (n.d.). https://ggplot2.tidyverse.org/articles/ggplot2.html#scales

[4] R Programming 101. (2021, February 2). ggplot for plots and graphs. An introduction to data visualization using R programming [Video]. YouTube. https://www.youtube.com/watch?v=HPJn1CMvtmI&list=PLtL57Fdbwb_C6RS0JtBojTNOMVlgpeJkS