# R Basics

Tomasz Mądry

2025-07-09

## Overview

- Important R libraries
  - dplyr
  - ggplot2 + scales
  - stringr
- Exercises

## dplyr - Introduction

### Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

```r
mutate() # adds new variables that are functions of existing variables
select() # picks variables based on their names.
filter() # picks cases based on their values.
summarise() # reduces multiple values down to a single summary.
arrange() # changes the ordering of the rows.
```

These all combine naturally with group_by() which allows you to perform any operation "by group".

### More info

https://dplyr.tidyverse.org/

## dplyr - Data set preview

Load exemplary data set:

```
data(mtcars)
```

Preview the data:

```
head(mtcars, 6) # display first 6 rows
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

## dplyr - Data set preview

```r
tail(mtcars, 2) # display last 2 rows
```

```
##                mpg cyl disp  hp drat   wt qsec vs am gear carb
## Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
## Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

```r
nrow(mtcars)  # display number of rows
```

```
## [1] 32
```

```r
ncol(mtcars) # display number of cols
```

```
## [1] 11
```

## dplyr - Pipe %>%

Pipe (ctrl + shift + M):

```
%>% # magrittr pipe
```

Output from the left function is passed as input to right function:

```
mtcars %>% head()
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

## dplyr - Subset rows: filter()

Select rows meeting certain conditions.

Example: Cars with 6 cylinders

```
mtcars %>% filter(cyl == 6) %>% head(6)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

## dplyr - Choose columns: select()

Pick specific columns from the dataset.

Example: Select mpg, cyl, and hp columns

```r
mtcars %>% select(mpg, cyl, hp) %>% head(6)
```

```
##                    mpg cyl  hp
## Mazda RX4         21.0   6 110
## Mazda RX4 Wag     21.0   6 110
## Datsun 710        22.8   4  93
## Hornet 4 Drive    21.4   6 110
## Hornet Sportabout 18.7   8 175
## Valiant           18.1   6 105
```

## dplyr - Add/modify columns: mutate()

Create new columns or modify existing ones.

Example: Calculate fuel consumption in liters per 100 km

```r
mtcars %>%
  select(mpg, cyl) %>%
  mutate("l per 100km" = round((235.215 / mpg), 0)) %>%
  head(6)
```

```
##                    mpg cyl l per 100km
## Mazda RX4         21.0   6          11
## Mazda RX4 Wag     21.0   6          11
## Datsun 710        22.8   4          10
## Hornet 4 Drive    21.4   6          11
## Hornet Sportabout 18.7   8          13
## Valiant           18.1   6          13
```

## dplyr - Sort rows: arrange()

Order rows by column values ascending or descending.

Example: Sort cars by mpg descending

```r
mtcars %>%
  arrange(desc(mpg)) %>%
  head(6)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
```

## dplyr - Aggregate data by groups: group_by() + summarize()

Group data and compute summary statistics per group.

Example: Average mpg per cylinder count

```r
mtcars %>%
  group_by(cyl) %>%
  summarize(avg_mpg = mean(mpg),
            count = n())
```

```
## # A tibble: 3 x 3
##     cyl avg_mpg count
##   <dbl>   <dbl> <int>
## 1     4    26.7    11
## 2     6    19.7     7
## 3     8    15.1    14
```

## dplyr - Alternative for group_by() + summarize()

Grouping can be done within summarize() function.

Example: Average mpg per cylinder count

```
mtcars %>%
  summarize(.by = cyl,
            avg_mpg = mean(mpg),
            count = n())
```

```
##   cyl  avg_mpg count
## 1   6 19.74286     7
## 2   4 26.66364    11
## 3   8 15.10000    14
```

## dplyr - Remove duplicate rows distinct()

Returns unique rows from a data frame.

Example: Get unique combinations of cylinders and gears:

```
mtcars %>% select(cyl, gear) %>% distinct()
```

```
##                   cyl gear
## Mazda RX4           6    4
## Datsun 710          4    4
## Hornet 4 Drive      6    3
## Hornet Sportabout   8    3
## Toyota Corona       4    3
## Porsche 914-2       4    5
## Ford Pantera L      8    5
## Ferrari Dino        6    5
```

## dplyr - Rename columns `rename()`

Changes column names in a data frame.

Example: Rename `mpg` to `miles_per_gallon`

```r
mtcars %>%
  select(mpg, cyl, disp, hp) %>%
  rename(miles_per_gallon = mpg) %>%
  head(6)
```

```
##                   miles_per_gallon cyl disp  hp
## Mazda RX4                     21.0   6  160 110
## Mazda RX4 Wag                 21.0   6  160 110
## Datsun 710                    22.8   4  108  93
## Hornet 4 Drive                21.4   6  258 110
## Hornet Sportabout             18.7   8  360 175
## Valiant                       18.1   6  225 105
```

## dplyr - Rename columns pull()

Useful for quickly extracting a single column.

Example: Get the mpg column as a vector

```
mtcars %>%
  head(6) %>%
  pull(mpg)
```

```
## [1] 21.0 21.0 22.8 21.4 18.7 18.1
```

## dplyr - Perform operations row-by-row `rowwise()`

Applies functions across columns for each row.

Example: Calculate row-wise average of `mpg`, `hp`, and `wt`

```r
mtcars %>%
  head(2) %>%
  select(mpg, hp, wt) %>%
  rowwise() %>%
  mutate(avg_value = mean(c(mpg, hp, wt)))
```

```
## # A tibble: 2 x 4
## # Rowwise:
##     mpg    hp    wt avg_value
##   <dbl> <dbl> <dbl>     <dbl>
## 1    21   110  2.62      44.5
## 2    21   110  2.88      44.6
```

## dplyr - Conditional column creation if_else()

Vectorized conditional logic (used inside mutate()).

Example: Label cars as "high" or "low" mpg

```r
mtcars %>%
  tail(4) %>%
  select(mpg, hp) %>%
  mutate(mpg_type = if_else(mpg > 20, "high", "low"))
```

```
##                 mpg  hp mpg_type
## Ford Pantera L 15.8 264      low
## Ferrari Dino   19.7 175      low
## Maserati Bora  15.0 335      low
## Volvo 142E     21.4 109     high
```

## ggplot2 - Introduction

ggplot2 is a powerful and flexible R package for creating declarative, layered graphics. It is based on the Grammar of Graphics, which allows users to build plots step-by-step by adding components.

Key Features:

- Intuitive syntax using + to add layers
- Supports a wide range of plots: `scatter`, `bar`, `line`, `boxplot`, `histogram`, etc.
- Enables layered plotting (`data` + `geoms` + `stats` + `scales` + `themes`)
- Powerful faceting system for multi-panel plots
- Highly customizable aesthetics (`color`, `size`, `shape`, `labels`, etc.)

## ggplot2 - Core functions

- `ggplot()` – creates a plotting object
- `geom_*()` – add geometric objects (points, bars, lines)
- `aes()` – define aesthetic mappings (e.g., x, y, color)
- `facet_wrap()` / `facet_grid()` – split plots by variables
- `scale_*()` – control axes, colors, and legends
- `theme_*()` – customize non-data elements (background, font)

### More info

https://ggplot2.tidyverse.org/

## scales - Introduction

scales is an R package that provides tools to customize how data values are mapped to visual properties (like axes, colors, and labels) in plots—especially those created with ggplot2.

Key Features:

- Format numbers and dates for axis labels (e.g., currency, percentages, commas)
- Create and customize color palettes and gradients
- Apply transformations (log, reverse, square root) to scales
- Generate breaks and labels automatically or manually for axes
- Support for continuous, discrete, and date/time scales

## `scales` - Core functions

- Label functions: `label_dollar()`, `label_percent()`, `label_comma()`, etc. — format axis and legend labels
- Color scales: `scale_color_gradient()`, `scale_fill_brewer()`, etc. — define color mappings
- Transformations: `trans_new()`, built-in like `log_trans()`, `reverse_trans()` — modify scale behavior
- Breaks and limits: control where ticks and labels appear on axes

### More info

https://scales.r-lib.org/

## ggplot2 + scales: diamonds data set

A data set containing the prices and other attributes of almost 54,000 diamonds. Part of the ggplot2 package.

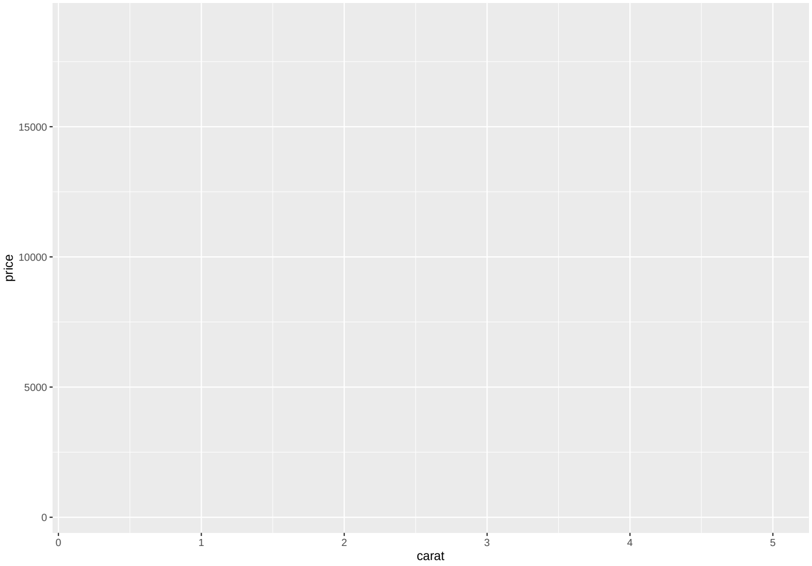
Key columns:

- price: price in US dollars
- carat: weight of the diamond
- cut, color, clarity: quality metrics
- x, y, z: dimensions

```
## # A tibble: 3 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good    E     VS1      56.9    65   327  4.05  4.07  2.31
```

## ggplot2 + scales: plot base

```r
ggplot(data = diamonds, aes(x = carat, y = price))
```



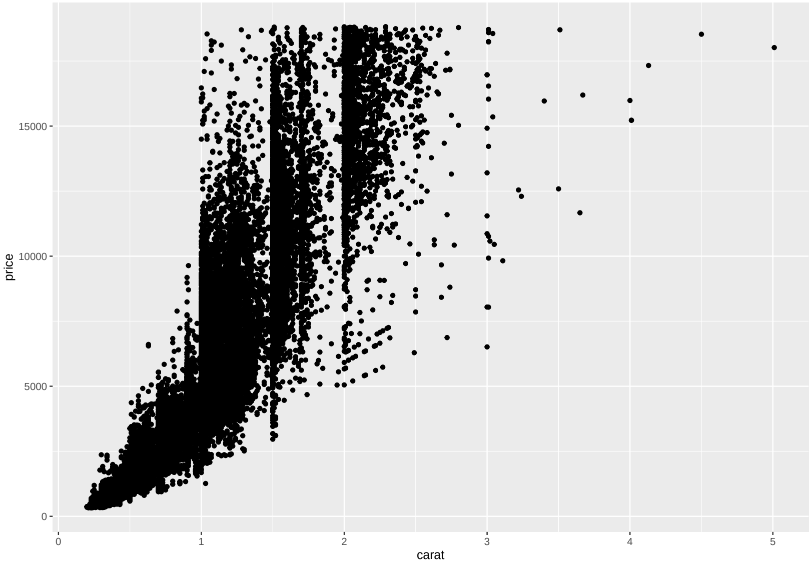
## ggplot2 + scales: geoms

Add layers to represent data.

- `geom_point()` – scatterplot
- `geom_bar()`, `geom_col()` – bar charts
- `geom_histogram()` – histogram
- `geom_boxplot()` – boxplot

## ggplot2 + scales: geom_point()

```r
ggplot(data = diamonds, aes(x = carat, y = price)) + geom_point()
```

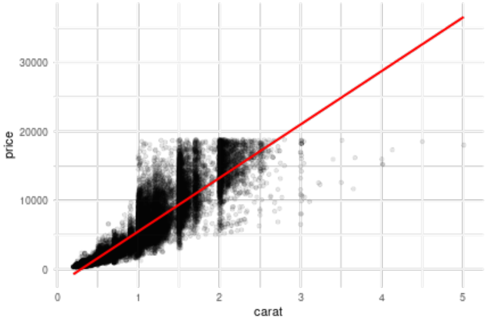


## ggplot2 + scales: Stats (stat_*)

Statistical transformations.

- `stat_smooth(method = "lm")` – regression line

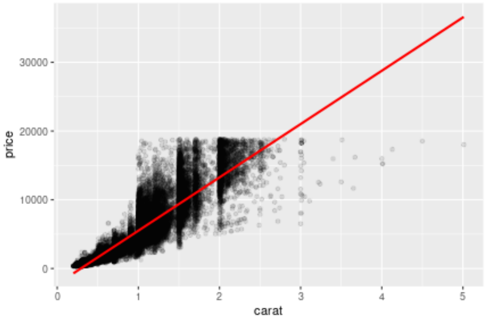
```r
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  stat_smooth(formula = y ~ x, method = "lm", col = "red") +
  theme_minimal() # select nice theme here
```



## ggplot2 + scales: Themes

Control non-data visuals (font, background, gridlines).

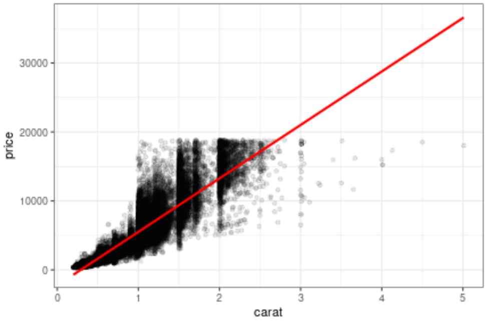
```
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  stat_smooth(formula = y ~ x, method = "lm", col = "red")
```



## ggplot2 + scales: Themes

Control non-data visuals (`font`, `background`, `gridlines`).

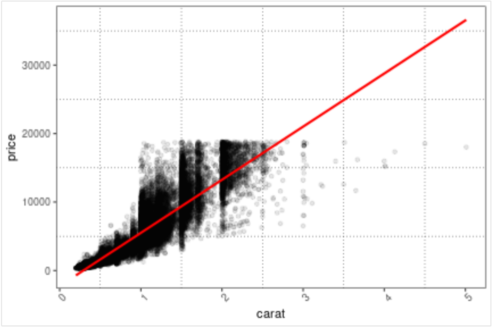
```r
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  stat_smooth(formula = y ~ x, method = "lm", col = "red") +
  theme_bw()
```



## ggplot2 + scales: Themes

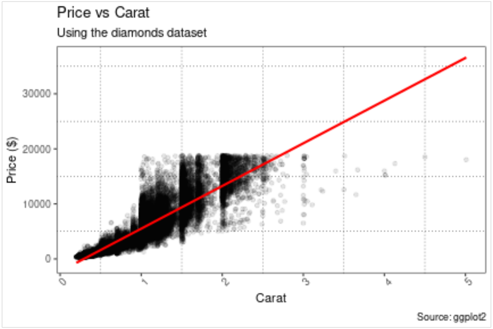
Control non-data visuals (`font`, `background`, `gridlines`).

```r
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.1) +
  stat_smooth(formula = y ~ x, method = "lm", col = "red") +
  theme_bw() +
  theme(plot.background = element_rect(fill = "transparent"),
        axis.text.x = element_text(angle = 45),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_line(linetype = 3, color = "black"))
```



## ggplot2 + scales: Labels and Titles

```
labs(title = "Price vs Carat",
     subtitle = "Using the diamonds dataset",
     x = "Carat", y = "Price ($)",
     caption = "Source: ggplot2")
```

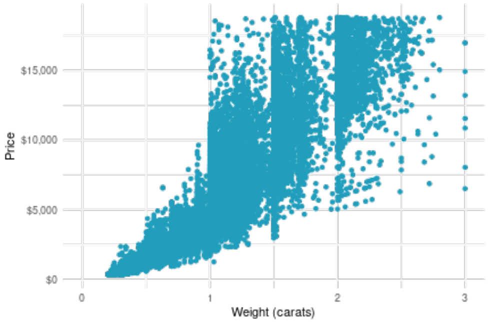


## ggplot2 + scales: Scales

Control axes and legends.

```
xlab("Weight (carats)") +
ylab("Price") +
scale_x_continuous(breaks = seq(0, 3, 0.5))
scale_y_continuous(labels = scales::label_dollar()) # or simply: scales::dollar
```

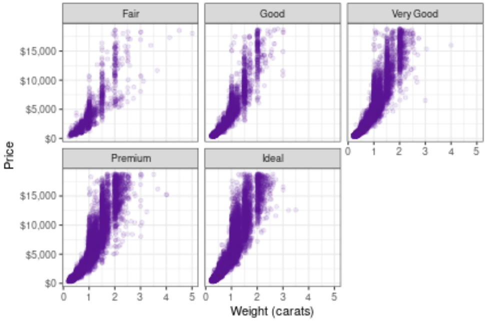
```
## Warning: Removed 32 rows containing missing values or values outside the sca
## (‘geom_point()‘).
```



## ggplot2 + scales: Facets

Split data into subplots.

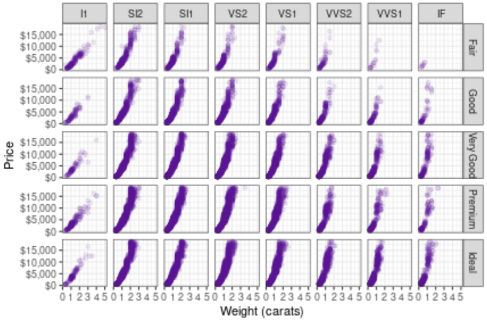
- facet_wrap(~cut)



## ggplot2 + scales: Facets

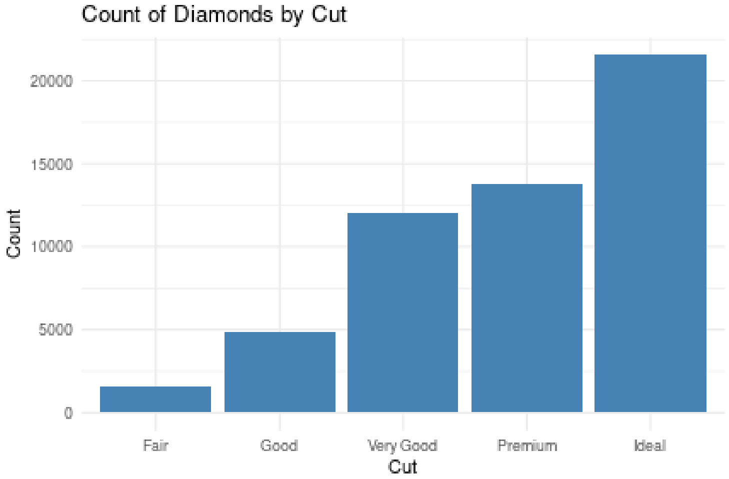
Split data into subplots.

- facet_grid(cut ~ clarity)



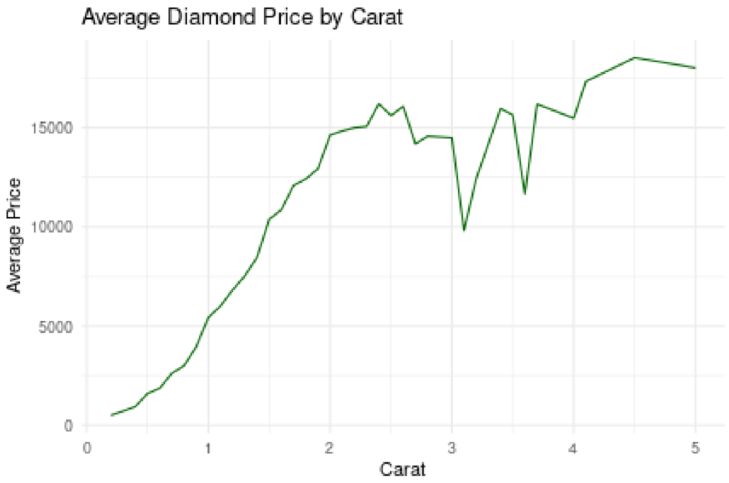
## Additional geometrics: `geom_bar`

Count of diamonds by cut.



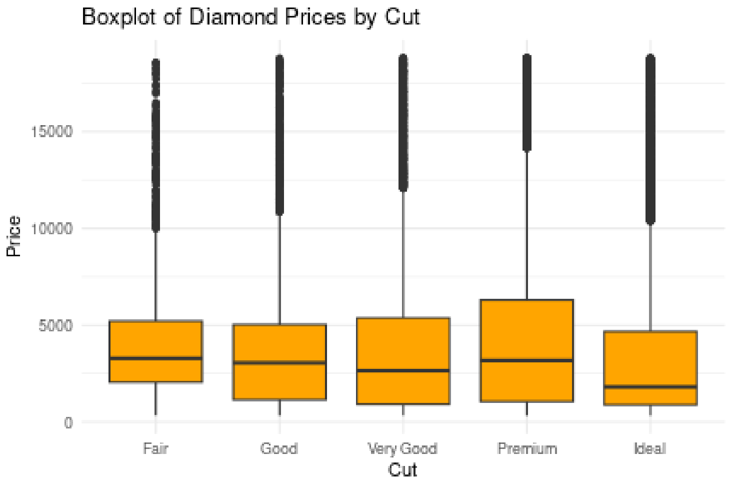
## Additional geometrics: `geom_line`

Average price by carat.



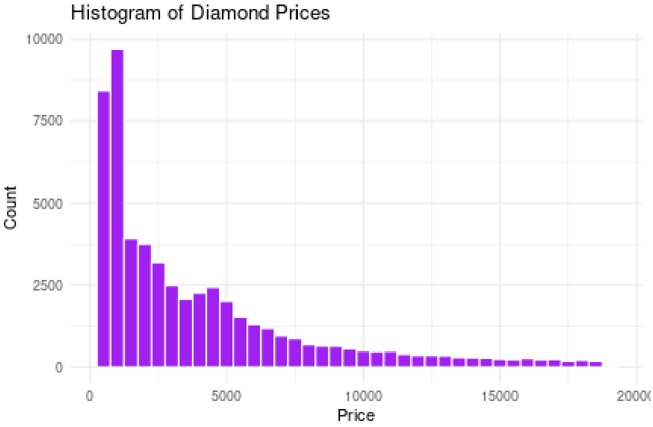
## Additional geometrics: `geom_boxplot`

Price distribution by cut.



## Additional geometrics: `geom_histogram`

Distribution of diamond prices.



## `stringr` - Introduction

`stringr` is an R package designed to make working with strings (character data) easier, consistent, and more user-friendly. It provides a cohesive set of functions that simplify common string manipulation tasks such as `detecting`, `extracting`, `replacing`, and `splitting` strings.

| Function | Purpose |
|---|---|
| `str_length()` | Returns the number of characters in a string |
| `str_sub()` | Extracts substrings based on position |
| `str_detect()` | Checks if a pattern exists in a string |
| `str_subset()` | Extracts strings that match a pattern |
| `str_replace()` | Replaces the first match of a pattern |
| `str_replace_all()` | Replaces all matches of a pattern |
| `str_split()` | Splits strings into pieces based on a pattern |
| `str_trim()` | Removes leading and trailing whitespace |

## stringr - Examples

```r
library(stringr)

text <- "The quick brown fox jumps over the lazy dog."

# Detect if 'fox' is in the string
str_detect(text, "fox")
```

```
## [1] TRUE
```

```r
# Extract the first 9 characters
str_sub(text, 1, 9)
```

```
## [1] "The quick"
```

```r
# Replace 'fox' with 'cat'
str_replace(text, "fox", "cat")
```

```
## [1] "The quick brown cat jumps over the lazy dog."
```

## stringr in real-life: modify gene_ids

```
df <- tibble(gene_id = c("ENSG00000000457.14", "ENSG00000000005.6", "ENSG000003
             biotype = c("protein_coding", "protein_coding", "lncRNA"))
df
```

```
## # A tibble: 3 x 2
##   gene_id            biotype
##   <chr>              <chr>
## 1 ENSG00000000457.14 protein_coding
## 2 ENSG00000000005.6  protein_coding
## 3 ENSG00000310545.1  lncRNA
```

```
df %>% mutate(gene_id = str_remove(gene_id, "\\.\\d+$"))
```

```
## # A tibble: 3 x 2
##   gene_id         biotype
##   <chr>           <chr>
## 1 ENSG00000000457 protein_coding
## 2 ENSG00000000005 protein_coding
## 3 ENSG00000310545 lncRNA
```

## stringr in real-life: modify biotypes

```r
df
```

```
## # A tibble: 3 x 2
##   gene_id            biotype
##   <chr>              <chr>
## 1 ENSG00000000457.14 protein_coding
## 2 ENSG00000000005.6  protein_coding
## 3 ENSG00000310545.1  lncRNA
```

```r
df %>% mutate(biotype = str_replace(biotype, "protein_coding", "pc"))
```

```
## # A tibble: 3 x 2
##   gene_id            biotype
##   <chr>              <chr>
## 1 ENSG00000000457.14 pc
## 2 ENSG00000000005.6  pc
## 3 ENSG00000310545.1  lncRNA
```

## Excercises

## Excercise 1:

### `iris` Dataset: Average Sepal Length by Species

**Dataset:** `iris`
**Task:**

- Use `dplyr` to compute the average sepal length per species.
- Use `ggplot2` to create a bar plot of the results.

**Hint:** Use `geom_bar(stat = "identity")` or `geom_col()`!

## Excercise 2:

### mtcars: Horsepower vs MPG by Cylinders

**Dataset:** `mtcars`
**Task:**

- Use `dplyr` to group by the number of cylinders (cyl) and compute the average mpg and hp.
- Create a scatter plot of `avg horsepower` vs `avg mpg`, colored by number of cylinders.

**Hint:** Use `factor(cyl)` for `color` aesthetics.

## Excercise 3:

### ToothGrowth: Effect of Dose on Tooth Length

**Dataset:** `ToothGrowth`
**Task:**

- Use `dplyr` to calculate mean tooth length for each dose.
- Create a line plot showing `tooth length` vs `dose`, colored by supplement type (`supp`).

**Hint:** Don't forget to `group_by` both: `dose` and `supp`!

## Excercise 4:

### `CO2`: Uptake Rate by Treatment

**Dataset:** `CO2`
**Task:**

- `Filter` for Type == "Quebec".
- Use `dplyr` to get average `uptake per treatment` and `concentration`.
- Use `ggplot2` to plot a line graph of uptake vs concentration for each treatment.

## Excercise 5:

### ChickWeight: Growth Over Time

**Dataset:** `ChickWeight`
**Task:**

- Use `dplyr` to filter for a few selected chicks (e.g., Chick %in% c(1, 5, 10)).
- Use `ggplot2` to create a line plot of `weight` over `time` per chick.

## Excercise 6:

### mtcars: Boxplot of MPG by Transmission

**Dataset:** mtcars
**Task:**

- Use dplyr to mutate am to a factor labeled “Automatic” and “Manual”.
- Create a boxplot of mpg by transmission type (am).

## Excercise 7:

### iris: Sepal Width Distribution

**Dataset:** iris
**Task:**

- Use ggplot2 to make a histogram of sepal width, filled by species.
- Use facet_wrap() to show separate plots per species.

## Excercise 8:

### `USArrests`: Violent Crime by Region

**Dataset:** `USArrests`
**Task:**

- Add a column for region (use `state.region` from base R, it matches row names of `USArrests`).
- Use `dplyr` to get average murder and assault rate per region.
- Create a bar plot of average assault rate per region.

**Hint:** To match regions use: `state.region[match(rownames(.), state.name)]`

## Excercise 9:

### `Titanic` Survival (from titanic package or datasets::Titanic)

**Dataset:** `as.data.frame(Titanic)`
**Task:**

- Use `dplyr` to calculate survival rate per class and sex.
- Use `ggplot2` to create a stacked bar plot showing survival by class and sex.