Lab 04 Visualization using R and ggplot2

RE 519 Data Analytics and Visualization | Autumn 2025

In this lab, we are focusing on produce nice diagram using `ggplot2`. We will go through the production process of three plots. Please try to understand each step, including the rational, functions, and arguments. The dataset we are using is **residential real estate rental listing data**, from RentHub and can be accessed through Dewey Data Platform (UW NetID required).

The due day for each lab can be found on the course wesbite. The submission should include Rmd, html, and any other files required to rerun the codes. From Lab 4, the use of any generative AI tool (ChatGPT, Copilot, etc.) is **allowed**. But, I still encourage you to write codes by yourself and use AI tools as a way to debug and explore new knowledge. More information about Academic Integrity and the Use of AI.

Lab 04: Visualization using R and ggplot2

GitHub Copilot (Optional)

Please allow me first introduce GitHub Copilot for RStudio, an AI coding assistant. GitHub Copilot is available as an opt-in integration with RStudio. Because of the rapid development of AI, tools like Copilot, OpenAI's GPT, Claude are becoming common in data science workflows. It has dramatically lowered the barrier to entry for data analysis and visualization across industries. As a student, we can use it freely!

1. Open GitHub Educationand click `Join GitHub Education`.
2. Log in your GitHub account.
3. Under "GitHub Education", click `Start an application`.
4. Complete the form, then click `Submit application`.
5. Once your application is approved, you will receive an Email.
6. Open `Edit – Settings` in RStudio, you can see `Copilot` on the left bar.
7. `Enable GitHub Copilot`, then `Sign in`.
8. Authorize RStudio to access your GitHub account.

You can also use other AI services, such as Gemini from Google, ChatGPT from OpenAI, or Copilot from Microsoft. Note: Microsoft Copilot with commercial data protection is enabled for UW faculty, staff, and students.

Data Preparation

RentHub provides us residential real estate rental listing data, including pricing, property information, location, and more from 2014 to present. The whole dataset is huge (over 100 GB). Therefore, we only select all data for the State of Washington during 2019-01-01 to 2024-12-31. Note: the time refers to when RentHub collected the data, rather than when the listing was originally posted. You can access the whole data through Dewey Data Platform (UW NetID required).

```
# load necessary packages
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
```

```
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.2     ✔ tibble    3.2.1
## ✔ lubridate 1.9.4     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ─────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become erro
```

```
library(ggplot2)
library(skimr)
```

Because the dataset is large and cannot be stored in GitHub, please download it through Dropbox and put the `rental_data_wa.csv` file within your lab 4 folder.

```
# load the rental listing data, it may take a while because its size
rental <- read_csv("rental_data_wa.csv")
```

```
## Rows: 3828059 Columns: 25
## ── Column specification ────────────────────────────────────────────────────
## Delimiter: ","
## chr  (14): ADDRESS, BUILDING_TYPE, CITY, CLUBHOUSE, COMPANY, DOORMAN, FURNIS...
## dbl   (9): BATHS, BEDS, GARAGE_COUNT, ID, LATITUDE, LONGITUDE, RENT_PRICE, S...
## dttm  (2): DATE_POSTED, SCRAPED_TIMESTAMP
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We filter the data to include listings posted between January 1, 2019, and December 31, 2023. There are 2,268,590 observations (rows) and 25 variables (columns) in total. The dataset is already tidy, so no further transformation is required. You can take a look at the meaning of each column.

```
rental <- rental %>%
  filter(DATE_POSTED >= as.Date("2019-01-01") &
         DATE_POSTED <= as.Date("2023-12-31"))
# check the data dimension
dim(rental)
```

```
## [1] 2268590      25
```

▶ **Click to Read Data Dictionary**

```
# skim the whole dataset
# skim(rental)
```

Visualizing Time: An Example

Visualizing time is particularly valuable and interpretable in the context of rental markets, as it helps reveal market cycles, spatial heterogeneity, seasonal fluctuations, and structural shifts. We will use this section to produce a nice plot.

Thinking about the Message

Between 2019 and 2024, one of the most significant global events was the COVID-19 pandemic, which profoundly affected housing and rental markets. In this analysis, we are interested in **rent resiliency** across different groups after the pandemic — that is, how quickly and to what extent rents in various apartment types and quality levels recovered following the COVID-19 shock.

First, we filter the rental dataset to apartment buildings only and add four useful variables:

- `month` : the posting month of each listing, for time-series analysis
- `bed_group` : bedroom categories, for comparing rent trends by unit size
- `amenities_sum` : the total number of key amenities
- `quality_group` : as the indicator of the quality level based on the values of `amenities_sum`

```
rental_apt <- rental %>%
  filter(BUILDING_TYPE == "apartment building") %>%
  mutate(
    # converts posting date into the first day of its month for monthly analysis
    month = as.Date(paste0(format(as.Date(DATE_POSTED), "%Y-%m"), "-01")),
    # categorizes apartments by the number of bedrooms
    bed_group = case_when(
      BEDS == 0 ~ "Studio",
      BEDS == 1 ~ "1 Bedroom",
      BEDS == 2 ~ "2 Bedrooms",
      BEDS >= 3 ~ "3+ Bedrooms",
      TRUE ~ "Unknown"),
```

```
    # creates a variable counting how many of six amenities each listing offers
    amenities_sum = (
      (CLUBHOUSE == "Y") +
      (GRANITE == "Y") +
      (STAINLESS == "Y") +
      (POOL == "Y") +
      (GYM == "Y") +
      (DOORMAN == "Y")),
    # groups apartments into Low / Medium / High quality levels
     quality_group = case_when(
      amenities_sum <= 1 ~ "Low Quality",
      amenities_sum %in% 2:4 ~ "Medium Quality",
      amenities_sum >= 5 ~ "High Quality",
      TRUE ~ "Unknown"
    ),
    # convert two variables from categorical to ordinal
    bed_group = factor(bed_group,
                       levels = c("Studio", "1 Bedroom", "2 Bedrooms", "3+ Bedrooms")),
    quality_group = factor(quality_group,
                           levels = c("Low Quality", "Medium Quality", "High Quality"))
  )
```

Then, let's calculate **average monthly rent** broken down by quality level and bedroom category. Notes: It is important to note that decisions about grouping, variable construction, and filtering should ideally be informed by exploratory data analysis and domain knowledge. For simplicity and clarity, these exploratory steps are not shown in this tutorial.

```
apt_rent_by_month <- rental_apt %>%
  # group by three variables: month, quality_group, bed_group
  group_by(month, quality_group, bed_group) %>%
  dplyr::summarise(mean_rent = mean(RENT_PRICE, na.rm = TRUE), .groups = "keep")
```

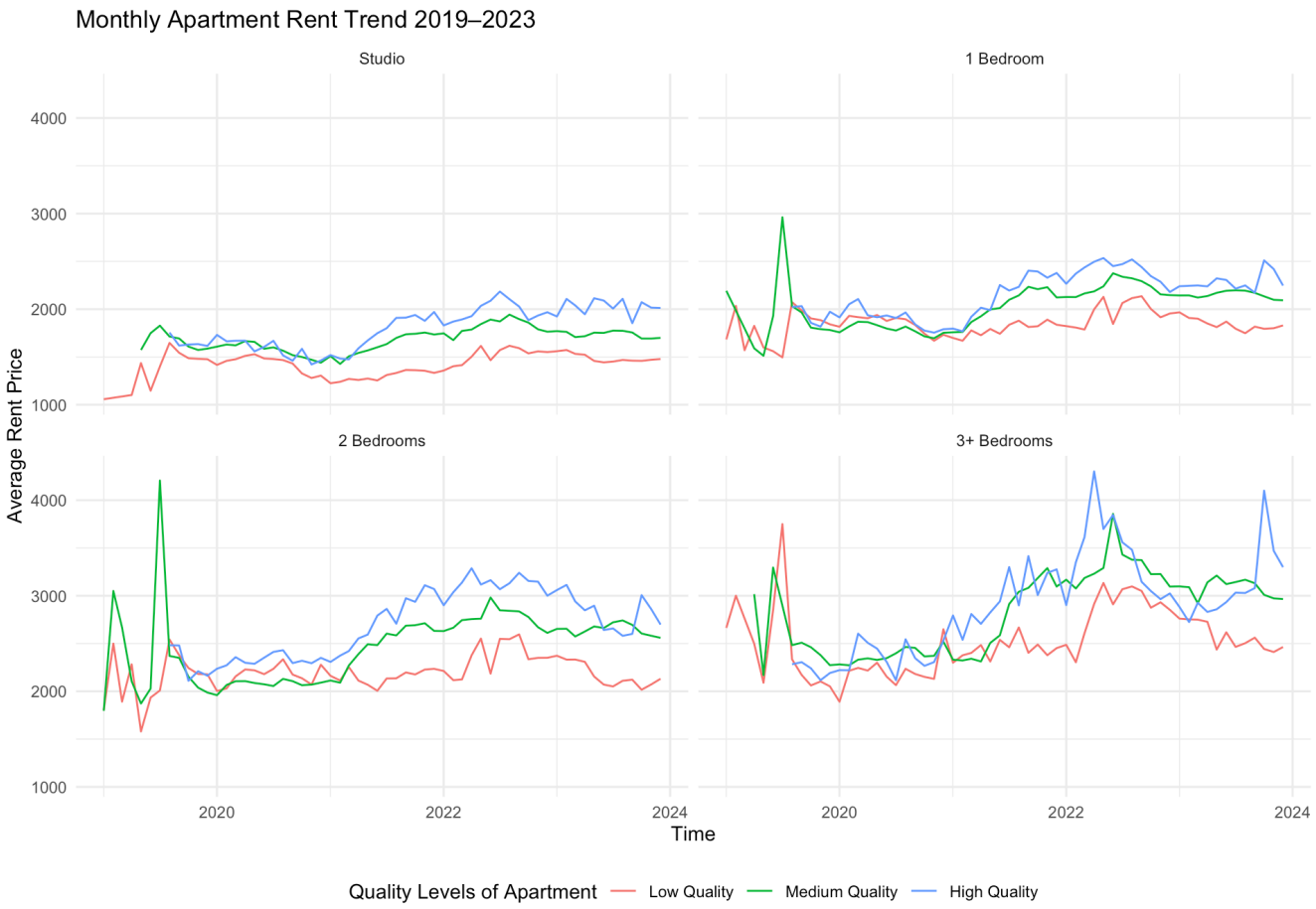
Let's produce the first plot about monthly rent trends for apartment buildings, separated by bedroom type and apartment quality. We have **four variables**: time, average rent, bedroom group, apartment quality. We need to determine the aesthetics (channels/encoding variables).

Recall the *Expressiveness + Effectiveness* we discussed: positions are always the best, so we put two important variables (time, average rent) as x and y.

Apartment quality and bedroom group are ordinal categorical variables. We need to think about whether we want to preserve this order. If preserving order is not essential for our analytically purpose, we may treat them simply as nominal categories. In the current design, apartment quality (`quality_group`) is represented by color, while bedroom type (`bed_group`)

is shown through faceting.

```
ggplot(apt_rent_by_month, aes(x = month, y = mean_rent, color = quality_group)) +
  geom_line() +
  facet_wrap(~bed_group)+
  labs(
    title = "Monthly Apartment Rent Trend 2019–2023",
    x = "Time",
    y = "Average Rent Price",
    color = "Quality Levels of Apartment"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```

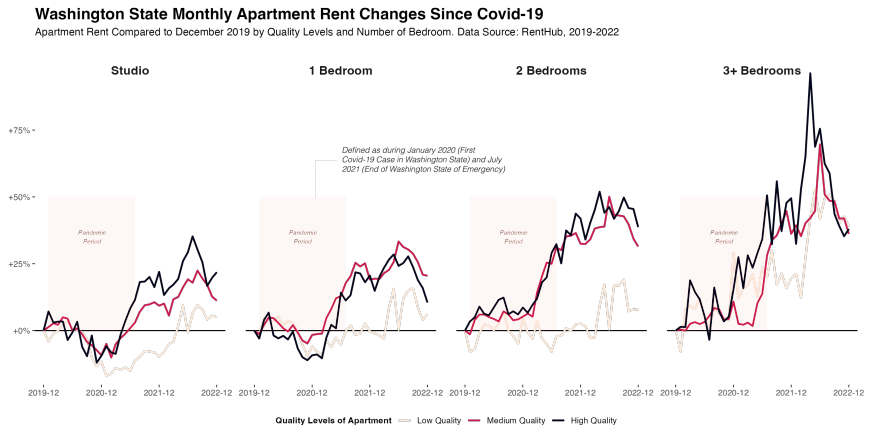


In general, the figure looks quite good after some adjustments if our goal is simply to show overall rent trends. However, the key message I'd like to emphasize is the **rent resiliency** across different groups after the pandemic—that is, how quickly and strongly rents in various apartment types and quality levels recovered following the COVID-19 shock. The absolute rent levels are not particularly meaningful on their own, so we can make an adjustment by converting them into **relative rents**, measured as rent changes compared to December 2019

levels (pre-pandemic).

```
apt_rent_rel <- apt_rent_by_month %>%
  # compute relative change within each group
  group_by(quality_group, bed_group) %>%
  mutate(
    # rent in Dec 2019 as baseline
    base_rent = mean_rent[month == as.Date("2019-12-01")],
    # rent change (e.g., +10%)
    rent_change = (mean_rent - base_rent)/base_rent
  ) %>%
  ungroup()
```

We are going to create a relatively complicated plot and you can have a look of the final result below. To achieve that, we need many functions in `ggplot2` and other software, such PowerPoint and PhotoShop, to do final edits (like the notes inside).



Washington State Monthly Apartment Rent Changes Since Covid-19

Basic Stucture

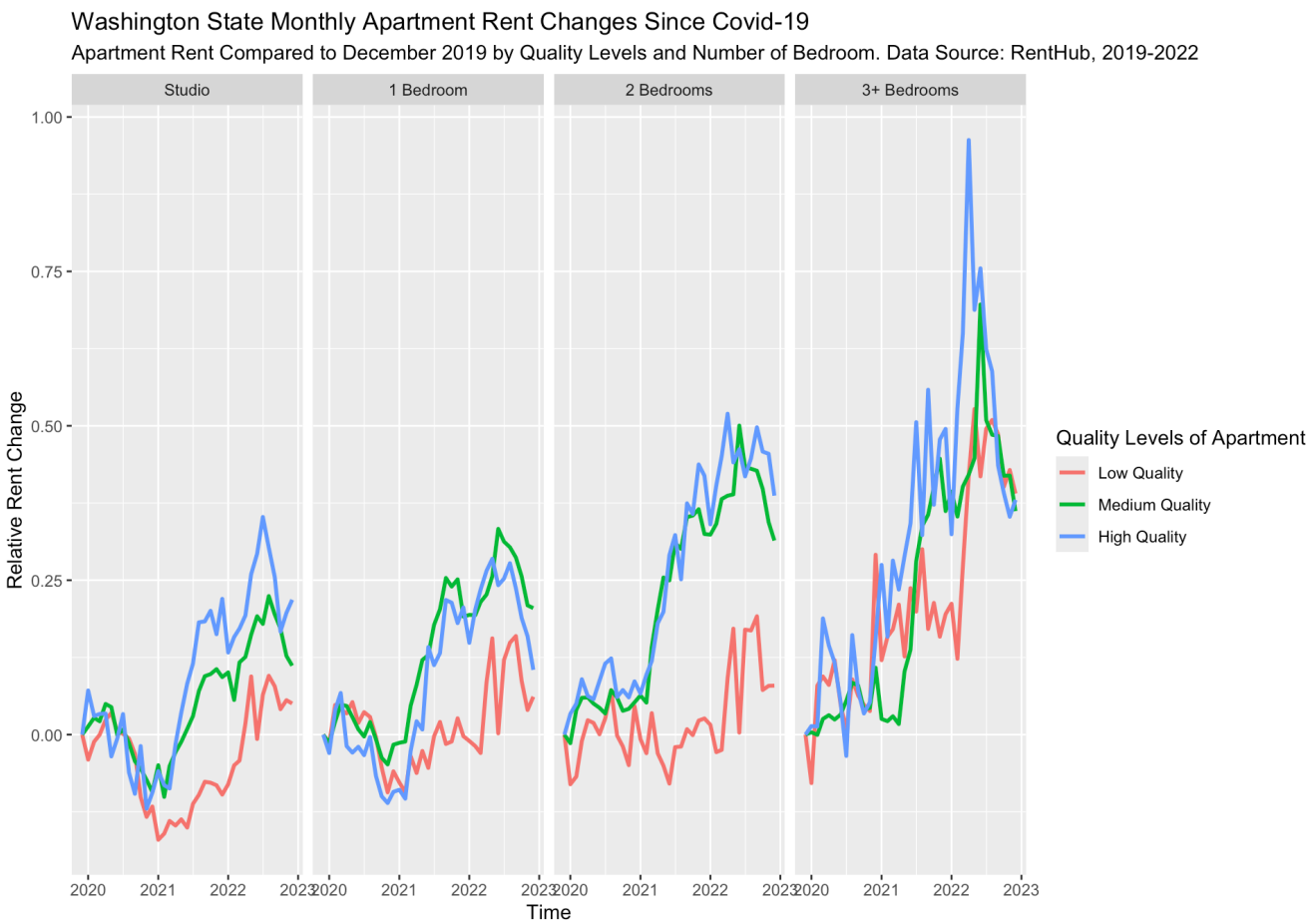
Similar as previous plot. We started from the **data**: `apt_rent_rel`. We can do filter here to make sure the plot will only show the time period we want: between **December 2019** (just before the first Covid-19 case in Washington State) and **December 2022**.

The we need to determine the **aesthetics** (channels/encoding variables), we map time to the x-axis, rent changes to the y-axis. There are two additional dimensions in the data: apartment quality (quality_group), which is represented by color, and bedroom type (bed_group), which is shown through **faceting**.

After that, we are going to add the first **geometry**: `geom_line`. Its location is decided by the global `aes` we already defined.

We also add the **labels**, including title, subtitle, x and y axis title, and the label for color group.

```
g <- ggplot(
  filter(apt_rent_rel, month >= as.Date("2019-12-01") & month <= as.Date("2022-12-01")),
  aes(x = month, y = rent_change, color = quality_group)
) +
  geom_line(linewidth = 1) +
  facet_wrap(~bed_group, nrow = 1) +
  labs(
    title = "Washington State Monthly Apartment Rent Changes Since Covid-19",
    subtitle = "Apartment Rent Compared to December 2019 by Quality Levels and Number of Bedroom. Data
    x = "Time",
    y = "Relative Rent Change",
    color = "Quality Levels of Apartment"
  )
g
```

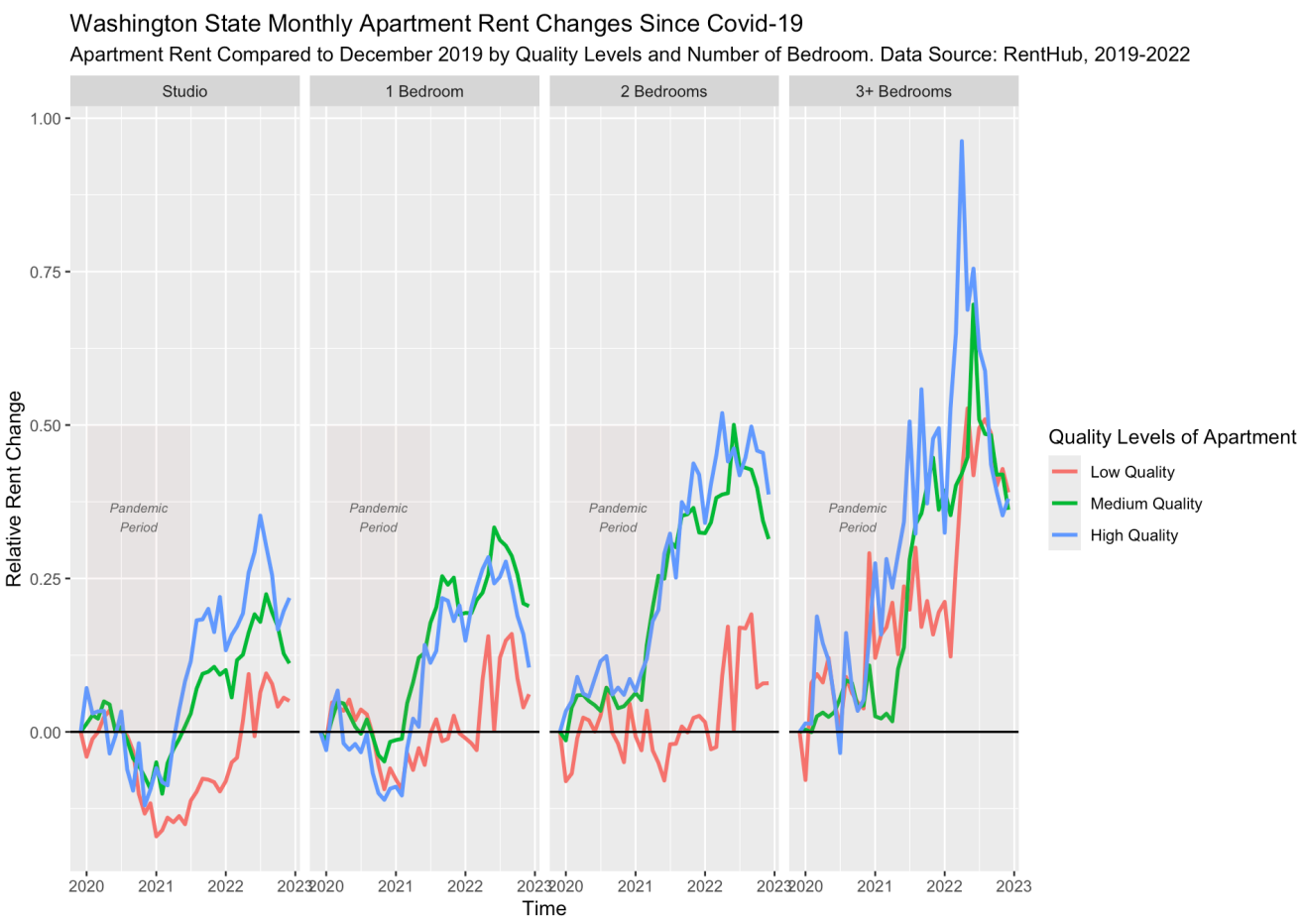


Graphical Primitives

We would like emphasize the line $y = 0$ because it means no change compared to the beginning of the pandemic. Meanwhile, we need to let readers know the time period of the pandemic. So, we use `geom_hline()` to add a horizontal line as another layer. (Remember to check Cheatsheet, R documentation, and using `?function` when you are not familiar with a function.)

There is another way we can simply draw a line is adding an **annotation** layer. But unlike a typical `geom` function, the properties of the `geoms` are not mapped from variables of a data frame, but are instead passed in as vectors. This is useful for adding small annotations (such as text labels or a line). We add a rectangular annotation to highlight a time period and text label with the rectangular.

```
g_line_rec <- g +
  # add a horizontal line across the plot at y = 0
  geom_hline(yintercept = 0, linewidth = 0.6) +
  annotate("rect", # a rectangular
           # we need 4 points to determine the location
           xmin = as.Date("2020-01-01"), xmax = as.Date("2021-07-01"),
           ymin = 0, ymax = 0.5,
           fill = "#e37d5a", alpha = 0.05) + # explain below
  annotate("text", # some texts
           # we need 1 points to determine the location
           x = as.Date("2020-10-01"),
           y = 0.35,
           label = "Pandemic\nPeriod",
           color = "gray40",
           size = 2.5,
           fontface = "italic") # explain below
g_line_rec
```



Fontface

`fontface` / `face` is a catch-all argument that describes the style of the typeface to use. "plain" is an upright normal-weight font, "italic" is a slanted normal-weight font, "bold" is an upright bold-weight font, and "bold.italic" is a slanted bold-weight font.

Colors

We can use R built-in colors (600+), you can check their names using `colors()`.

We can also use *Hex colors* and *palette functions*. Hex colors are written as `#RRGGBB`, where each pair (RR, GG, BB) represents the intensity of Red, Green, and Blue in hexadecimal (from 00 to FF, i.e., 0–255). ggplot2 provides built-in palette functions that automatically generate harmonious color sets for categorical or continuous variables.

R also supports numeric color systems like RGB, HCL, and HSV.

```
# colors()
# scale_color_brewer(palette = "Set2")
```

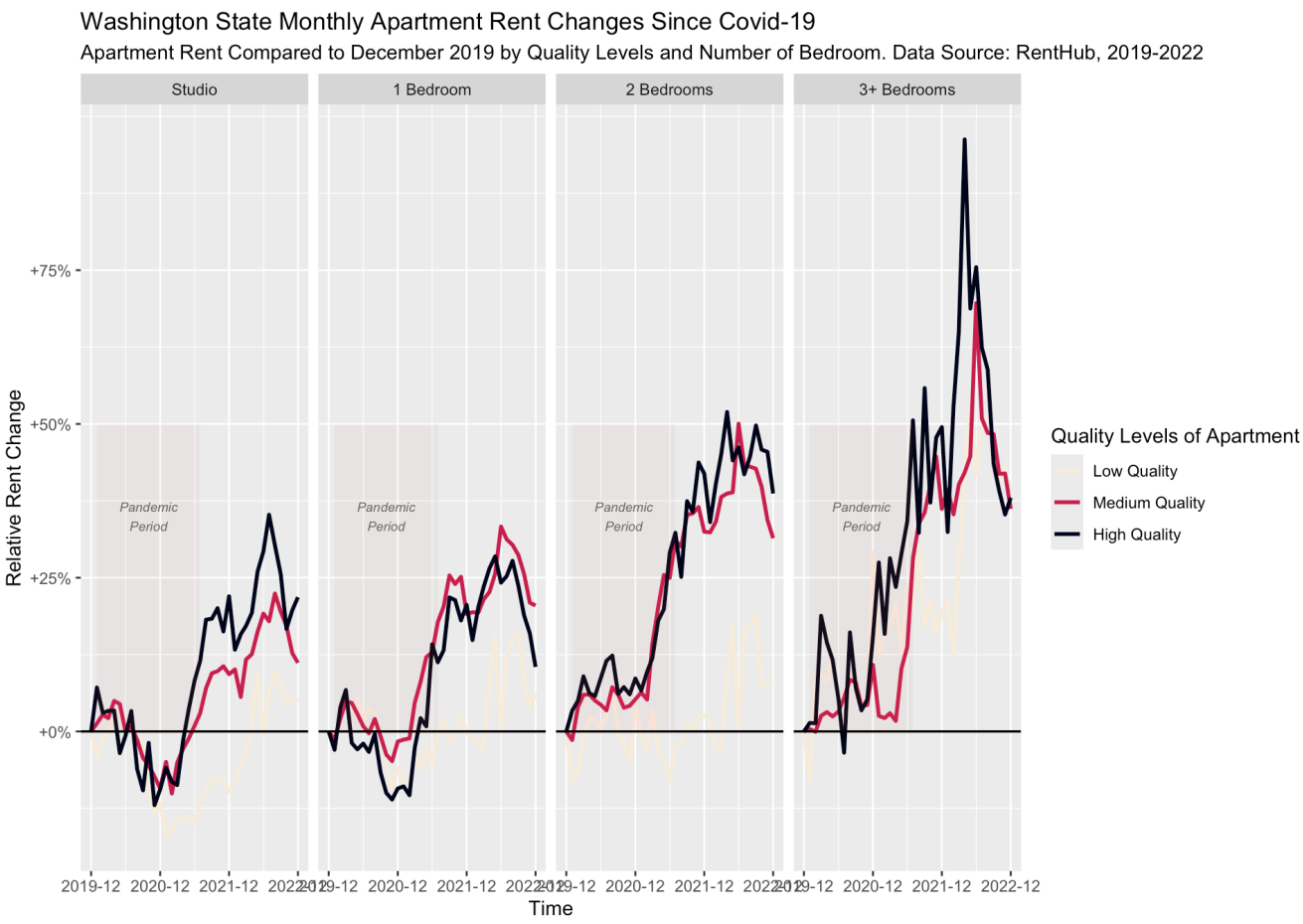
```
# scale_color_viridis_d(option = "plasma")
```

Scale

Scale controls **how** data values are mapped to aesthetics (channels/encoding variables). For example, we use `color = quality_group` and by default `ggplot2` will choose some colors for that. But we can change how color encodes through `scale_color_*`.

Aesthetic	Scale function	Controls
x-axis	`scale_x_*()`	axis range, tick marks, labels
y-axis	`scale_y_*()`	axis range, tick marks, labels
color	`scale_color_*()`	how color encodes categories or values
fill	`scale_fill_*()`	fill colors for bars, areas, etc.
size	`scale_size_*()`	point or line thickness
shape	`scale_shape_*()`	point shapes
alpha	`scale_alpha_*()`	transparency

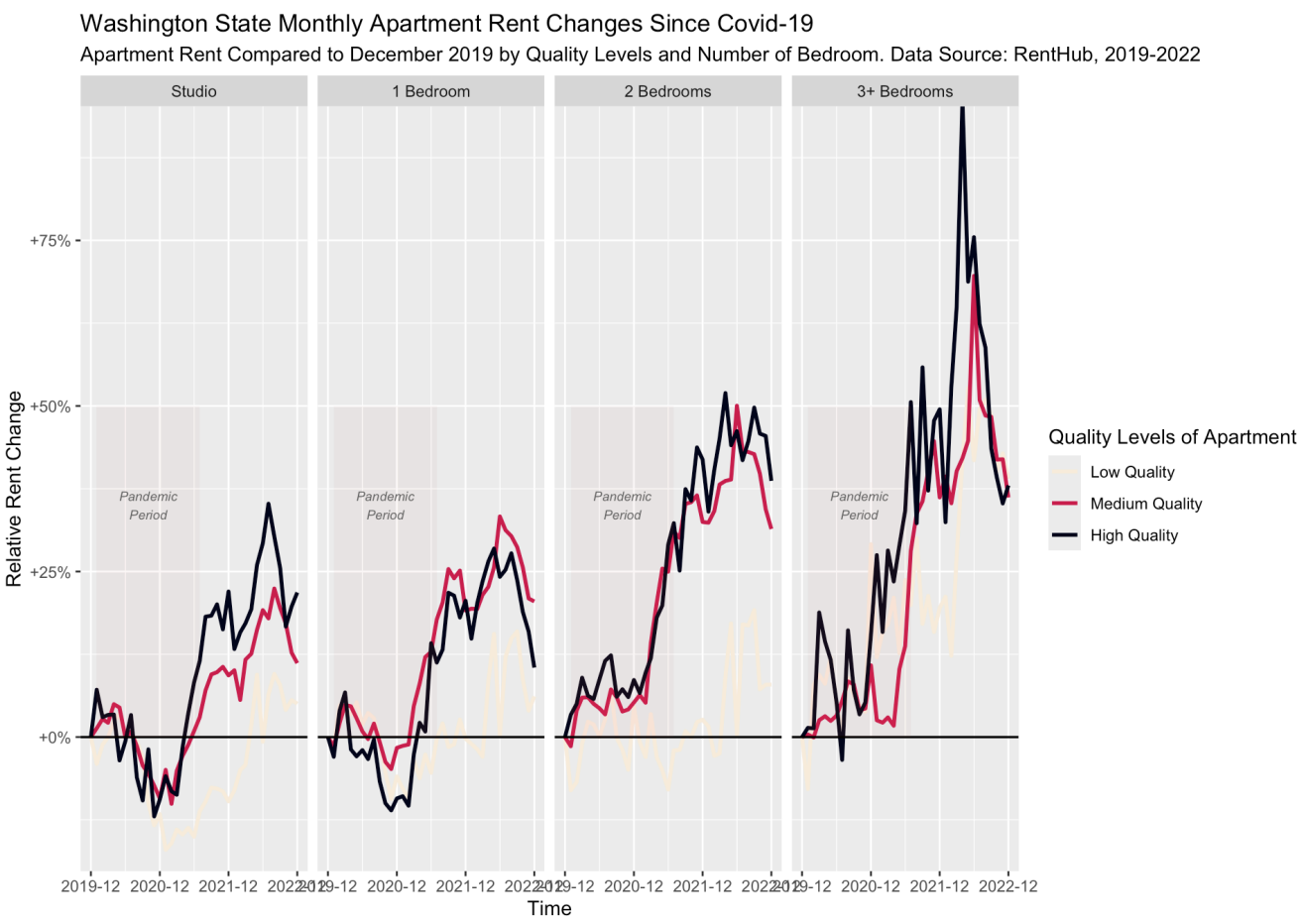
```
g_scale <- g_line_rec +
  # change how assigns colors to the variable quality_group
  # use rocket palette style within viridis family
  # direction = -1 reverses the order of the palette
  scale_color_viridis_d(option = "rocket", direction = -1) +
  # where to place tick marks on the x-axis
  scale_x_date(
    breaks = as.Date(c("2019-12-01", "2020-12-01", "2021-12-01","2022-12-01")),
    labels = c("2019-12", "2020-12", "2021-12", "2022-12")
  ) +
  # formats the numeric values into percentages with a plus or minus sign
  scale_y_continuous(
    breaks = c(0, 0.25, 0.5, 0.75),
    labels = function(x) sprintf("%+d%%", x * 100)
  )
g_scale
```



Coordinate System

The coordinate system determines how data coordinates are translated to positions on the plot. We will talk about this more in the future class on mapping and geospatial data.

```
g_coordinate <- g_scale +
  # limits the y-axis to between -0.15 and 0.9
  coord_cartesian(ylim = c(-0.15, 0.9))
g_coordinate
```

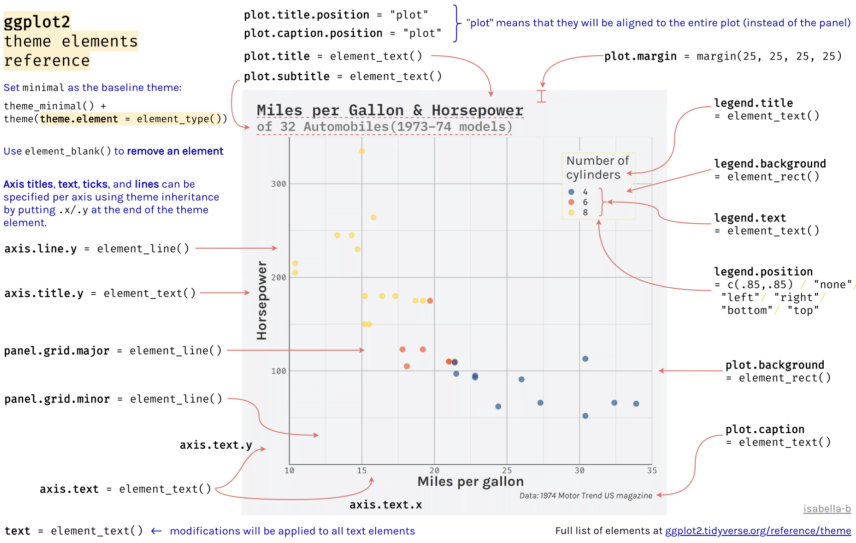


Theme

Theme controls all non-data visual components, text, background, grid lines, margins, etc.

Components

First, we need to know how `ggplot2` names those components:



Theme Elements Reference. Source: Reproducible Science for Busy Researchers by Dr. Andrew P. Lapointe.

You can see there is a hierarchy among those plot components. We list some here as the example, you can read all components of a theme from its official document.

- For the whole plot
  - `element_text()` : text, plot.title, plot.subtitle, plot.caption
  - `element_rect()` : plot.background, plot.margin
- For x and y axis
  - `element_text()` : axis.title.x/y, axis.text.x/y
  - `element_line()` : axis.line.x/y, axis.ticks.x/y
- For grid and panel
  - `element_line()` : panel.grid; panel.grid.major, panel.grid.minor
  - `element_rect()` : panel.border
  - `unit()` : panel.spacing
- For the legend
  - `element_text()` : legend.title, legend.text
  - `location using character/vector` : legend.position, legend.justification, legend.margin
- For facet
  - `element_text()` : strip.text, strip.text.x, strip.text.y
  - `element_rect()` : strip.background

Elements

There are several types of components, `ggplot2` calls them elements, such as text, line, and rectangular. For example, the line of axis is `element_line` while the title of axis is `element_text`. We remove some components through assigning them as `element_blank()`.

We change the attributes of those components (such as legend.title, axis.line) by assigning them an appropriate `element_*()` function inside `theme()`. Please refer to its official document or `?element_*()` for detailed arguments - Theme elements.

Complete Themes

Beyond those specific theme configuration, we can start with existing complete themes by `+ theme_*()`. The one I like most is `theme_minimal()` because it is super clean. Within `theme_*()`, we can set a few argument for the whole plot: `base_family` for default font, `base_size` for default font size, for example.

Theme Inheritance

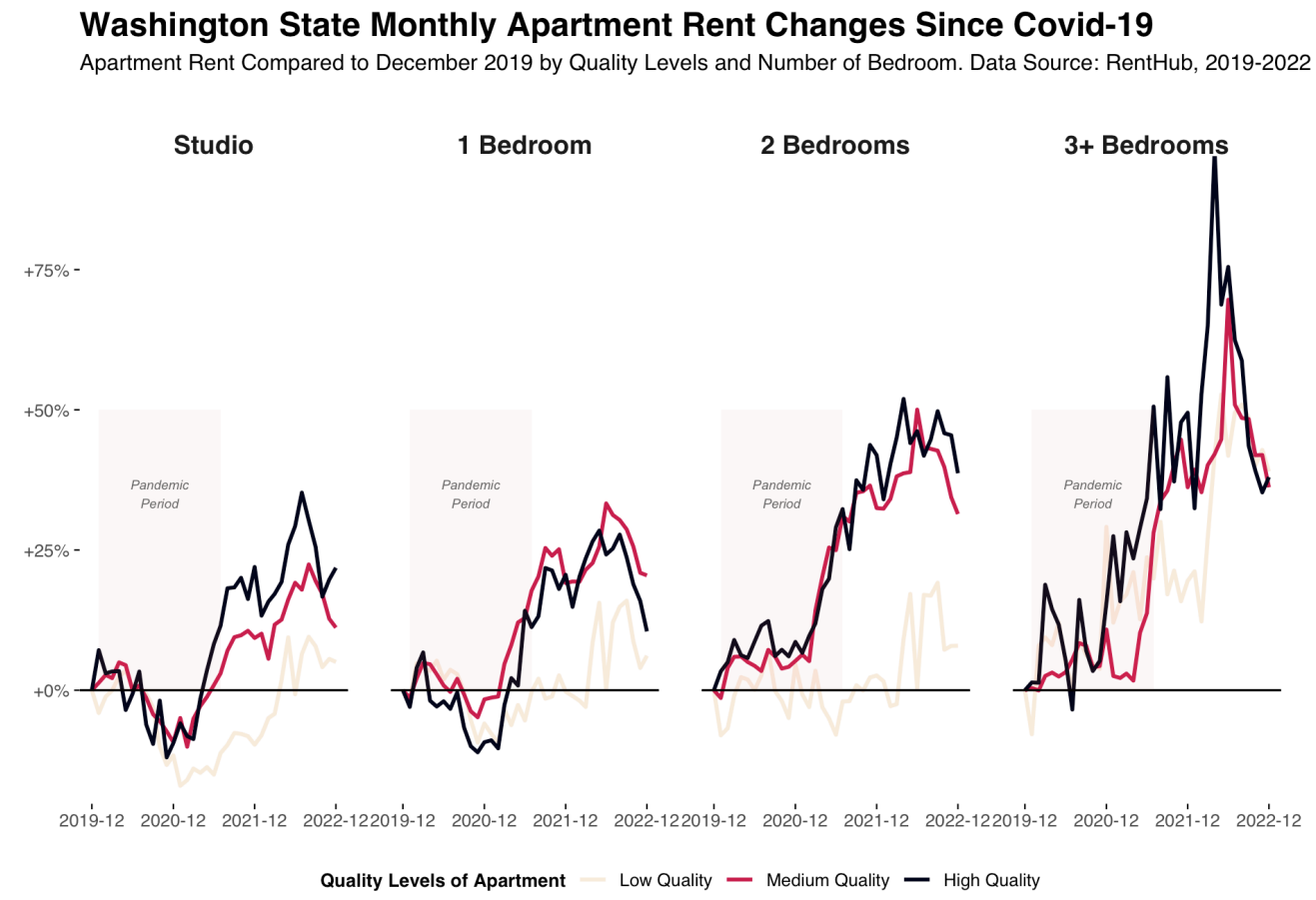
It is useful to know **theme inheritance** feature of `ggplot2`: theme elements inherit properties from other theme elements hierarchically. For example, there is the hierarchy for text elements:

text ├── plot.title ├── plot.subtitle ├── axis.title ├── axis.title.x / axis.title.y └── legend.text / legend.title

Modify our Plot using Theme

```
g_theme <- g_coordinate +
  theme_minimal(base_family = "Helvetica", base_size = 12) +
  theme(
    # ---------------- for the whole plot ----------------
    text = element_text(lineheight = 1.1), # set line height for all text
    plot.title = element_text(face = "bold", size = 18), # make title bold and large
    plot.margin = margin(10, 20, 10, 20), # external margins (top, right, bottom, left)
    # ---------------- for x and y axis ----------------
    axis.title.x = element_blank(), # remove axis.title.x
    axis.title.y = element_blank(),
    axis.ticks.x = element_line(linewidth = 0.4), # style tick marks
    axis.ticks.y = element_line(linewidth = 0.4),
    # ---------------- for grid and panel ----------------
```

```
    panel.grid = element_blank(), # remove the entire grid lines
    panel.spacing.x = unit(1.6, "lines"), # spacing between facets horizontally
    # ---------------- for the legend ----------------
    legend.position = "bottom", # place legend below the plot
    legend.title = element_text(size = 10, face = "bold"),
    # ---------------- for facet ----------------
    strip.background = element_blank(),
    strip.text.x = element_text(face = "bold", # facet labels
                                size = 14,
                                margin = margin(t = 25)) # top margin (t/r/b/l)
  )
g_theme
```

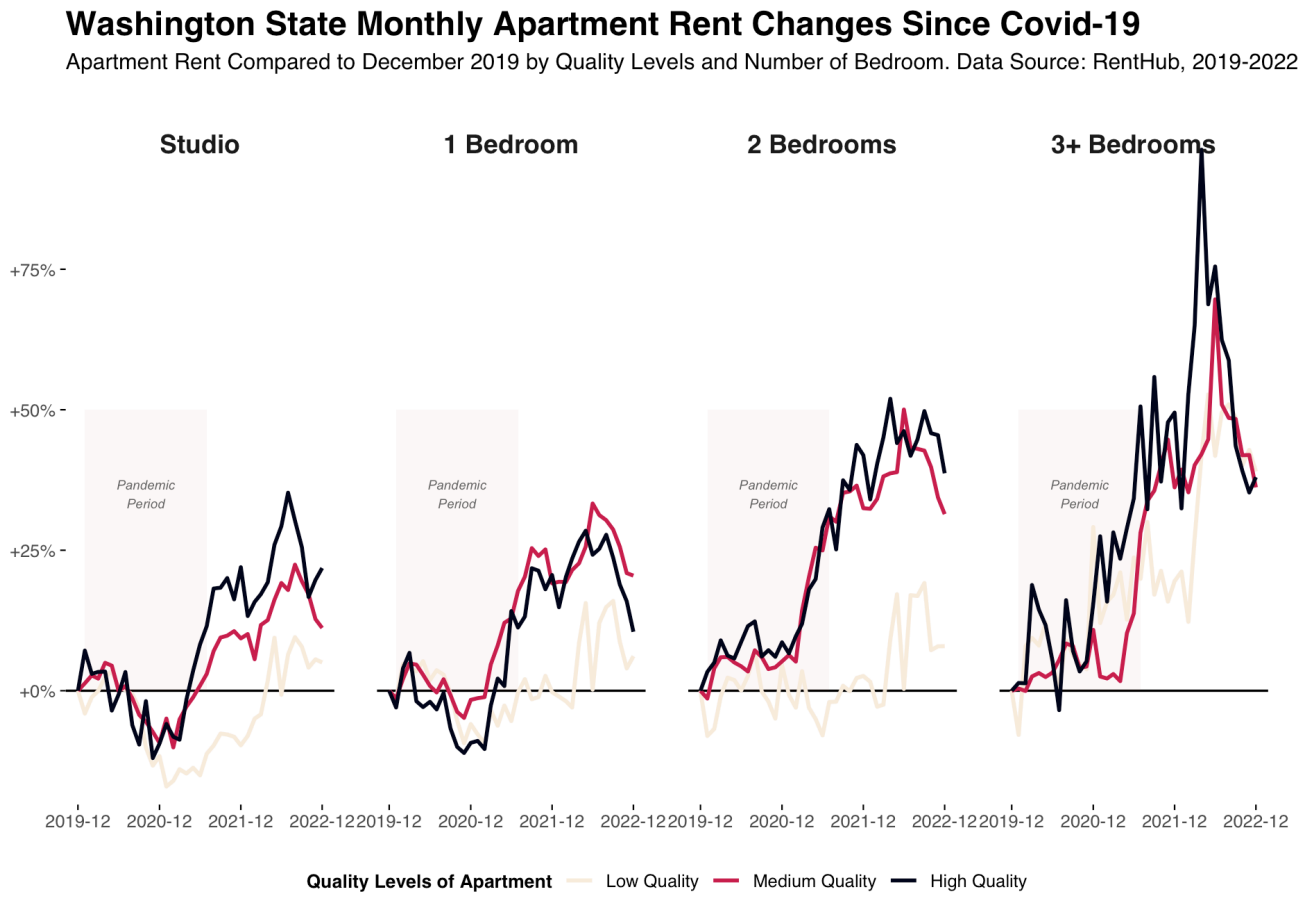


The Complete PLot

```
ggplot(
  filter(apt_rent_rel, month >= as.Date("2019-12-01") & month <= as.Date("2022-12-01")),
  aes(x = month, y = rent_change, color = quality_group)
) +
  # 1️⃣ geometry
  geom_hline(yintercept = 0, linewidth = 0.6) +
  annotate("rect",
```

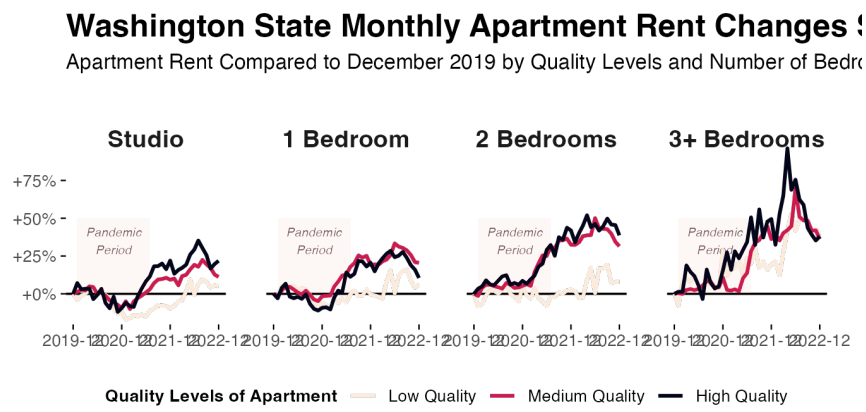
```
          xmin = as.Date("2020-01-01"), xmax = as.Date("2021-07-01"),
          ymin = 0, ymax = 0.5,
          fill = "#e37d5a", alpha = 0.05) +
annotate("text",
          x = as.Date("2020-10-01"),
          y = 0.35,
          label = "Pandemic\nPeriod",
          color = "gray40", size = 2.5, fontface = "italic") +
geom_line(linewidth = 1) +

# 2️⃣ scale
scale_color_viridis_d(option = "rocket", direction = -1) +
scale_x_date(
  breaks = as.Date(c("2019-12-01", "2020-12-01", "2021-12-01","2022-12-01")),
  labels = c("2019-12", "2020-12", "2021-12", "2022-12")
) +
scale_y_continuous(
  labels = function(x) sprintf("%+d%%", x * 100),
  breaks = c(-0.25, 0, 0.25, 0.5, 0.75, 1.0)
) +

# 3️⃣ facet and coordinate system
facet_wrap(~bed_group, nrow = 1) +
coord_cartesian(clip = "off", ylim = c(-0.15, 0.9)) +

# 4️⃣ labels
labs(
  title = "Washington State Monthly Apartment Rent Changes Since Covid-19",
  subtitle = "Apartment Rent Compared to December 2019 by Quality Levels and Number of Bedroom. Data
  x = "Time",
  y = "Relative Rent Change",
  color = "Quality Levels of Apartment"
) +

# 5️⃣ theme
theme_minimal(base_family = "Helvetica", base_size = 12) +
theme(
  text = element_text(lineheight = 1.1),
  plot.title = element_text(face = "bold", size = 18),
  axis.title.x = element_blank(),
  axis.ticks.x = element_line(linewidth = 0.4),
  axis.ticks.y = element_line(linewidth = 0.4),
  axis.title.y = element_blank(),
  legend.position = "bottom",
  legend.title = element_text(size = 10, face = "bold"),
  panel.grid = element_blank(),
  strip.background = element_blank(),
  strip.text.x = element_text(face = "bold", size = 14, margin = margin(t = 25)),
  panel.spacing.x = unit(1.6, "lines"),
  plot.margin = margin(10, 20, 10, 20)
```

```
)
```




```
# save it as 14 inch * 7 inch
ggsave("rent_change_plot.png", width = 14, height = 7, dpi = 200)
# save it as 7 inch * 3.5 inch
ggsave("rent_change_plot_735.png", width = 7, height = 3.5, dpi = 200)
```

When working with ggplot2, customizing the `theme()` controls visual style — but the final output size and aspect ratio are just as important. RStudio preview **IS NOT** the actual exported figure. I will recommend that you work within a single chunk and check changes in exported figure all the time. I decided to use 14 * 7 inch at the beginning so it is perfect when I export as 14 * 7 inch. But the following figure was exported as 7 * 3.5 inch: everything squeezed together.

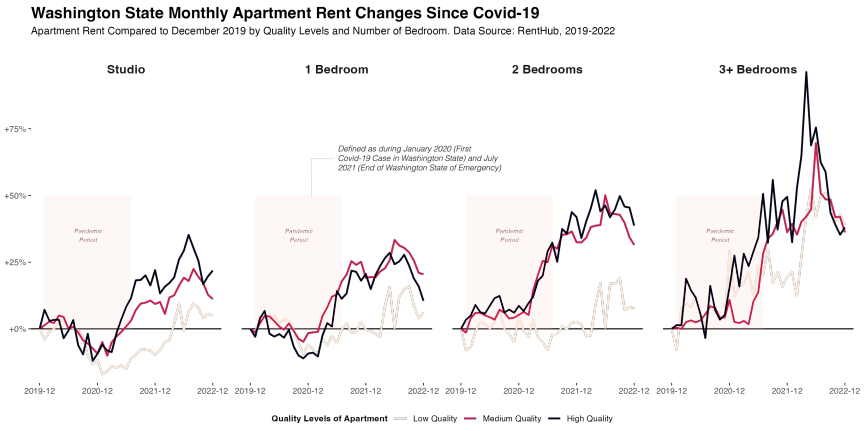


The Exported Figure

Final Edits

Not every figure — or every visual component within a figure — is best created entirely in R.

While `ggplot2` is powerful for data-driven visualization, it's not always flexible for fine-grained graphic design tasks. For example, in this plot, it is cumbersome to adding a single line and text because we used facets. We can always combine R/`ggplot2` with other tools, like PowerPoint and Photoshop. Here, we added an explanation on pandemic and a directional line using Photoshop.



Washington State Monthly Apartment Rent Changes Since Covid-19

📚 TODO: Critique of Data Visualizations

4 points - Please comment on Ed Discussion and no submission needed in the lab

As we learn more about data visualization, we have begun to discuss some general principles. For this section, please find an example of a visualization or infographic that impresses you- either in a good or bad way. You can refer to them to produce the plots for the following two questions (creative and deceptive visualization). Please attach the example image, give the link/source, and:

1. Describe what the visualization is trying to communicate
2. Discuss what it does and does not do well using the principles we talked about in class
3. Comment on two visualizations posted by classmates (optional)

A few probing questions to think about:

1. Is the data visualization static, motion, or interactive?
2. Is it narrative or exploratory?
3. What kind of purpose do you think it serves? How would you order its objectives (appeal, comprehension, and retention)?
4. Does it contain the basic elements of a graph (e.g., title, axis label, encoding)? Is anything missing or not clear to the audience?
5. What kind of encoding does it use (e.g., position, length, slope, angle, area, color hue, etc)? How effective in terms of precision of such encoding elements?
6. What kind of organizational principles do you observe (e.g.hierarchy, unity, balance, grouping, and spacing)? Is the principle lacking in the graph?
7. Do you think your psychological needs have all been met by this source (e.g., autonomy, relatedness, and competence)?

📚 TODO: Creative Visualization

8 points

In this task, you will select one central theme or message and explore how different visual design choices can communicate it. You are welcome to use the rental dataset or other datasets you are interested in.

- **What to do**
  - think about actions/targets of visualization.
  - choose different visualization idioms rather than line plots whenever possible. Reference website A and B.

    - experiment with as many `ggplot2` functions, arguments, and customization features as possible — such as different geometries, scales, themes, annotations, and color schemes — to creatively express your message.
    - add clear code comments to explain what it does and why you used it, so that others can follow your reasoning.
    - demonstrate both technical variety (breadth of ggplot usage) and conceptual clarity (a clear story or insight).
    - be bold, try new features, and document your process carefully, creativity and transparency matter more than perfection here.
    - short write-up stating the message, key design choices, and what the viewer should notice.
- **Grading**
    - [2 pts] a clear theme or message and short write-up
    - [2 pts] proper usage of multiple `ggplot2` components with enough elements of a graph (such as data source)
    - [2 pts] well-structured codes with meaningful comments explaining arguments and choices
    - [2 pts] thoughtful design, effective use of color, layout, and style

📚 TODO: Deceptive Visualization

4 points

Building on your previous figure, intentionally create a deceptive version of the same visualization — one that misleads the audience while still appearing legitimate at first glance. Your challenge is to subtly distort perception without using obvious tricks such as removing data or adding false numbers.

For instance, you might manipulate scales, reorder categories, or change visual emphasis in ways that guide the viewer toward an incorrect interpretation. In your write-up, briefly reflect on what design decisions make the visualization deceptive, and how small graphical changes can influence interpretation. This exercise is meant to help you understand visual ethics and recognize how easy it is to mislead unintentionally through design.

- **Grading**
    - [3 pts] misleading in a realistic and non-obvious

- [1 pts] reflection on how the visualization misleads and what design decisions contributed to it

📚 TODO: Tableau Installization (optional)

0 point

Next week, we are going to use Tableau, which provides free access to students. Please try to download Tableau for Students in advance.

Acknowledgement

The materials are developed by Haoyu Yue based materials from Dr. Feiyang Sun at UC San Diego, Siman Ning and Christian Phillips at University of Washington, Dr. Charles Lanfear at University of Cambridge.