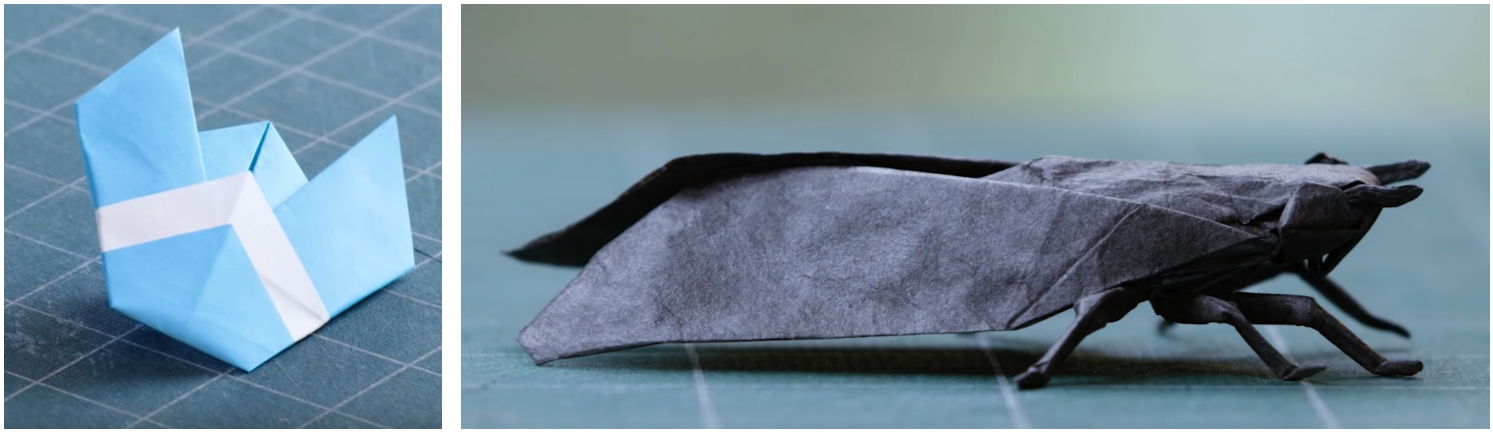
10 Levels of ggplot2: Fro

Ryan Harrington Follow
Dec 13, 2019 · 9 min read

Recently I discovered WIRED's "Levels" series on YouTube. The concept is simple. An expert of some interesting skill (i.e., ice sculpture, origami, or knife making) explains the concept in many levels — from easy to complex.



Level 1 origami cicada vs. Level 11 origami cicada from **11 Levels of Origami: Easy to Complex**

This format is a wonderful way to explore many different skills.

One skill that is essential for anyone who works with data to learn is how to build a graph that tells a story. The `ggplot2` package is one of the best tools to do that. In this article I'll explore how to build a graph with `ggplot2` from basic to beautiful.

Data To Explore

We'll be using a dataset near and dear to my heart. **Philadelphia Parking Tickets.** You can get some more information about the dataset here.

Also, as we're exploring the data, you can follow along with all of the code for it here:

Level 0: A Basic Plot

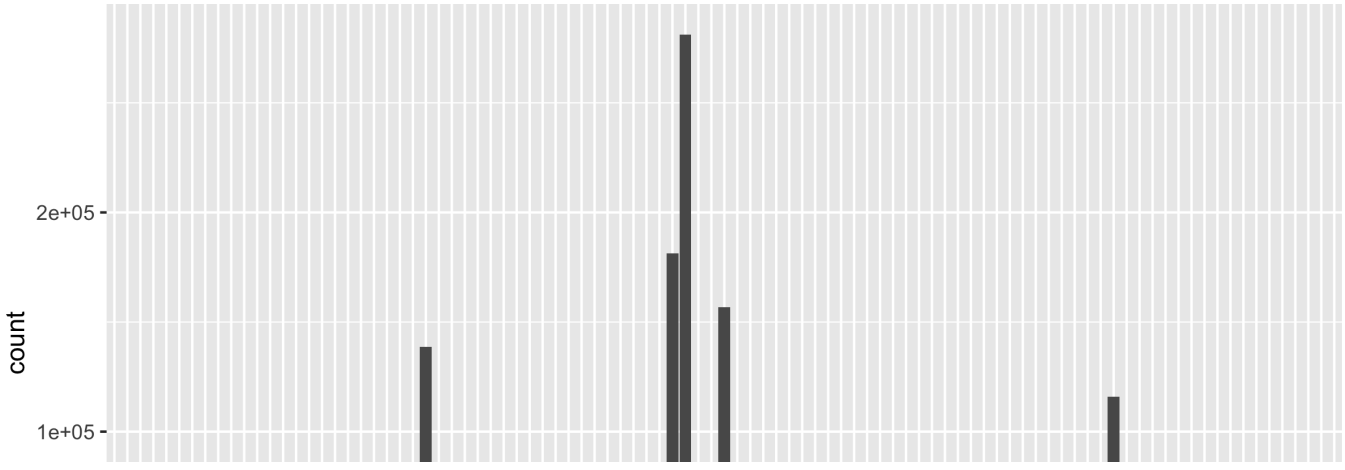
I can't even count this as a real level. It's a pre-level. Fundamentally, this is what a `ggplot2` code block will always look like. A `ggplot2` code block will always include:

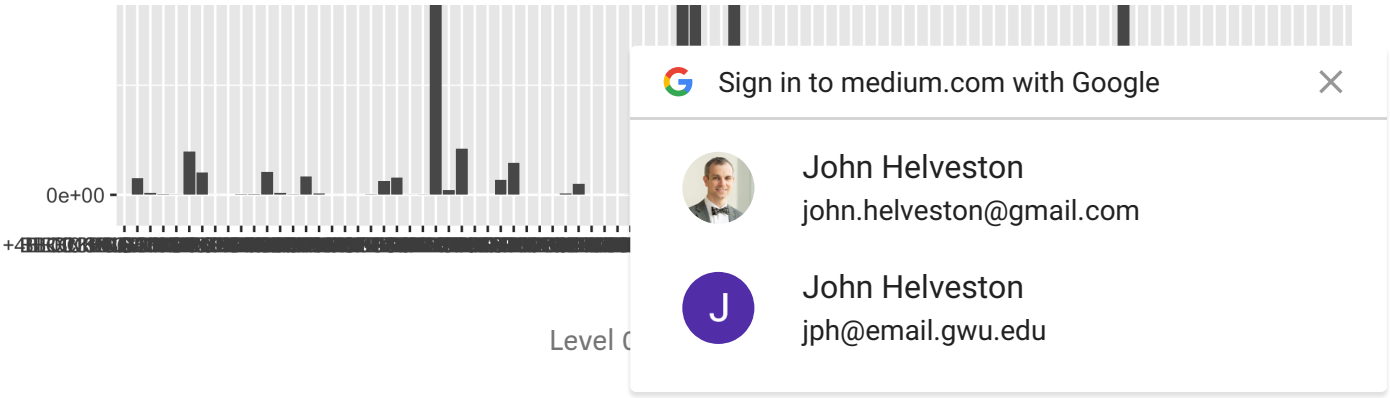
- **Data** (which we see in the `tickets` object)
- **Mapping** (which we see in the `aes(x = violation_desc)` parameter)
- **Geometry** (which we see from `geom_bar`)

For more details about how this works, check out this article. Our code block ends up looking like:

```
tickets %>%
  ggplot(aes(x = violation_desc)) +
  geom_bar()
```

Here's what we get from this: hot garbage. There's good news, though. We can only get better from here!





Level 1: Data Cleaning

There are a lot of fundamental issues with our Level 0 graph. The biggest issue, though, comes *before* we ever even make the graph. Before we do anything else, we need to clean our data.

The Level 0 graph has 95 distinct values on the x-axis. This is *far* too many for a person to be able to interpret. Trimming the number of distinct values to 10 will dramatically improve the interpretability of the graph. We can start doing this by inspecting the distinct values. Here's a sample of the values with their counts:

```
 1 METER EXPIRED CC     281060
 2 METER EXPIRED        181329
 3 OVER TIME LIMIT      156859
 4 EXPIRED INSPECTION   138575
 5 STOP PROHIBITED CC   115898
 6 STOPPING PROHIBITED   47395
 7 PARKING PROHBITED     47232
 8 PARKING PROHBITED CC  45082
 9 OVER TIME LIMIT CC    24585
10 PASSENGR LOADNG ZONE  24359
```

Immediately we start to see how to improve the number of distinct values that we have. First, for our purposes, we can remove any of the `CC` suffixes. We notice that there are some words that are misspelled like `PROHBITED`. There are also words that are similar to each other like `STOP` and `STOPPING`. Using the `stringr` package's `str_replace` function can help us make those changes.

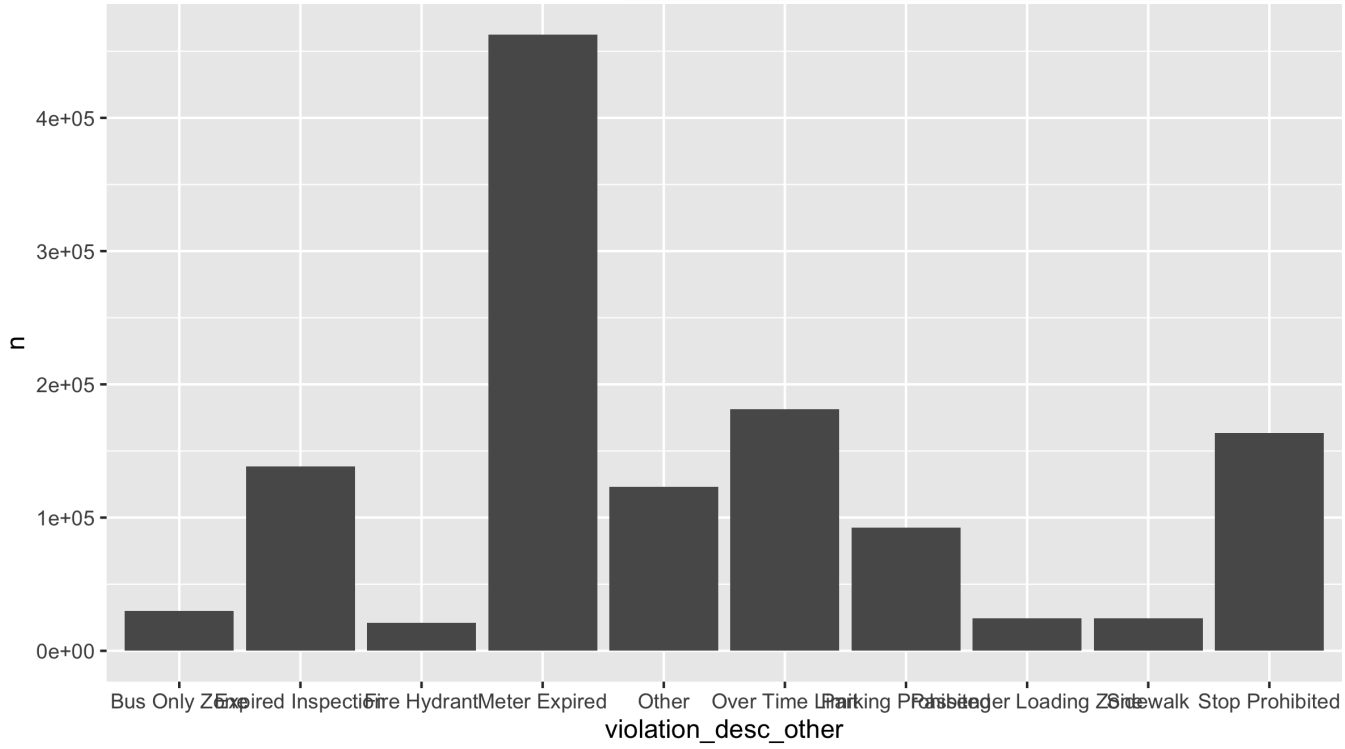
Once we make all of these changes, we're still left with 81 distinct values. A great method to fix this is to use the `fct_lump` function from the `forcats` package. The

function is simple enough — it allows you to decide how many distinct values you'd like to keep based upon some other value. Ev

Check those steps out here.

After implementing these changes we're a

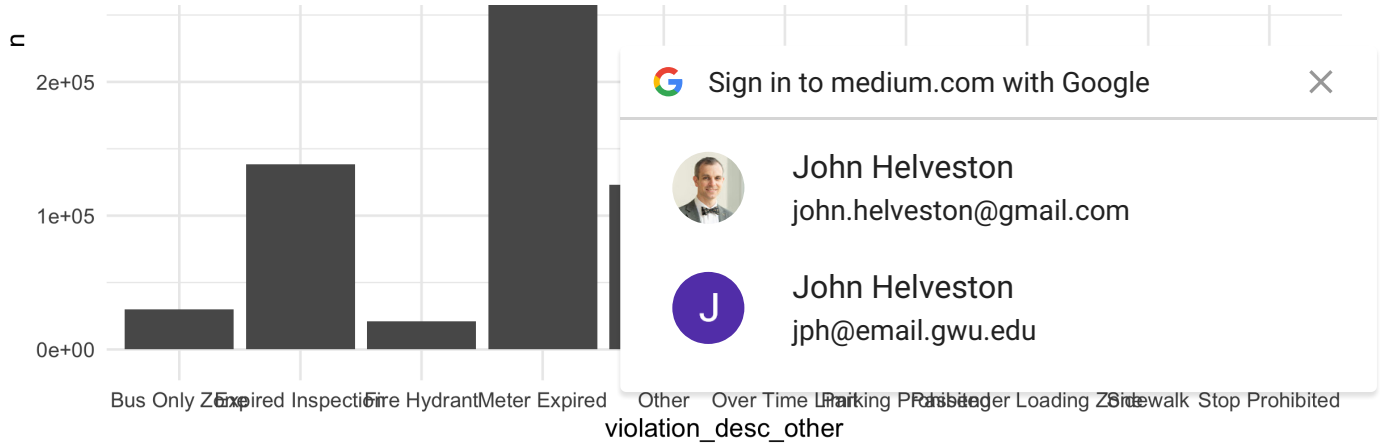
axis. There are a bunch of ways to take ca



Level 1: Less Hot Garbage

Level 2: Use Default Themes

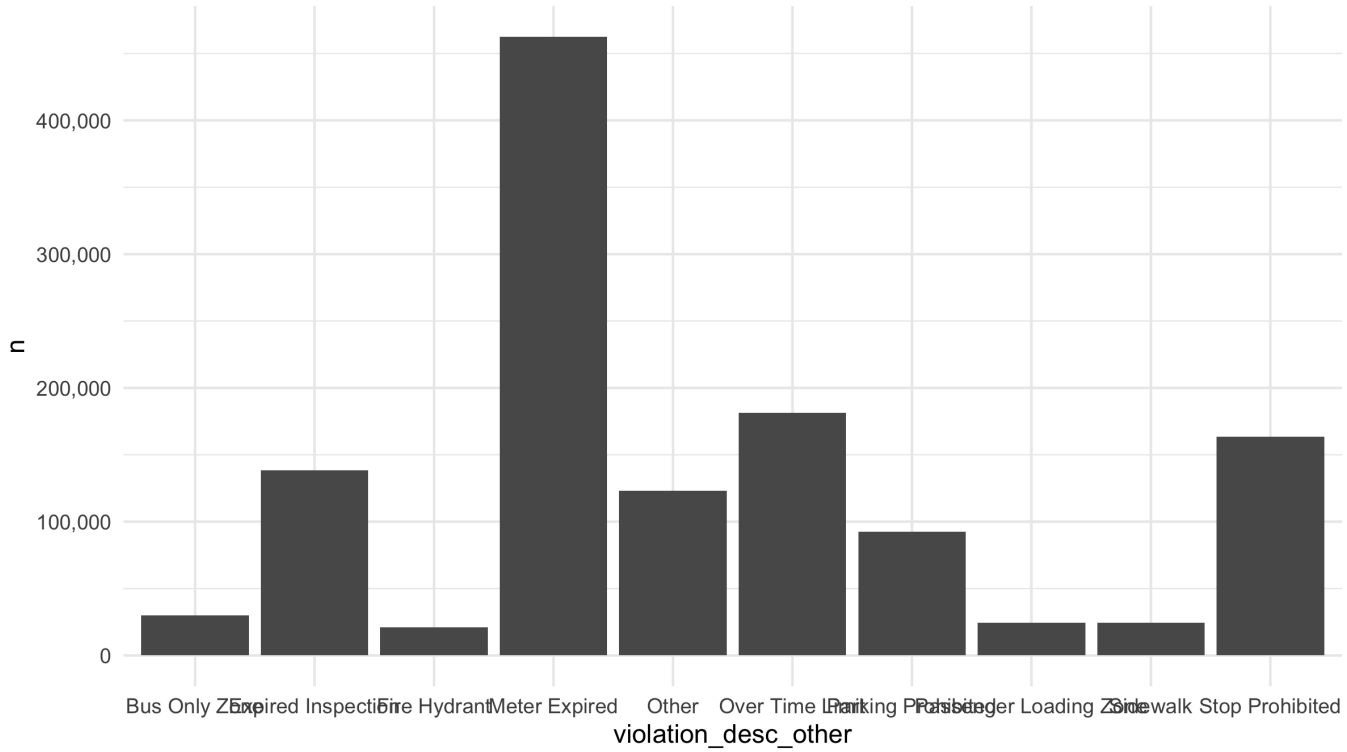
The gray background of the default `ggplot2` theme is borderline iconic (and never changing). There are plenty of other themes to choose from, though. I have always been a fan of the simplicity of `theme_minimal`. You should explore some of the others here. When we add this code in, we're left with:



Level 2: Merely Garbage

Level 3: `scale_y_continuous + scales::comma`

The next level is to control how our axes look. It turns out that scientific notation (`0e+00`, `1e+05`, etc.) is a less-than-ideal way of looking at an axis. We can use the `scale_y_continuous` function coupled with the `comma` function from the `scales` package to adjust the y-axis. You can see that in code here. We end up with a much nicer y-axis.



Level 3: Approaching Okay

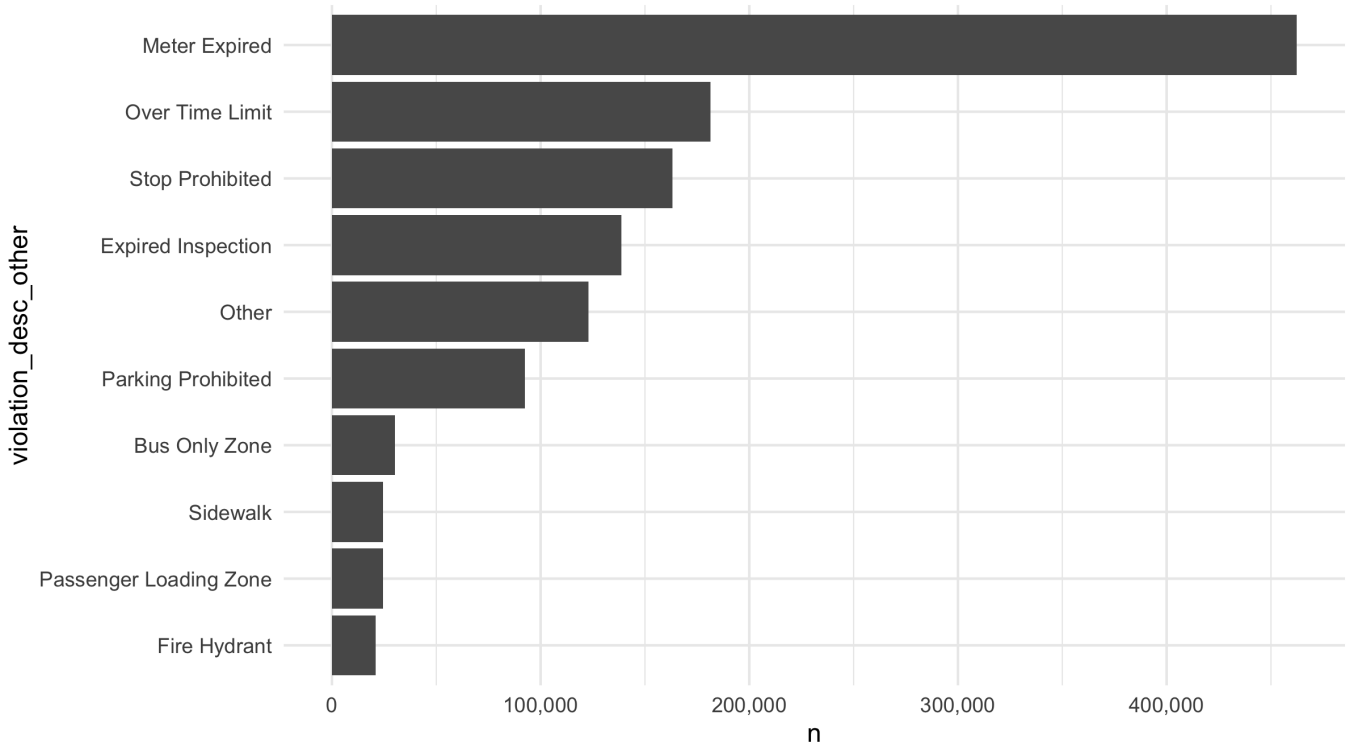
Level 4: fct_reorder + coord_flip()

After some of that prep work, we can now take care of making it easier to read our x-axis. One great technique to use for this is to p

`coord_flip` function. Similar to `fct_lump` `forcats` package and does what it says: i We add this to our data prep process by u descriptions. After doing this, the bars of

The `coord_flip` function can then be included as part of our `ggplot2` code. This function also does what it says: it flips the axes so that our x-axis becomes our y-axis and vice versa. Our code ends up looking like this. After implementing it we end up with our first passable graph.

This would be a great graph to use internally or as part of a draft. I would want to make more changes if it were being shared externally.



Level 4: Not Too Shabby!

Level 5: Window Dressing

There are a few simple tweaks that we can make in order to make the graph presentable:

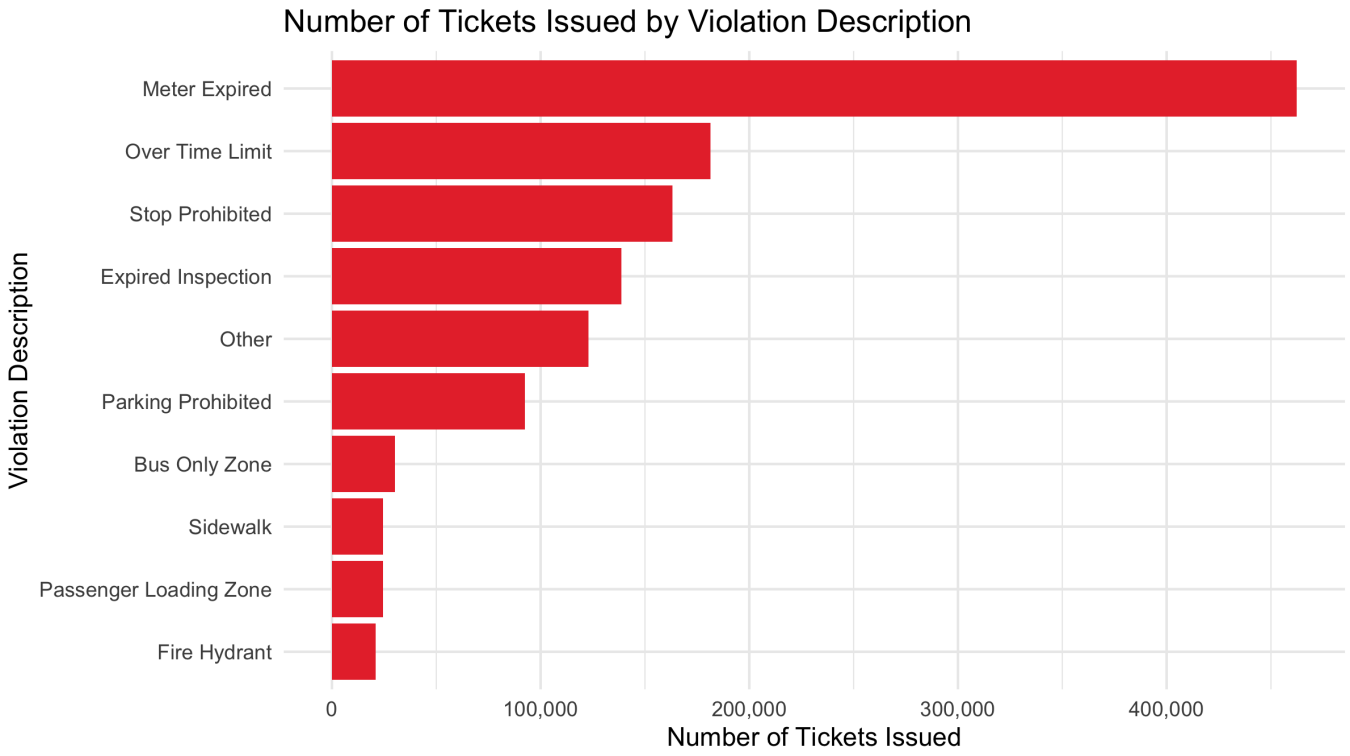
1. Change the titles of the axes

2. Add a title for the graph

3. Add some color

We can take care of the first two issues by
`ggplot2` code. Any of the titles on the plo
`legends` , etc.) can be altered in this way.

To add color, we need to go to our `geom_col` function. We can pass an additional parameter to it called `fill` . We're able to designate the fill color in with this parameter in several ways — for example, as a name (`"red"`) or a hex code (`"#E83536"`). Now we have a graph that is very presentable. We're well past "basic" and on our way to "beautiful"!

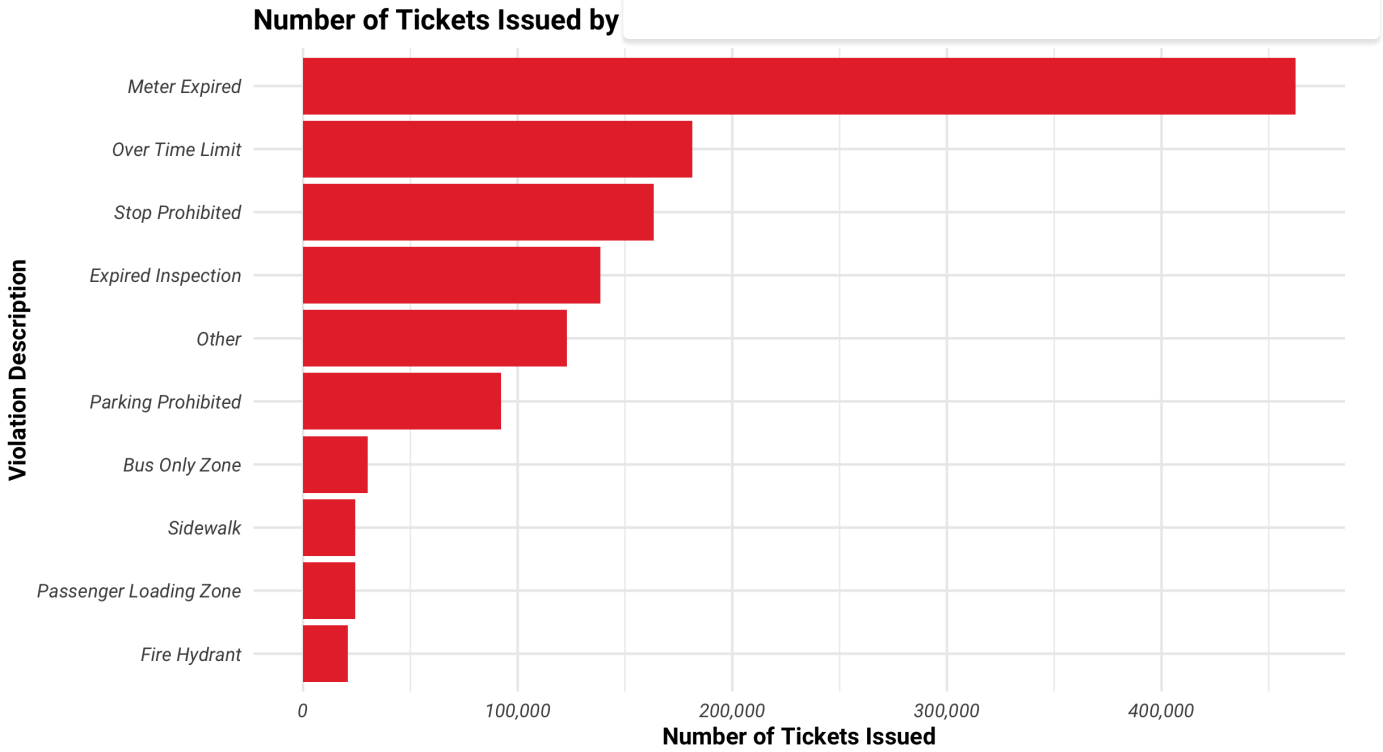


Level 5: The Midway Point

Level 6: Build Your Own Theme

Once we've applied some basic window dressing, we now have the opportunity to dive into the `theme` function. There are many components that can be modified via `theme` . Changes can be made at a high level to the entire graph and then further fine tuned. For example, in this case, we might decide to change the `family` of **all** of the `text` to `Roboto`

with a default `size` of 10. From there, we are able to specifically change the format of the titles of the axes by specifying `axis.t` (change each individual axis on its own), `axis.text.y`. After these implementation highly customized looking graph.



Level 6: Feeling Cute, Might Graph Later

At this point, we have something that looks like it could be our own theme. If we're making multiple plots, it would be worthwhile to officially *build* our own theme. Here's what that could look like:

```
theme_compassred <- function () {
  theme_minimal(base_size = 10, base_family = "Roboto") %+replace%
    theme(axis.title = element_text(face = "bold"),
          axis.text = element_text(face = "italic"),
          plot.title = element_text(face = "bold",
                                    size = 12)

    )
}
```

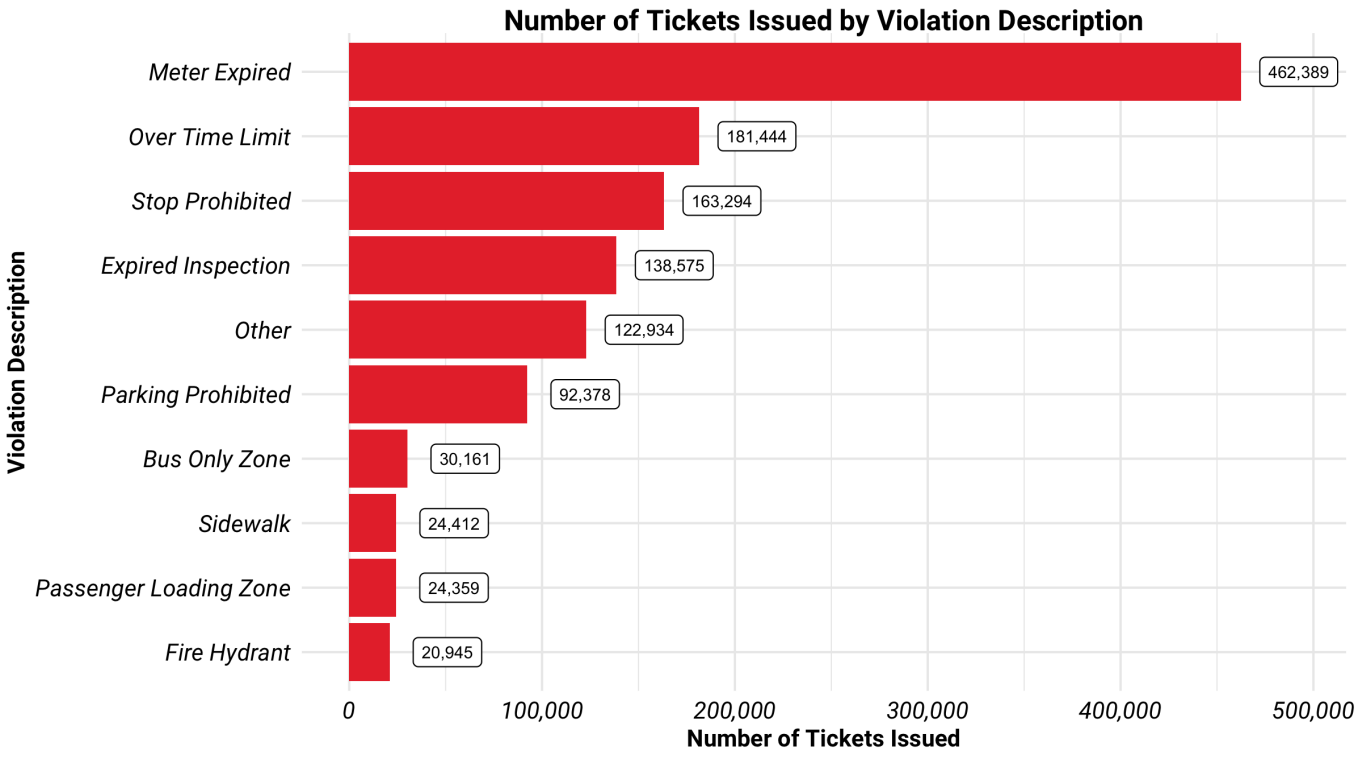
Even further, we can set this theme for every plot automatically:


```
theme_set(theme_compassred())
```

Level 7: Layer Multiple Geo

One of the beautiful features of `ggplot2` is the ease by which multiple geometries can be used in sync with each other. Currently, we have been using `geom_col` on its own. However, we can make it easier to read the plot by adding a `geom_label` as well.

When we have multiple `geom_` functions as part of our code, it is possible for the geometries to either use the same aesthetic mapping or to have different aesthetic mappings. In this case, `geom_label` can use the same mappings for its `x` and `y` coordinates, but requires an additional mapping called `label`. There are also several parameters that we must pass to `geom_label` in order to properly place the labels on the graph. Here's the code and the graph:



Level 7: Onions Have Layers

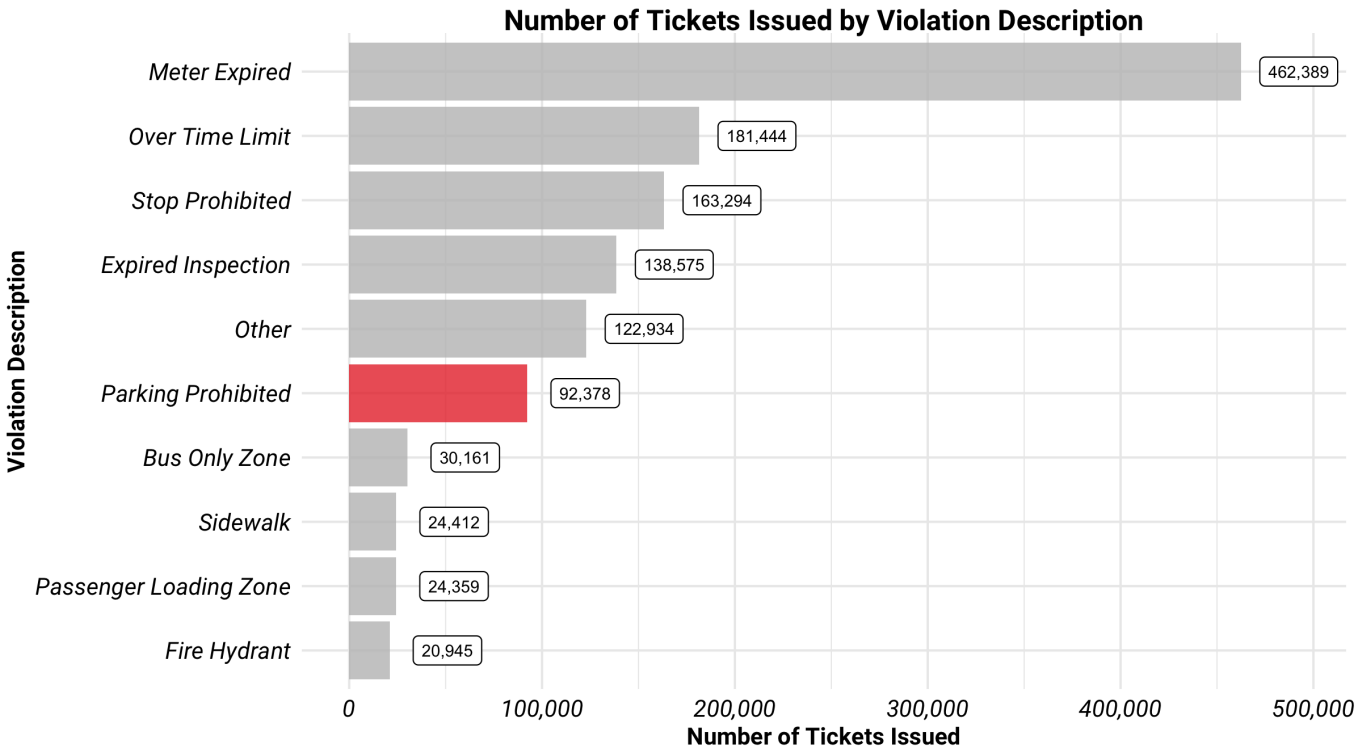
Level 8: Highlight a Key Field

At this point with our graph, the end user still has to do quite a bit of work to interpret the results. The key goal of our last levels and make it easier for them to immediate

One great way to accomplish this is by hi small changes:

- Transform the data by creating a new field called `highlight`
- Move the `fill` to inside the aesthetic mapping and set it to `highlight`
- Change colors with `scale_fill_manual`
- Remove the legend with `guides`

After those small code changes, we end up with:



Level 8: Ombre

Level 9: Annotate

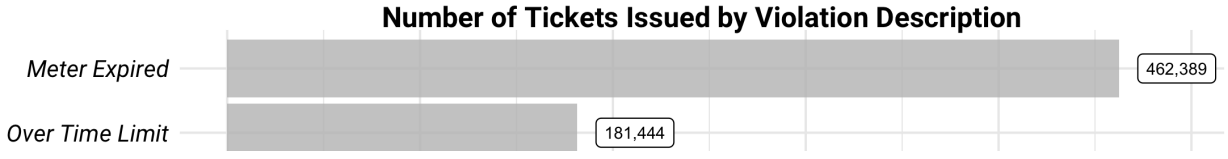
In this level, our goal is still to remove the cognitive load on the end user. Highlighting the value that we care about helps a lot, but we can literally spell out our takeaway even

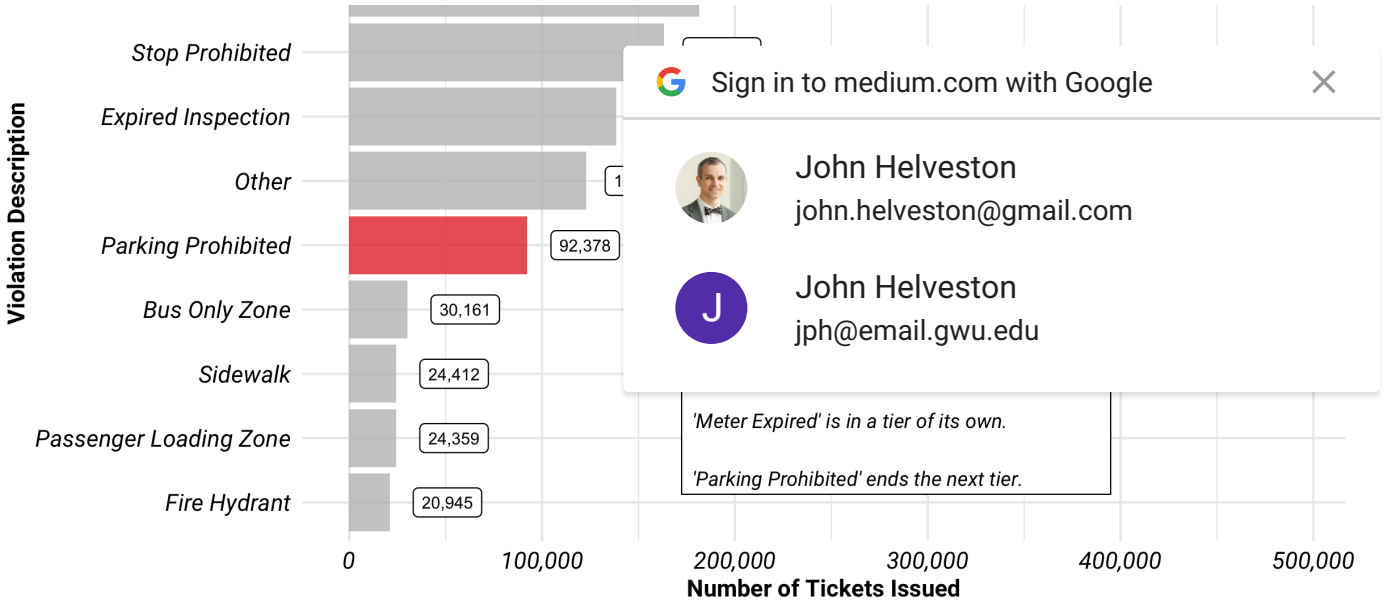
further. There are many ways to do this, but we'll do so by taking advantage of an additional `geom_label`, a `geom_curve`, and

At a high level, here is how our code is re

```
# Build an aggregated data frame
tickets_agg <-
   mutate(...)

# Create the note that we'd like to share
ticket_note <-
   paste("Your text here")

# Create a dataframe to position your arrow
arrow_position <-
   data.frame(...)

# Build your graph
ggplot() +
   geom_col(data = tickets_agg,
            mapping = aes(x    = field,
                          y    = count,
                          fill = highlight)) +
   geom_label(data = tickets_agg,
              mapping = aes(x     = field,
                            y     = count,
                            label = count)) +
   geom_label(data = filter(tickets_agg,
                            highlight = T),
              mapping = aes(x     = field,
                            y     = count,
                            label = ticket_note)) +
   geom_curve(data = arrow_position,
              mapping = aes(x    = x_start,
                            y    = y_start,
                            xend = x_end,
                            yend = y_end))
```

In Level 7 we mentioned that when there are multiple `geom_` functions in our code that sometimes we need to give the different `geom_` functions different data and mappings. We take full advantage of this here.



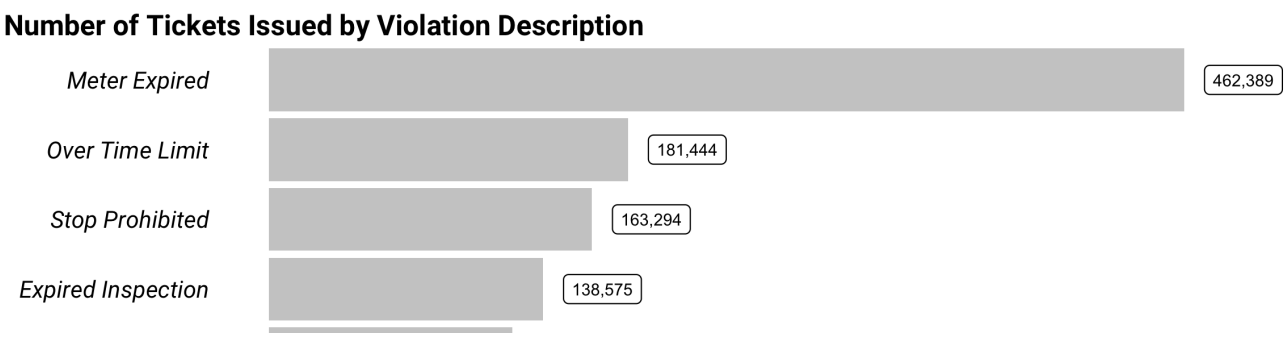


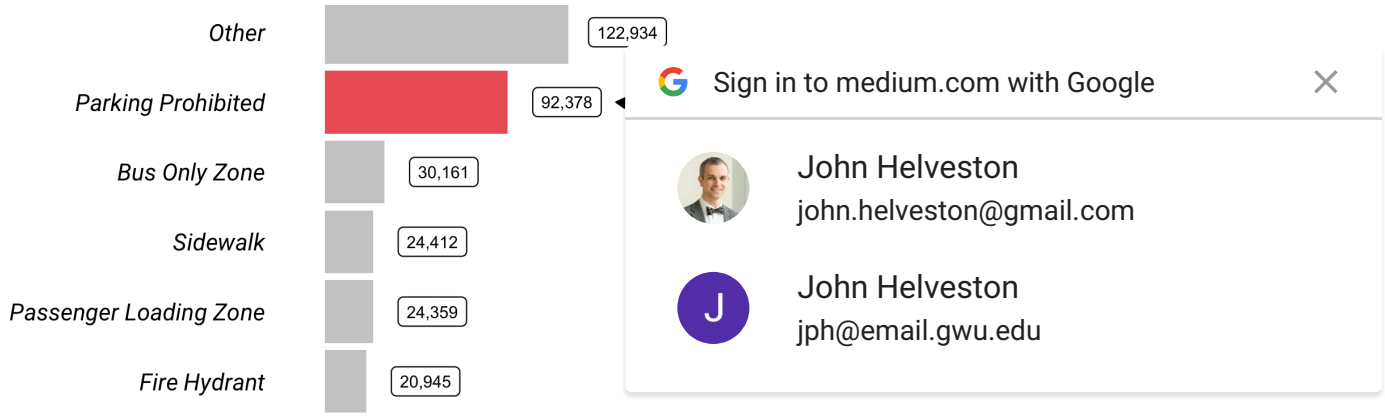
Level 10: Tidy It Up

The last level is to reduce the cognitive load on the end user by removing everything that isn't necessary. This is actually a relatively straightforward exercise, but requires you to put yourself in the shoes of the end user. You must answer the question "what can be removed while still making the graph clear?" Here are all of the things that we remove here:

- We don't need all of the grid lines. Having labels for all of the bars makes the grid lines redundant.
- Similarly, we don't need the values of our x-axis. These are also redundant because of the labels.
- We don't need either of our axis labels. The title makes it very clear what each axis represents. They end up just being noise.

Putting all of this together, we end up with our final, Level 10 graph.

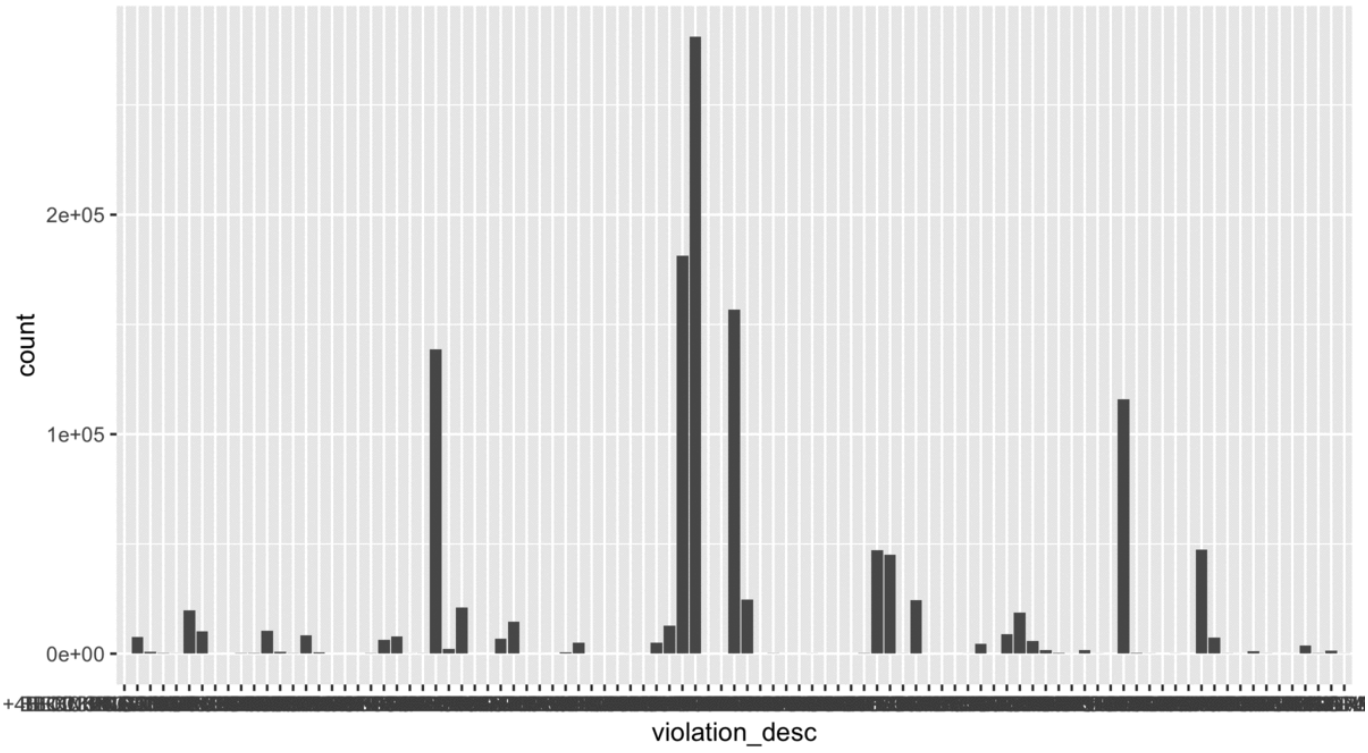




Level 10: 🔥 🔥 🔥

Putting it All Together

Overall, a lot of simple steps (and a couple of more complicated ones) can take your graph from basic to beautiful. Here's what that transformation looks like, one step at a time:



And here's how our final block of code ends up looking:

```
tickets_agg <-
  tickets %>%
  mutate(violation_desc_condensed = str_remove(violation_desc,
```

```
4                                                  " CC$"),
5           violation_desc_condensed = str_t
6           violation_desc_condensed = str_r
7           violation_desc_condensed = str_r
8           violation_desc_condensed = str_r
9           violation_desc_condensed = str_r
10          violation_desc_condensed = str_r
11          violation_desc_condensed = str_r
12          violation_desc_other = fct_lump(
13          violation_desc_other = str_to_title(violation_desc_other)) %>%
14    count(violation_desc_other) %>%
15    mutate(violation_desc_other = fct_reorder(violation_desc_other, n),
16           highlight = violation_desc_other == "Parking Prohibited")
17
18  ticket_note <-
19    paste("There appears to be different tiers of violations\n",
20          "based upon the number of tickets issued.\n\n",
21          "'Meter Expired' is in a tier of its own.\n\n",
22          "'Parking Prohibited' ends the next tier.")
23
24  arrow_position <- data.frame(x_start = 3.85,
25                               x_end   = 5,
26                               y_start = 300000,
27                               y_end   = 147000)
28
29  ggplot() +
30    geom_col(data = tickets_agg,
31             mapping = aes(x    = violation_desc_other,
32                           y    = n,
33                           fill = highlight),
34             alpha = 0.8) +
35    geom_label(data = tickets_agg,
36               aes(x     = violation_desc_other,
37                   y     = n,
38                   label = scales::comma(n)),
39               size = 2.5,
40               nudge_y = 30000) +
41    geom_label(data = filter(tickets_agg,
42                             highlight == T),
43               mapping = aes(x     = violation_desc_other,
44                             y     = n,
45                             label = ticket_note),
46               hjust = 0,
47               nudge_x = -2.5,
48               nudge_y = 80000,
```

```r
            family = "Roboto",
            size = 3,
            fontface = "italic",
            label.r = unit(0, "lines")) +
  geom_curve(data = arrow_position,
            mapping = aes(x    = x_start,
                          y    = y_start,
                          xend = x_end,
                          yend = y_end),
            curvature = 0.1,
            size = 0.25,
            # color = "gray75",
            arrow = arrow(length = unit(0.015, "npc"),
                          type = "closed")) +
  scale_y_continuous(labels = scales::comma) +
  scale_fill_manual(values = c("gray75", "#E83536")) +
  coord_flip() +
  labs(title = "Number of Tickets Issued by Violation Description") +
  guides(fill = "none") +
  theme(panel.grid = element_blank(),
        axis.title = element_blank(),
        axis.text.x = element_blank(),
        plot.title = element_text(hjust = -0.5))
```

Level 10 Code.R hosted with ❤ by GitHub view raw

Thanks to Eugene Olkhov.

Data Visualization Ggplot2 Tutorial Rstats