

## Reporting data results #2

---

## Example data

---

## Example data

This week, we'll be using some example data from NOAA's Storm Events Database.

Go to <https://www1.ncdc.noaa.gov/pub/data/swdi/stormevents/csvfiles/> and download the bulk storm details data for 2017, in the file that starts "StormEvents\_details" and includes "d2017".

Move this into a good directory for your current working directory and read it in using `read_csv` from the `readr` package.

## Example data

This data lists major weather-related storm events during 2017.

For each event, it includes information like the start and end dates, where it happened, associated deaths, injuries, and property damage, and some other characteristics.

## Example data

```
colnames(storms_2017)
```

##	[1]	"BEGIN_YEARMONTH"	"BEGIN_DAY"	"BEGIN_TIME"
##	[4]	"END_YEARMONTH"	"END_DAY"	"END_TIME"
##	[7]	"EPISODE_ID"	"EVENT_ID"	"STATE"
##	[10]	"STATE_FIPS"	"YEAR"	"MONTH_NAME"
##	[13]	"EVENT_TYPE"	"CZ_TYPE"	"CZ_FIPS"
##	[16]	"CZ_NAME"	"WFO"	"BEGIN_DATE_TIME"
##	[19]	"CZ_TIMEZONE"	"END_DATE_TIME"	"INJURIES_DIRECT"
##	[22]	"INJURIES_INDIRECT"	"DEATHS_DIRECT"	"DEATHS_INDIRECT"
##	[25]	"DAMAGE_PROPERTY"	"DAMAGE_CROPS"	"SOURCE"
##	[28]	"MAGNITUDE"	"MAGNITUDE_TYPE"	"FLOOD_CAUSE"
##	[31]	"CATEGORY"	"TOR_F_SCALE"	"TOR_LENGTH"
##	[34]	"TOR_WIDTH"	"TOR_OTHER_WFO"	"TOR_OTHER_CZ_STATE"
##	[37]	"TOR_OTHER_CZ_FIPS"	"TOR_OTHER_CZ_NAME"	"BEGIN_RANGE"
##	[40]	"BEGIN_AZIMUTH"	"BEGIN_LOCATION"	"END_RANGE"
##	[43]	"END_AZIMUTH"	"END_LOCATION"	"BEGIN_LAT"
##	[46]	"BEGIN_LON"	"END_LAT"	"END_LON"
##	[49]	"EPISODE_NARRATIVE"	"EVENT_NARRATIVE"	"DATA_SOURCE"

## Example data

Each row is a separate **event**. However, often several events are grouped together within the same **episode**.

```
nrow(storms_2017)
```

```
## [1] 56989
```

```
length(unique(storms_2017$EVENT_ID))
```

```
## [1] 56989
```

```
length(unique(storms_2017$EPISODE_ID))
```

```
## [1] 8964
```

## Example data

Some of the event types are listed by their county ID (FIPS code) ("C"), but some are listed by a forecast zone ID ("Z"). Which ID is used is given in the column CZ\_TYPE:

```
summary(factor(storms_2017$CZ_TYPE))
```

```
##      2      C      Z  
##    17 36933 20039
```

## Example data

For the first in-course exercise, you will clean up this data a bit for us to use in the rest of the class.

- Download and read in the data
- Limit the dataframe to: the beginning and ending dates and times, the episode ID, the event ID, the state name and FIPS, the “CZ” name, type, and FIPS, the event type, the source, and the beginning latitude and longitude and ending latitude and longitude
- Convert the beginning and ending dates to a “date-time” class (there should be one column for the beginning date-time and one for the ending date-time)
- Change state and county names to title case (e.g., “New Jersey” instead of “NEW JERSEY”)
- Limit to the events listed by county FIPS (CZ\_TYPE of “C”) and then remove the CZ\_TYPE column
- Pad the state and county FIPS with a “0” at the beginning (hint: there’s a function in `stringr` to do this) and then unite the two columns to make one `fips` column with the 5-digit county FIPS code
- Change all the column names to lower case (you may want to try the `rename_all` function for this)



# Example data

```
storms_2017 %>%  
  slice(1:3)
```

```
## # A tibble: 3 x 13
```

```
##   begin_date_time      end_date_time      episode_id event_id state fips  
##   <dtm>              <dtm>              <int>    <int> <chr> <chr>  
## 1 2017-04-06 15:09:00 2017-04-06 15:09:00    113355    678791 New ~ 34015  
## 2 2017-04-06 09:30:00 2017-04-06 09:40:00    113459    679228 Flor~ 12071  
## 3 2017-04-05 17:49:00 2017-04-05 17:53:00    113448    679268 Ohio  39057  
## # ... with 7 more variables: event_type <chr>, cz_name <chr>,  
## #   source <chr>, begin_lat <dbl>, begin_lon <dbl>, end_lat <dbl>,  
## #   end_lon <dbl>
```

## State data

There is data that comes with R on U.S. states. Use that to create a dataframe with the state name, area, and region:

```
data("state")  
us_state_info <- data_frame(state = state.name,  
                             area = state.area,  
                             region = state.region)
```

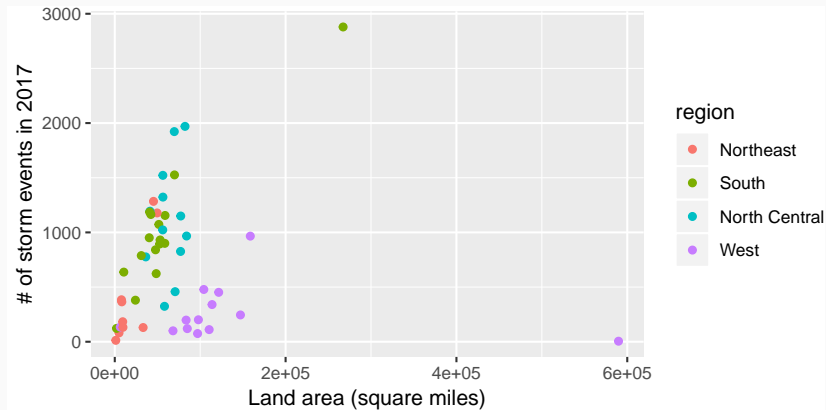
## Example data

Create a dataframe with the number of events per state in 2017. Merge in the state information dataframe you just created. Remove any states that are not in the state information dataframe.

```
state_storms <- storms_2017 %>%  
  group_by(state) %>%  
  count() %>%  
  ungroup() %>%  
  right_join(us_state_info, by = "state")
```

## Example data

Ultimately, in this group exercise, you will create a plot of state land area versus the number of storm events in the state:



We'll now take a break to do the first part of the in-course exercise.

## ggplot2 extras and extensions

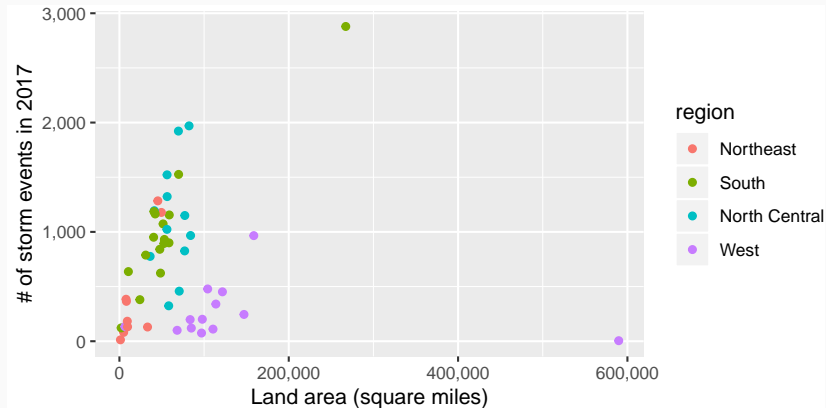
---

The `scales` package gives you a few more options for labeling with your `ggplot` scales. For example, if you wanted to change the notation for the axes in the plot of state area versus number of storm events, you could use the `scales` package to add commas to the numeric axis values.

For the rest of these slides, I've saved the `ggplot` object with out plot to the object named `storm_plot`, so we don't have to repeat that code every time.

# scales package

```
library(scales)
storm_plot +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma)
```





The scales package also includes labeling functions for:

- `dollars(labels = dollar)`
- `percent(labels = percent)`

The ggplot2 framework is set up so that others can create packages that “extend” the system, creating functions that can be added on as layers to a ggplot object.

Some of the types of extensions available include:

- More themes
- Useful additions (things that you may be able to do without the package, but that the package makes easier)
- Tools for plotting different types of data

## Where to find ggplot2 extensions

There is a gallery with links to ggplot2 extensions at <https://www.ggplot2-exts.org>.

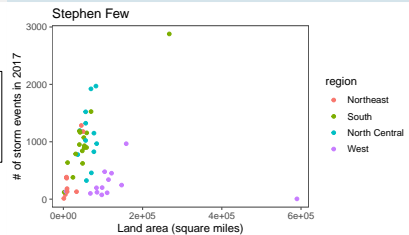
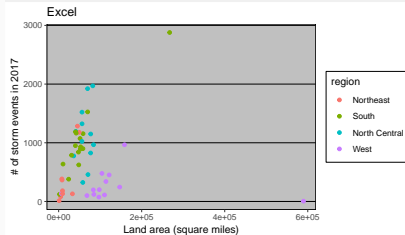
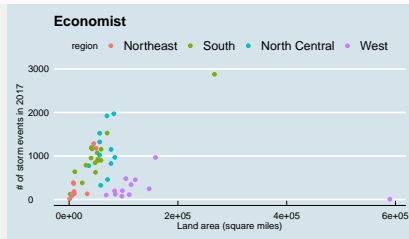
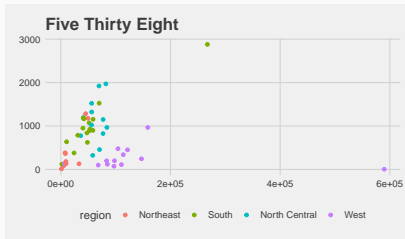
This list may not be exhaustive—there may be other extensions on CRAN or on GitHub that the package maintainer did not submit for this gallery.

You have already played around a lot with using ggplot themes to change how your graphs look.

Several people have created packages with additional themes:

- `ggthemes`
- `ggthemr`
- `ggtech`
- `ggsci`

# Some ggthemes themes

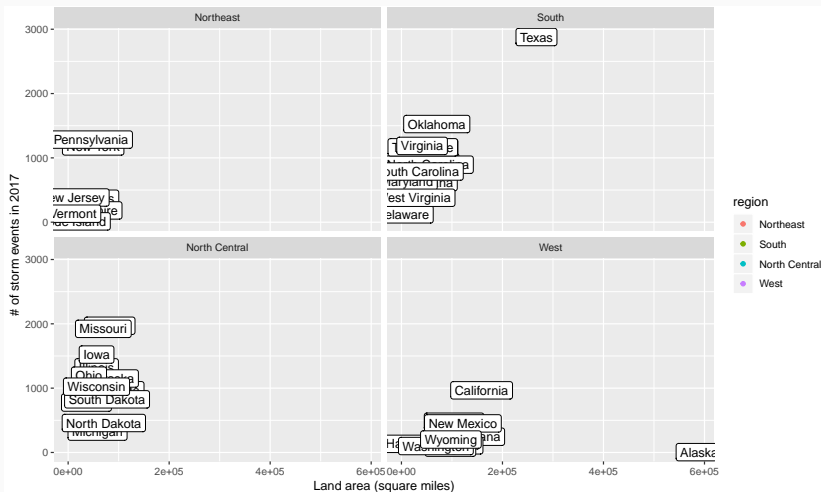


- Highlighting interesting points
- “Repelling” text labels
- Arranging plots

# ggplot2 extensions: repelling text labels

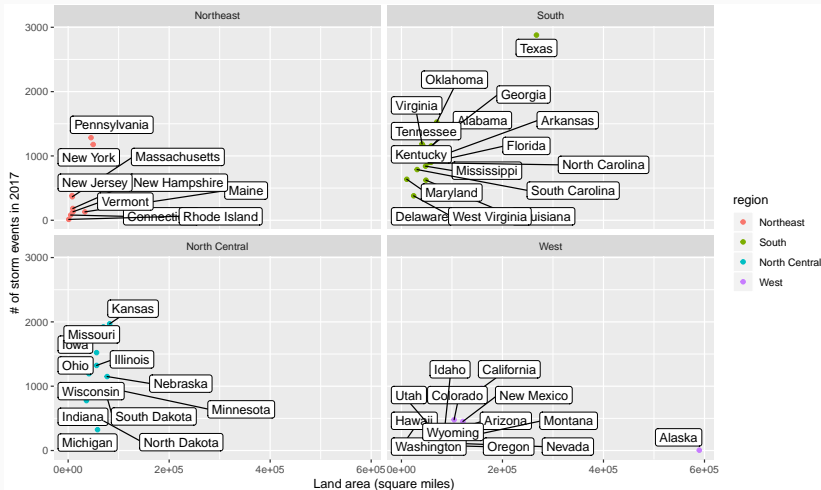
When you add labels to points on a plot, they often overlap:

```
storm_plot + facet_wrap(~ region) +  
  geom_label(aes(label = state))
```



# ggplot2 extensions: repelling text labels

```
library(ggrepel)  
storm_plot + facet_wrap(~ region) +  
  geom_label_repel(aes(label = state))
```



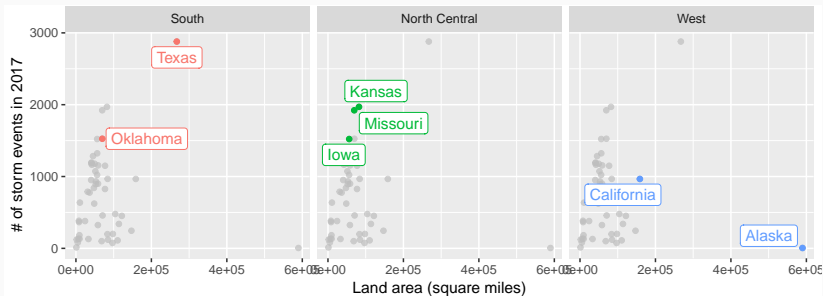


# gghighlight package

It may be too much to label every point. Instead, you may just want to highlight notable point.

You can use the `gghighlight` package to do that.

```
library(gghighlight)
storm_plot + facet_wrap(~ region) +
  gghighlight(area > 150000 | n > 1500, label_key = state)
```



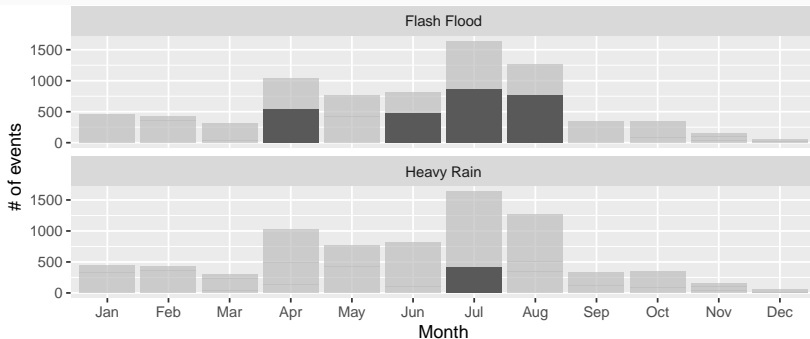
## gghighlight package

The gghighlight package also works for things like histograms. For example, you could create a dataset with the count by day-of-year of certain types of events:

```
storms_by_month <- storms_2017 %>%  
  filter(event_type %in% c("Flood", "Flash Flood", "Heavy Rain")) %>%  
  mutate(month = month(begin_date_time, label = TRUE)) %>%  
  group_by(month, event_type) %>%  
  count() %>%  
  ungroup()  
storms_by_month %>%  
  slice(1:4)  
  
## # A tibble: 4 x 3  
##   month event_type      n  
##   <ord> <chr>      <int>  
## 1 Jan   Flash Flood   113  
## 2 Jan   Flood          255  
## 3 Jan   Heavy Rain      80  
## 4 Feb   Flash Flood     65
```

# gghighlight package

```
ggplot(storms_by_month, aes(x = month, y = n, group = event_type)) +  
  geom_bar(stat = "identity") +  
  labs(x = "Month", y = "# of events") +  
  gghighlight(max(n) > 400, label_key = event_type) +  
  facet_wrap(~ event_type, ncol = 1)
```



## ggplot2 extensions: Arranging plots

You may have multiple related plots you want to have as multiple panels of a single figure.

There are a few packages that help with this. One very good one is `patchwork`.

You need to install this from GitHub:

```
devtools::install_github("thomasp85/patchwork")
```

Find out more: <https://github.com/thomasp85/patchwork#patchwork>

## ggplot2 extensions: Arranging plots

Say we want to plot seasonal patterns in events in the five counties with the highest number of events in 2017. We can use `dplyr` to figure out these counties:

```
top_counties <- storms_2017 %>%  
  group_by(fips, state, cz_name) %>%  
  count() %>%  
  ungroup() %>%  
  top_n(5, wt = n)
```

## ggplot2 extensions: Arranging plots

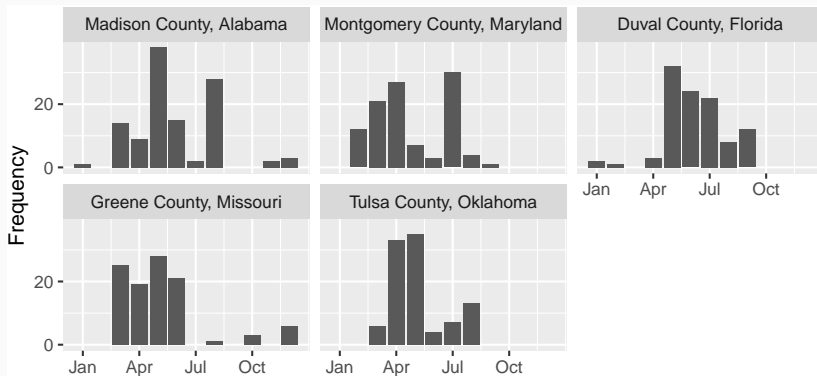
Then create a plot with the time patterns:

```
library(forcats)
top_counties_month <- storms_2017 %>%
  semi_join(top_counties, by = "fips") %>%
  mutate(month = month(begin_date_time),
         county = paste(cz_name, " County, ", state, sep = "")) %>%
  count(county, month) %>%
  ggplot(aes(x = month, y = n)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ fct_reorder(county, n, .fun = sum, .desc = TRUE), nrow = 2) +
  scale_x_continuous(name = "", breaks = c(1, 4, 7, 10),
                    labels = c("Jan", "Apr", "Jul", "Oct")) +
  scale_y_continuous(name = "Frequency", breaks = c(0, 20))
```

# ggplot2 extensions: Arranging plots

Here's this plot:

```
top_counties_month
```



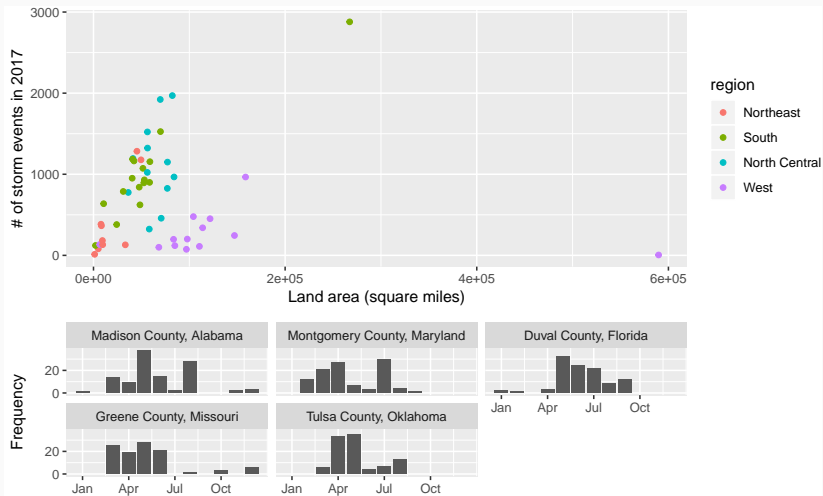
## ggplot2 extensions: Arranging plots

Now that you have two ggplot objects (`storm_plot` and `top_counties_month`), you can use `patchwork` to put them together:

```
library(patchwork)
storm_plot +
  top_counties_month +
  plot_layout(ncol = 1, heights = c(2, 1))
```



# ggplot2 extensions: Arranging plots

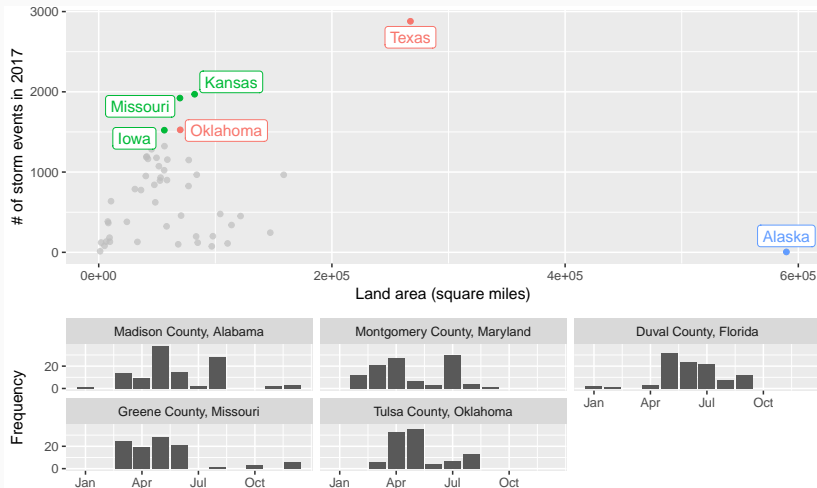


## ggplot2 extensions: Arranging plots

A slightly fancier version:

```
(storm_plot + theme(legend.position = "top") +  
  gghighlight(n > 1500 | area > 200000,  
    label_key = state)) +  
top_counties_month +  
plot_layout(ncol = 1, heights = c(2, 1))
```

# ggplot2 extensions: Arranging plots



## ggplot2 extensions: Arranging plots

Other packages for arranging ggplot objects:

- `gridExtra`
- `cowplot`

We'll take a break now to do the second part of the in-course exercise.

# Heat maps

**Heat maps** are helpful for exploring the measurements for many variables, including how similar they are, by showing small boxes of color.

In `ggplot2`, there is a `geom_tile` that is useful for creating heatmaps.

# Heat maps

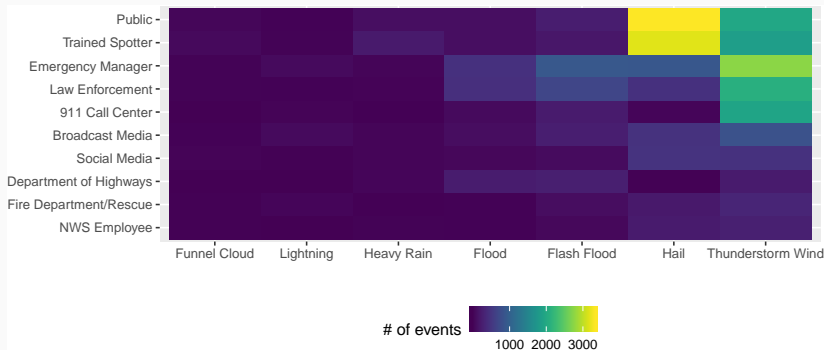
For example, we might want to look at how many events of each type were reported by each type of source. First, we can summarize the data to get this data:

```
type_by_source <- storms_2017 %>%  
  filter(event_type %in% c("Thunderstorm Wind", "Heavy Rain",  
                           "Funnel Cloud", "Flash Flood",  
                           "Flood", "Hail", "Lightning")) %>%  
  filter(source %in% c("911 Call Center", "Broadcast Media",  
                       "Department of Highways",  
                       "Emergency Manager", "Fire Department/Rescue",  
                       "Law Enforcement", "NWS Employee", "Public",  
                       "Social Media", "Trained Spotter")) %>%  
  group_by(event_type, source) %>% count() %>% ungroup()  
type_by_source %>% slice(1:3)
```

```
## # A tibble: 3 x 3  
##   event_type source          n  
##   <chr>      <chr>      <int>  
## 1 Flash Flood 911 Call Center    238  
## 2 Flash Flood Broadcast Media  268  
## 3 Flash Flood Department of Highways 276
```

# Heat maps

```
library(viridis); library(forcats)
ggplot(type_by_source,
       aes(x = fct_reorder(event_type, n, .fun = sum),
          y = fct_reorder(source, n, .fun = sum))) +
  geom_tile(aes(fill = n)) +
  scale_fill_viridis() +
  labs(x = "", y = "", fill = "# of events") +
  theme(legend.position = "bottom")
```

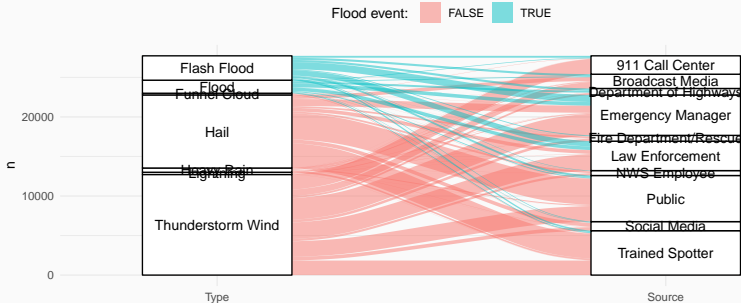




# Alluvial plots

The package `ggalluvial` allows you to make **alluvial plots**.

```
library(ggalluvial)
ggplot(data = type_by_source,
       aes(axis1 = event_type, axis2 = source, y = n)) +
  scale_x_discrete(limits = c("Type", "Source"), expand = c(.1, .05)) +
  geom_alluvium(aes(fill = event_type %in% c("Flash Flood", "Flood"))) +
  theme_minimal() + geom_stratum() +
  geom_text(stat = "stratum", label.strata = TRUE) +
  theme(legend.position = "top") + labs(fill = "Flood event: ")
```



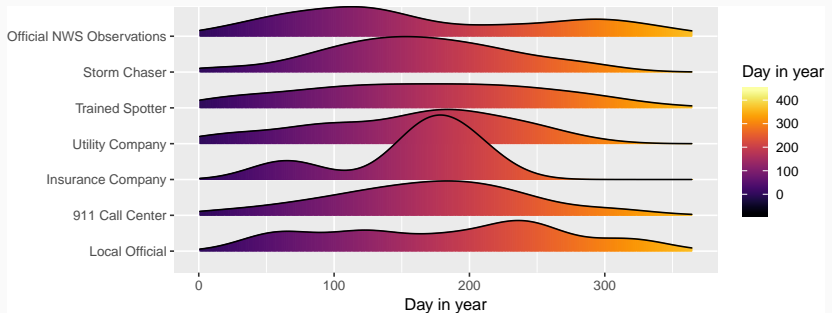
The ggribes package lets you plot the distribution of several levels of a factor across another variable.

Create a dataframe with the timing versus the source of the storm:

```
library(ggridges)
library(forcats)
storms_by_yday <- storms_2017 %>%
  filter(source %in% c("River/Stream Gages", "Insurance Company",
                      "Airline Pilot", "Trained Spotter",
                      "Storm Chaser", "Local Official",
                      "Official NWS Observations", "Utility Company",
                      "PostOffice", "911 Call Center")) %>%
  mutate(yday = yday(begin_date_time)) %>%
  group_by(yday, source) %>%
  count() %>%
  ungroup()
```

# ggridges package

```
library(viridis)
ggplot(storms_by_yday, aes(x = yday,
                           y = fct_reorder(source, yday,
                                             .fun = median, .desc = TRUE),
                           fill = ..x..)) +
  geom_density_ridges_gradient() +
  scale_fill_viridis(name = "Day in year", option = "B") +
  xlim(c(0, 365)) +
  labs(x = "Day in year", y = "")
```



# Mapping in the tidyverse

---

sf objects: “Simple features”

- R framework that is in active development
- There will likely be changes in the near future
- Plays very well with tidyverse functions, including `dplyr` and `ggplot2` tools

# Coordinate reference system

## Basics of creating an sf object



## Mapping an sf object

## Getting U.S. geography for `sf` objects

## Building maps with multiple components



## dplyr tricks with sf objects

## Putting it all together

---

## Putting it all together

- Writing a function
- Applying it across many facets of the data
- Plotting or mapping the result of the analysis