Cheatsheet 1

components of graphics
- a frame/coordinate system
- x-axis and y-axis specifications and etc.
- a layer
- geometric elements such as lines and points
- each type is a separate layer
- scales
- aesthetics added such as colors, sizes and shapes
- faceting
- splitting up of the data into multiple subplots
- a theme
- additional controls of the plot aesthetics such as font types, backgrounds and color scheme

- examine data properties using visualization
    - typical outcome
    - variability & range
    - shape
    - outliers
- univariate - data visualization

```
# only categorical x
# bar graph
ggplot(data, aes(x = x)) +
  geom_bar(color = "orange", fill = "blue")  +
  labs(x = "Rating", y = "Number of hikes") +
  theme_minimal()

# quantitative x
# histogram
# divide up the observed range of the variable into bins of equal width
# count up the number of cases that fall into each bin
ggplot(data, aes(x = x)) +
  geom_histogram(color = "white", binwidth = 5) +
  labs(x = "Elevation (feet)", y = "Number of hikes")

# quantitative x
# density plot
ggplot(data, aes(x = x)) +
  geom_density()
```

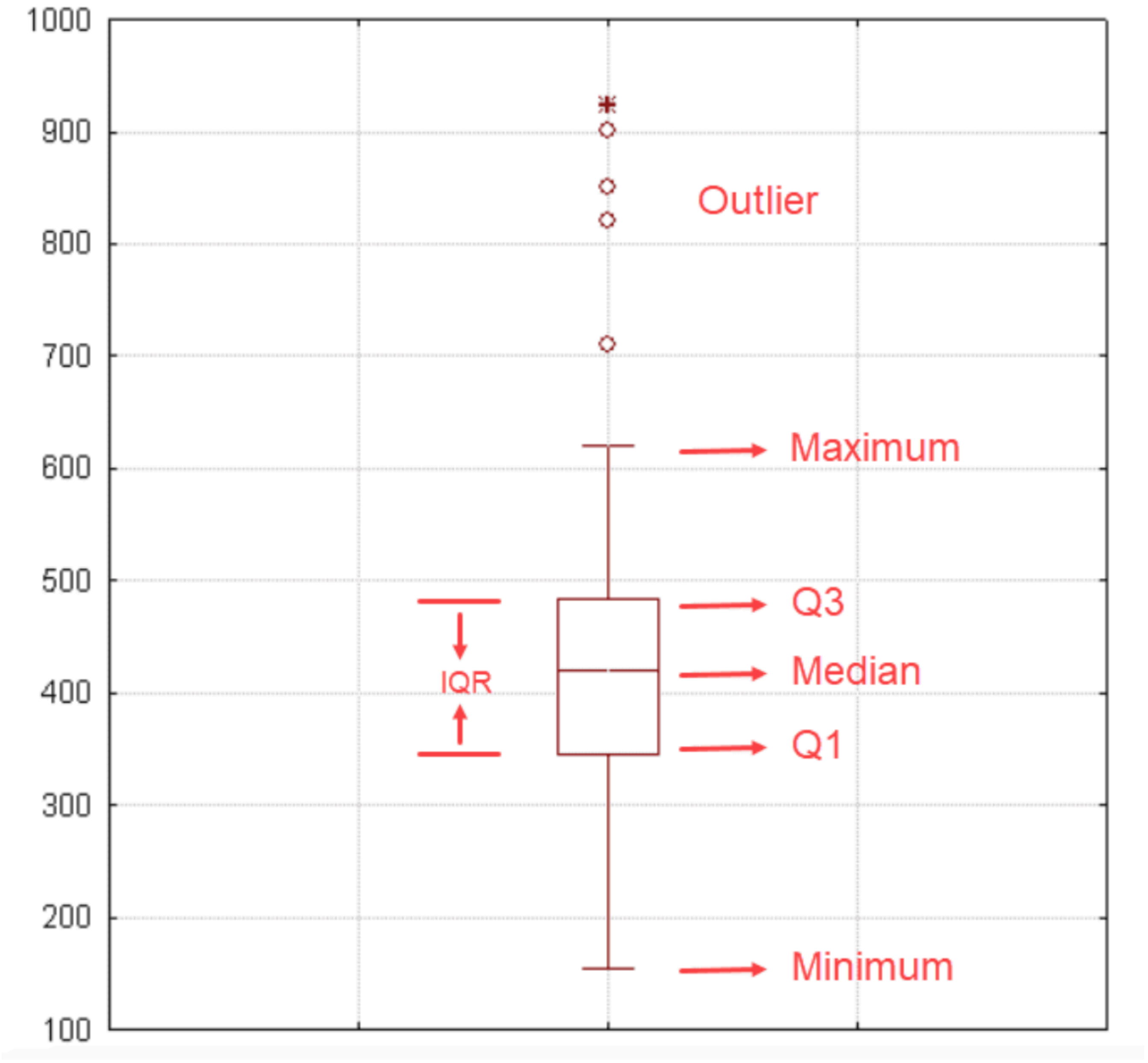
- bivariate - data visualization

```
# scatter plot
# numerical x
# numerical y
ggplot(data, aes(x=x, y=y)) +
geom_point()

# Scatter + trend line
ggplot(data, aes(x=x, y=y))+
# size sets the size of the points
# alpha sets transparency (1 = fully opaque, 0 = fully transparent)
geom_point(size = 2, alpha=0.7) +
# geom_smooth() adds a smoothed line to represent a trend using LOESS
# method = "lm" specifies that a linear model
# se = TRUE includes the shaded confidence interval around the regression line
# span = 0.5 controls 50% of the nearest points is used to compute the local
smooth fit
geom_smooth(method=”lm”,se=TRUE, span = 0.5)

# Line charts
# numerical x or data x
# numerical y
ggplot(data, aes(x=x, y=y)) +
geom_line()

# jitter plot
# alternative for scatter plots when categorical variables are involved
# categorical x
# numerical y
ggplot(data, aes(x = x, y = y)) +
# width sets jitter along x-axis (0 ~ infy)
# you can adjust both width and height
geom_jitter(width = 0.2)

# violin plot
# categorical x
# numerical y
ggplot(elections, aes(y = y, x = x)) +
geom_violin()

# box plot
# categorical x
# numerical y
ggplot(data, aes(x = x, y = y)) +
geom_boxplot()

# jitter plot
```

```r
# alternative for scatter plots when categorical variables are involved
# numerical x
# categorical y
ggplot(data, aes(x = x, y = y)) +
# height sets jitter along y-axis (0 ~ infy)
geom_jitter(height = 0.2) +
coord_flip()

# density plot
# The density plots are on top of each other
# numerical x
# categorical y
ggplot(data, aes(x = x, fill = y)) +
# alpha = 0.5 adds transparency
geom_density(alpha = 0.5) +
# scale_fill_manual defines what colors to use for the fill categories
scale_fill_manual(values = c("blue", "purple", "red")) +
# facet_wrap separates the density plots into facets for each category
facet_wrap(~ y)

# histogram plot
# The histogram plots are on top of each other
# numerical x
# categorical y
ggplot(data, aes(x = x, fill = y)) +
# color sets the color for frame
geom_histogram(color = "white") +
scale_fill_manual(values = c("blue", "purple", "red"))

# ridge plot
# numerical x
# categorical y
# used for y with many categories
# Install ggridges package
library(ggridges)
ggplot(data, aes(x = x, y = y)) +
geom_density_ridges()

# count plot
# categorical x
# categorical y
ggplot(data, aes(x = x, y = y)) +
geom_count()

# A side-by-side bar plot
# the same as count plot
# y-axis = counts
# categorical x
# categorical y
ggplot(data, aes(x = x, fill = y)) +
  geom_bar(position = "dodge")

# A stacked bar plot
# y-axis = counts
# categorical x
# categorical y
ggplot(data, aes(x = x, fill = y)) +
  geom_bar()

# A faceted bar plot
# categorical x
# categorical y
ggplot(data, aes(x = x)) +
  geom_bar() +
  facet_wrap(~ y)

# A proportional bar plot
# y-axis = percentage of each y category in x's category groups
# categorical x
# categorical y
ggplot(elections, aes(x = x, fill = y)) +
  geom_bar(position = "fill")
```

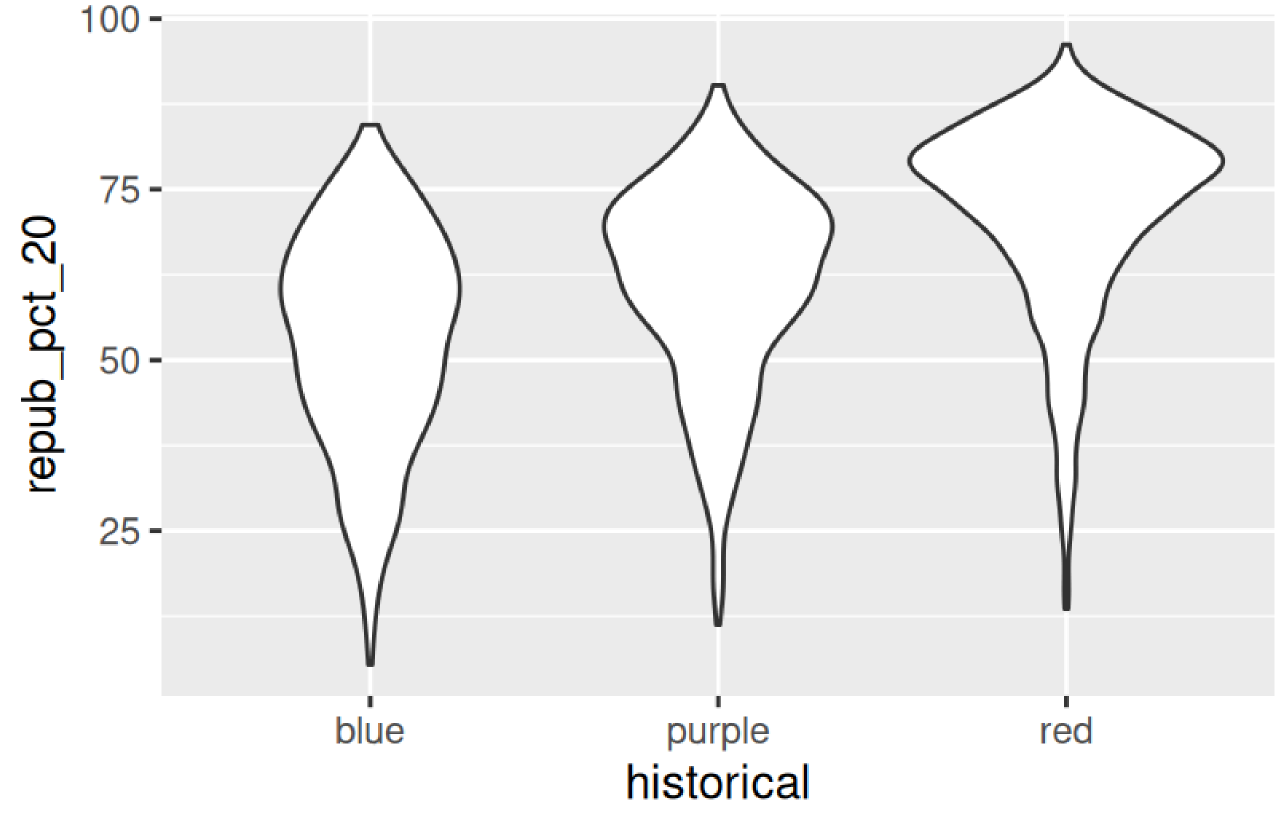
- box plot
    - interpretation
        - box: 50% of the data
        - median: central line

- example

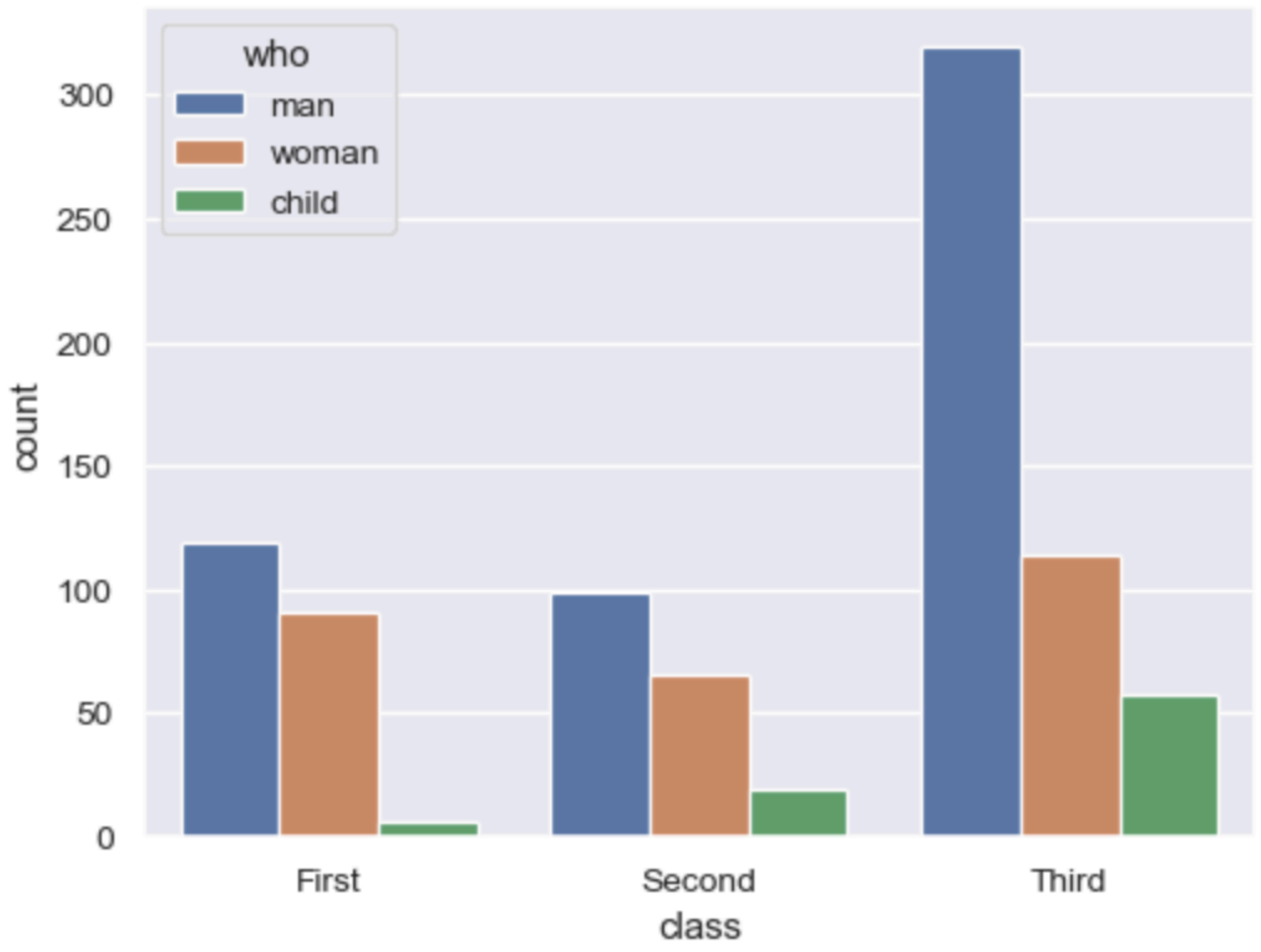


- violin plot

- example

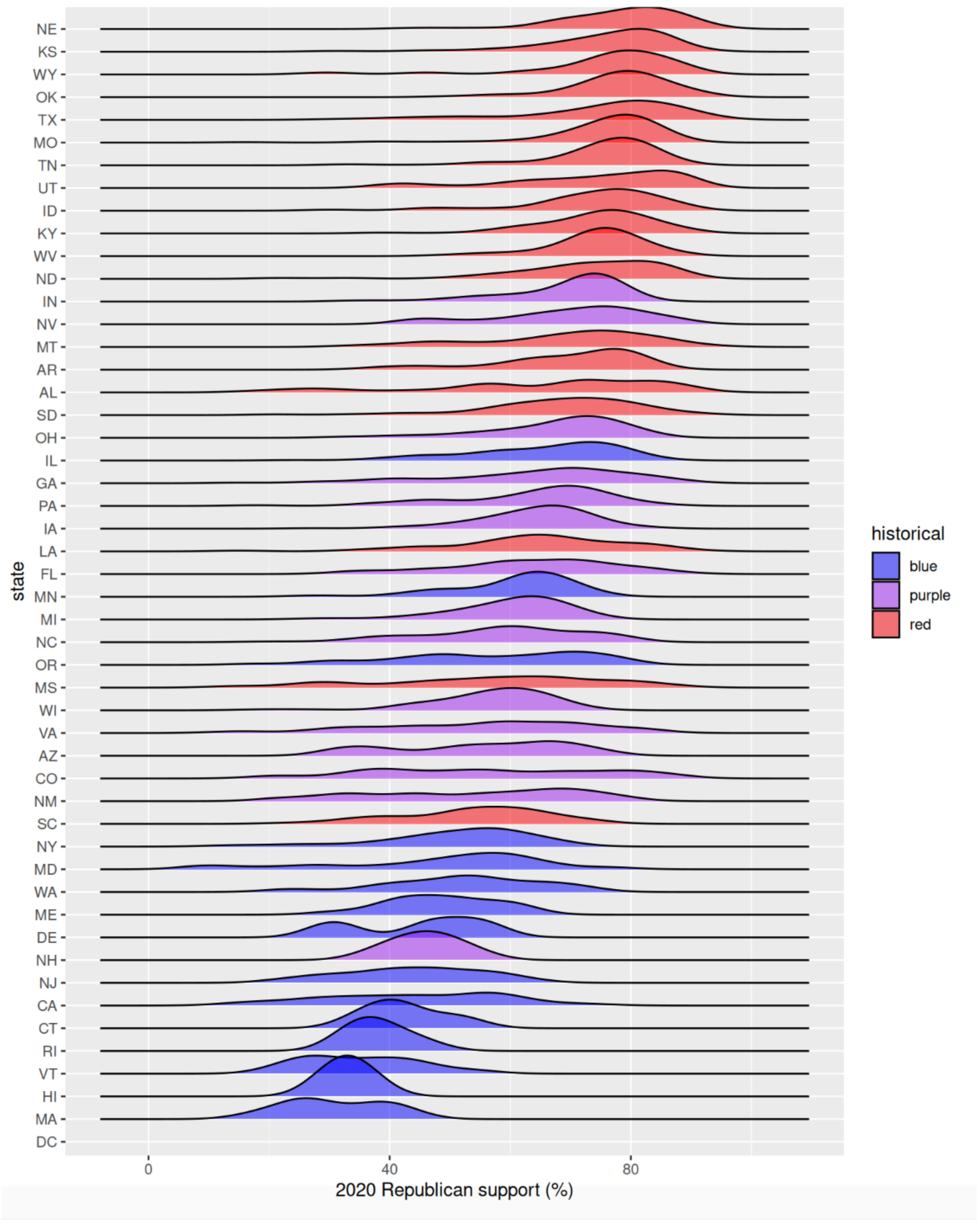


- count plot
  - example



- ridge plot

- example



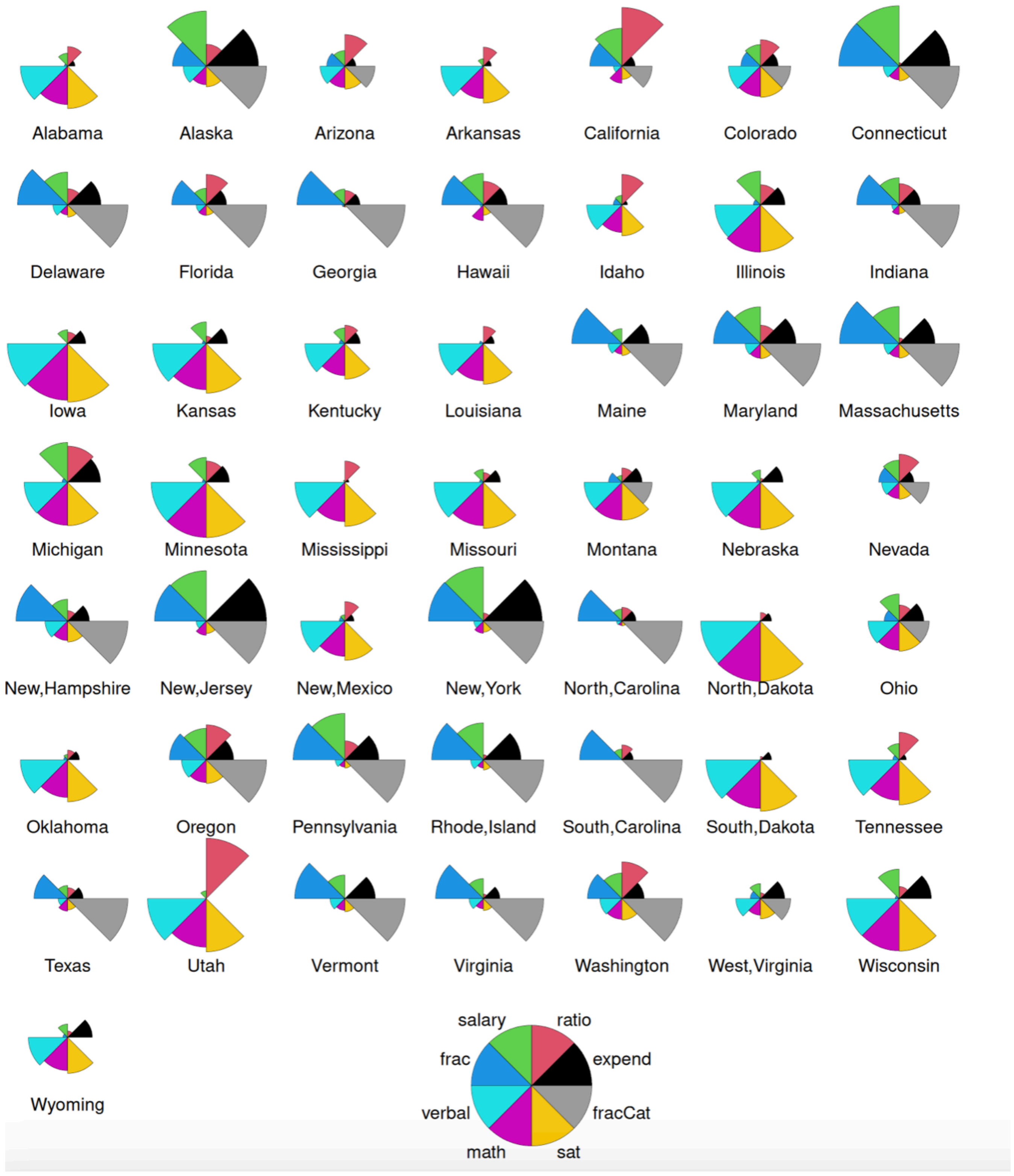
- multivariate - data visualization

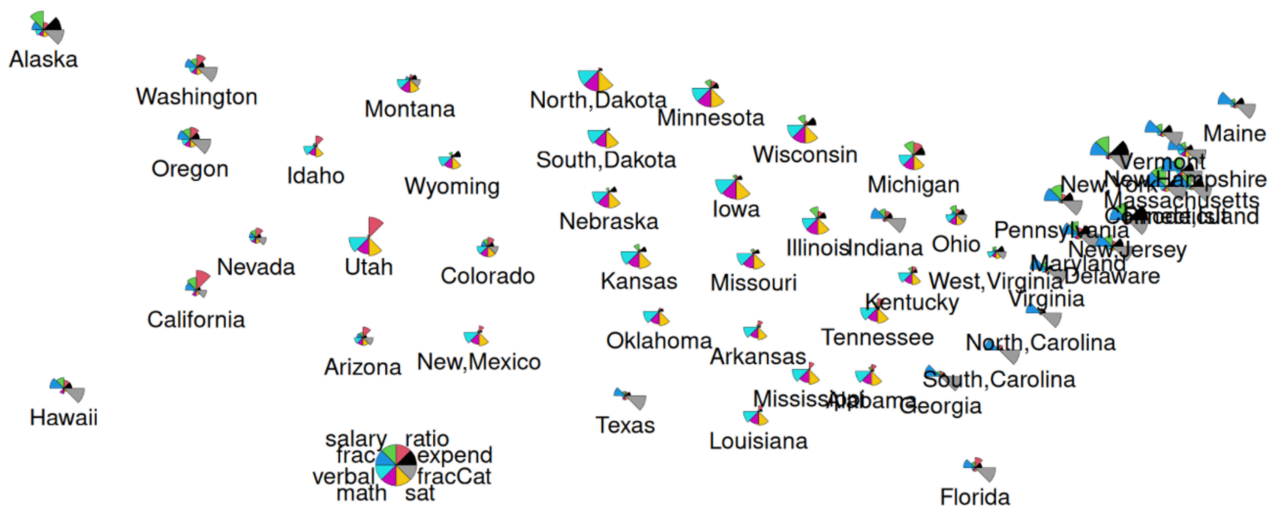
```
# ridge plot
# numerical x
# categorical y
# categorical z
# reorder y based on x
# desc. = TRUE sets the reorder from largest to smallest from top to bottom
# desc. = FALSE sets the reorder from smallest to largest from top to bottom
```

```
ggplot(data, aes(x = x, y = fct_reorder(y, x, desc. = TRUE), fill = z)) +
geom_density_ridges(alpha = 0.5) +
labs(y = "state", x = "2020 Republican support (%)") +
# note z have only 3 cateogories / colors
scale_fill_manual(values = c("blue", "purple", "red"))

# scatter plot
# numerical x
# numerical y
# categorical z
# applicable when x and y has the same scale and unit such as election info in
different years
ggplot(data, aes(x = x, y = fct_reorder(z, x))) +
geom_point(color = "red") +
geom_point(aes(x = y, y = z))

# scatter plot
# numerical x
# numerical y
# categorical g
# size is not a variable
# categorical sh
ggplot(data, aes(x=x, y=y, color=g, size=s, shape=sh)) +
geom_point()

# scatter plot + trend
# numerical x
# numerical y
# numerical z
# cut(z, 2) divides z variable into 2 groups and assign different colors for each
# note that there are 2 color groups thus 2 fitted lines
ggplot(data, aes(y = y, x = x, color = cut(z, 2))) +
  geom_point() +
  geom_smooth(se = FALSE, method = "lm")

# heatmap
# categorical x
# categorical y
# numerical z
ggplot(data, aes(x = x, y = y)) +
geom_tile(aes(fill = z)) +
scale_fill_gradient(low = "white",high = "red")

# heatmap
# scale the data before viz
# Load the gplots package needed for heatmaps
```

```
library(gplots)
# heatmap.2() = heatmap function
# data: a numerical matrix where rows and columns are heatmap axes and value
represented by color intensity
heatmap.2(data,
# dendrogram : tree-like diagram that shows hierachical clustering
# if = "row" cluster rows only
# if = "column" cluster columns only
# if = "both" cluster columns and rows
dendrogram = "none",
# Rowv = NA prevent reordering of rows based on hierachical clustering
# if = TRUE rows will be reordered
Rowv = NA,
# standardize each column
# if = "row" standardize each row
# if = "none" no standardization
scale = "column",
# adjust the size of the color key
keysize = 0.7,
# if = "density" it would shouw value density as a histogram
# if = "histogram" it would display a frequency histogram
density.info = "none",
# define the color scale for the heatmap
# hcl.colors(256) generates 256 gradient colors using the Hue-Chroma-Luminance
color model
col = hcl.colors(256),
# adjust margin around the heatmap
# 10 units of margin for row label
# 20 units of margin for column labels
margins = c(10, 20),
# colsep = c(1:7) adds vertical seperators after the first 7 columns
# rowsep = (1:50) adds horizontal seperators after the first 50 rows
# sepwidth = c(0.05, 0.05) defines the thinkness of column and row seperators
colsep = c(1:7), rowsep = (1:50), sepwidth = c(0.05, 0.05),
# sepcolor defines the color of the seperator
# trace = "none" removes trace lines
# if = "row" adds trace lines along rows
# if = "column" adds trace lines along columns
# if = "both" adds trace lines in both directions
sepcolor = "white", trace = "none"
)

# star plots
# data must only contain numerical scaled values
stars(data,
  flip.labels = FALSE,
  key.loc = c(10, 1.5),
  cex = 1,
  draw.segments = TRUE
)
```

- starplot
    - example





- map - data visualization
    - types
        - point maps
            - plotting locations of individual observations
        - contour maps
            - plotting the *density* or distribution of observations
        - choropleth maps
    - methods
        - ggplot() + geom_map()
            - Plot data points on top of a map
            - create choropleth maps
        - leaflet
            - create interactive maps
            - general steps
                - create a map widget
                - add a base map
                - add layers
                - print the map widgets to display it

```
# leaflet

# Load the leaflet package
library(leaflet)

# create a plotting frame (no map)
leaflet(data = fave_places)

# add the map background
leaflet(data = fave_places) |>
  addTiles()
```

```
# add map background and markers
# longitude and latitude refer to the variables in our data
leaflet(data = fave_places) |>
  addTiles() |>
  addMarkers(lng = ~longitude, lat = ~latitude)

# if we named them "longitude" and "latitude", the function automatically
recognizes these variables
leaflet(data = fave_places) |>
  addTiles() |>
  addMarkers()

# Load package needed to change color
library(gplots)

# We can add colored circles instead of markers at each location
leaflet(data = fave_places) |>
  addTiles() |>
# colors need to be in hex form
# col2hex is used to convert the color into the hex form
  addCircles(color = col2hex("red"))

leaflet(data = fave_places) |>
# can change the background
  addProviderTiles("USGS") |>
# weight sets the thinkness to make the lines and circles
  addCircles(weight = 10, opacity = 1, color = col2hex("yellow")) |>
# connect the dots, in their order in the dataset, with green lines
  addPolylines(
    lng = ~longitude,
    lat = ~latitude,
    color = col2hex("green")
  )
```

```
# ggplot

# Load the package
library(rnaturalearth)

# Get info about country boundaries across the world
# in a "sf" or simple feature format
# world maps
# the continent of Africa: ne_countries(continent = 'Africa', returnclass = 'sf')
# a set of countries: ne_countries(country = c('france', 'united kingdom',
'germany'), returnclass = 'sf')
# boundaries within a country: ne_states(country = 'united states of america',
```

```r
returnclass = 'sf')
world_boundaries <- ne_countries(returnclass = "sf")

# produce a basic map
# initialize a ggplot object
ggplot(world_boundaries) +
# adds a simle features layers for maps
  geom_sf()

# Load package needed to change map theme
library(mosaic)

# Add a point for each Starbucks
# NOTE: The Starbucks info is in our starbucks data, not world_boundaries
ggplot(world_boundaries) +
  geom_sf() +
  geom_point(
    data = starbucks,
    aes(x = Longitude, y = Latitude),
    alpha = 0.3, size = 0.2, color = "darkgreen"
  ) +
  theme_map()

ggplot(cma_boundaries) +
  geom_sf() +
  geom_point(
    data = starbucks_cma,
    aes(x = Longitude, y = Latitude),
    alpha = 0.3,
    size = 0.2,
    color = "darkgreen"
  ) +
      # restrict the x-axis  to the range between -179.14 and -50
  coord_sf(xlim = c(-179.14, -50)) +
  theme_map()

# create background maps of state and county level
# Load packages
library(sf)
library(maps)

# Get the boundaries
# extract county-level boundary data
# st_as_sf() converts the output into a simple feature (sf) object
# maps::map() uses the map function from the maps package
# fill = TRUE ensures that county boundaries are closed ploygons
# plot = FALSE prevents the function from automatically plotting the map
midwest_boundaries <- st_as_sf(maps::map("county",region = c("minnesota",
"wisconsin", "north dakota", "south dakota"), fill = TRUE, plot = FALSE))

# Get the latitude and longitude coordinates of state boundaries
states_map <- map_data("state")

# create choropleth maps
# map_id specifies which variable in our dataset indicates the region
# map_id and the region variable in our mapping background must have the same
possible outcomes in order to be matched up
# expand_limits assures that the map covers the entire area by pulling
longititudes and latitudes
ggplot(elections_by_state, aes(map_id = state_name, fill = repub_pct_20)) +
  geom_map(map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  theme_map()

ggplot(elections_by_state, aes(map_id = state_name, fill = repub_pct_20)) +
  geom_map(map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  theme_map() +
  scale_fill_gradientn(name = "% Republican", colors = c("blue", "purple", "red"),
values = scales::rescale(seq(0, 100, by = 5)))

ggplot(elections_by_state, aes(map_id = state_name, fill = repub_20_categories)) +
  geom_map(map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  theme_map()

# Load package needed for refining color palette
library(RColorBrewer)

# use the refined colors
ggplot(elections_by_state, aes(map_id = state_name, fill = repub_20_categories)) +
  geom_map(map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  theme_map() +
  scale_fill_manual(values = rev(brewer.pal(8, "RdBu")), name = "% Republican")



# add points
ggplot(elections_by_state, aes(map_id = state_name, fill = repub_20_categories)) +
  geom_map(map = states_map) +
  geom_point(
    data = starbucks_us,
    aes(x = Longitude, y = Latitude),
    size = 0.05,
    alpha = 0.2,
    inherit.aes = FALSE
  ) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  theme_map() +
  scale_fill_manual(values = rev(brewer.pal(8, "RdBu")), name = "% Republican")

ggplot(elections_by_counties, aes(map_id = county_fips, fill =
repub_20_categories)) +
  geom_map(map = county_map) +
  scale_fill_manual(values = rev(brewer.pal(10, "RdBu")), name = "% Republican") +
  expand_limits(x = county_map$long, y = county_map$lat) +
  theme_map() +
# moves the legend to the right
  theme(legend.position = "right") +
# ensures equal aspect ratio between xy axises
  coord_equal()

ggplot(elections_by_counties, aes(map_id = county_fips, fill = median_rent)) +
  geom_map(map = county_map) +
  expand_limits(x = county_map$long, y = county_map$lat) +
  theme_map() +
  theme(legend.position = "right") +
  coord_equal() +
  scale_fill_gradientn(name = "median rent", colors = c("white", "lightgreen",
"darkgreen"))

ggplot(elections_by_counties, aes(map_id = county_fips, fill = median_age)) +
  geom_map(map = county_map) +
  expand_limits(x = county_map$long, y = county_map$lat) +
  theme_map() +
  theme(legend.position = "right") +
  coord_equal() +
# scale_fill_gradientn() specifies a custom color gradient for a continuous
variable mapped to fill
# name = "median age" sets the legend title for the color scale
# colors = terrain.colors(10) uses 10 colors from the "terrain" color palette to
create a smooth gradient
  scale_fill_gradientn(name = "median age", colors = terrain.colors(10))
```

- addition - data visualization

```
# Faceting
# create seperate plots for each level of category
# scales=”free y” allows the y-axis scale to vary independently in each facet
facet_wrap(~ category, scales=”free y”)

# Faceting
# creates a grid layout
# scales=”free” allows both axises to adjust independently
facet_grid(row ~ column, scales=”free”)

# Customization
# title is at the top and caption is at the low end
labs(title=”Title”, x=”X-axis”, y=”Y-axis”, caption=”Source”)

# themes
# no background color
# light gray gridlines
# no box around the plto
# modern and clean appearance
theme_minimal()

# themes
# white background with no gridlines
# blakc axises
# no additional distractions
theme_classic()

# themes
# white background with black gridlines
# box around plot
theme_bw()

# swaps x and y axises
coord_flip()

# converts Cartesian coordinates to polar
# used to create pie charts, radical bar plots, and circular graphs
coord_polar()

# transform the x-axis to a logarithmic scale
scale_x_log10()

# restrict y-axis range between 0 and 100
scale_y_continuous(limits=c(0,100))

# applies a color palette from RColorBrewer package for discrete/cateforical
variables
scale_fill_brewer(palette=”Set1”)
```

```
# applies the viridis color scale for continous data
# viridis is colorblind-friendly
scale_color_viridis_c()

# applies the viridis color scale for discrete data
# viridis is colorblind-friendly
scale_color_viridis_d()

# add text annotate
annotate(”text”, x=5, y=20, label=”Point”)

# remove legend
theme(legend.position=”none”)
```