Homework 3. Interactive Visualization

Drashti Thummar

2023-03-17

```
library(data.table)
library(dplyr)
library(tidyr)
library(plotly)
library(lubridate)
library(maps)
library(ggplot2)
library(viridis)
```

In this homework you should use plotly unless said otherwise.

To create pdf version of your homework, knit it first to html and then print it to pdf. Interactive plotly plots can be difficult sometimes to convert to static images suitable for insertion to LaTex documents (that is knitting to PDF).

Look for questions in R-chunks as comments and plain text (they are prefixed as Q.).

Part 1. Iris Dataset. (20 points)

> "The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis" https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)

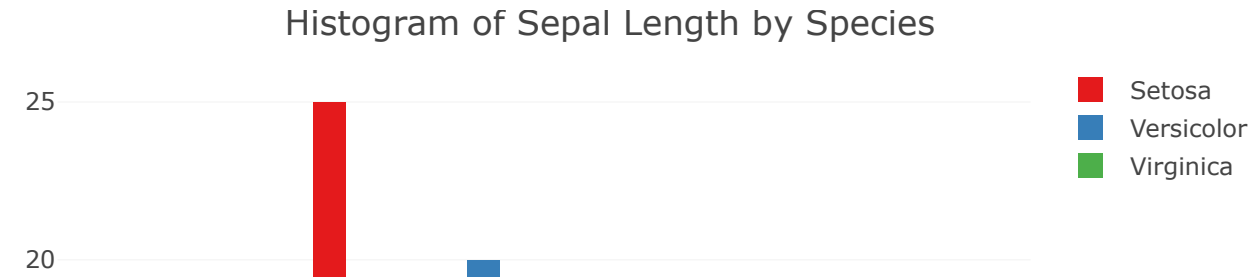
```
# Q1.1. Read the iris.csv file  (2 points)
# hint: use fread from data.table, it is significantly faster than default methods
#       be sure to have strings as factors (see stringsAsFactors argument)
df <- fread("Iris.csv")
```

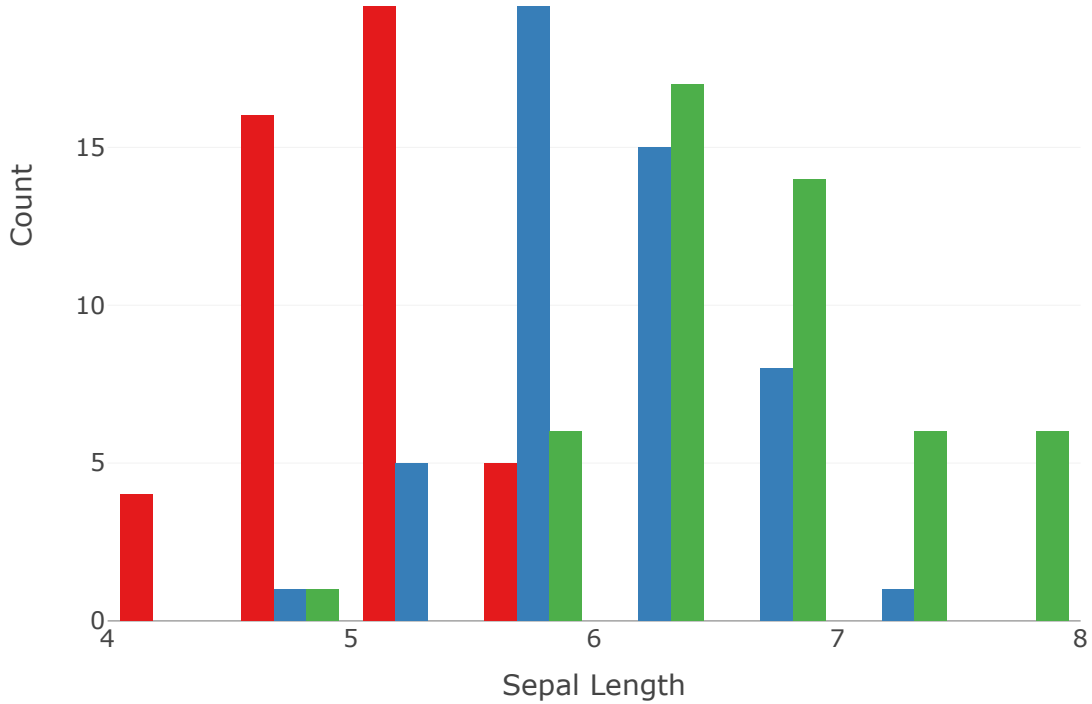
```
# Q1.2. Show some values from data frame (2 points)
head(df)
```

sepal.length <dbl>	sepal.width <dbl>	petal.length <dbl>	petal.width <dbl>	variety <chr>
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
4.6	3.1	1.5	0.2	Setosa
5.0	3.6	1.4	0.2	Setosa
5.4	3.9	1.7	0.4	Setosa

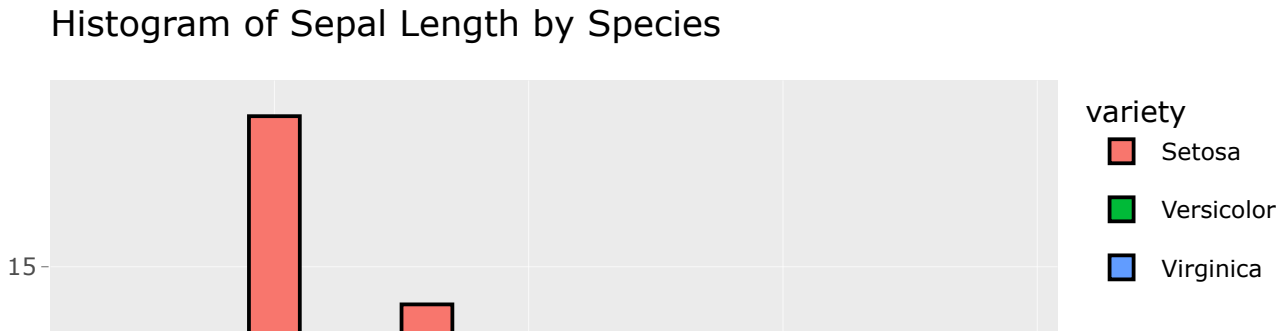
6 rows

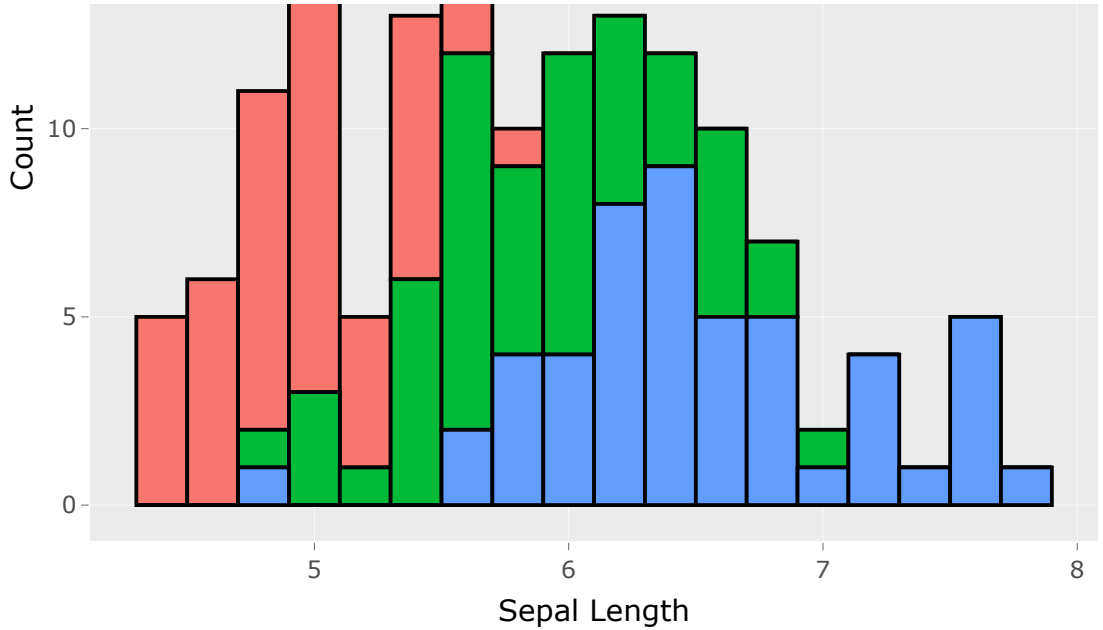
```
# Q1.3. Build histogram plot for Sepal.Length variable for each species using plot_ly
# (use color argument for grouping) (2 points)
# should be one plot
iris_grouped <- df %>% group_by(variety)

# Build the plot using plot_ly
plot_ly(data = iris_grouped, x = ~sepal.length, type = "histogram",
        color = ~variety, colors = "Set1") %>%
  layout(title = "Histogram of Sepal Length by Species",
         xaxis = list(title = "Sepal Length"),
         yaxis = list(title = "Count"))
```



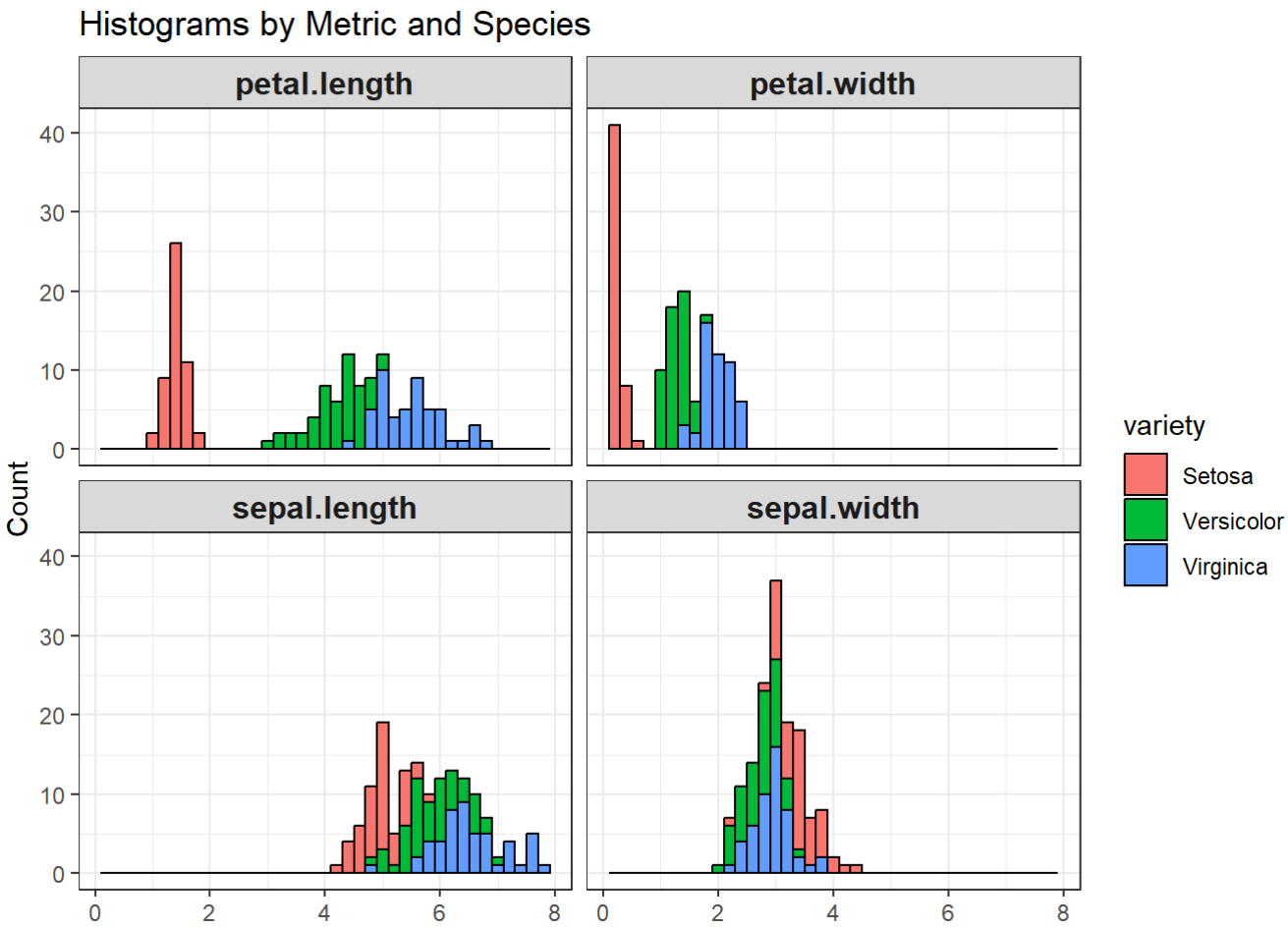



```
# Q1.4. Repeat previous plot with ggplot2 and convert it to plotly with ggplotly (2 points)
# Build the plot using ggplot2
p <- ggplot(df, aes(x = sepal.length, fill = variety)) +
  geom_histogram(color = "black", binwidth = 0.2) +
  labs(title = "Histogram of Sepal Length by Species", x = "Sepal Length", y = "Count")

# Convert ggplot2 plot to plotly
ggplotly(p)
```





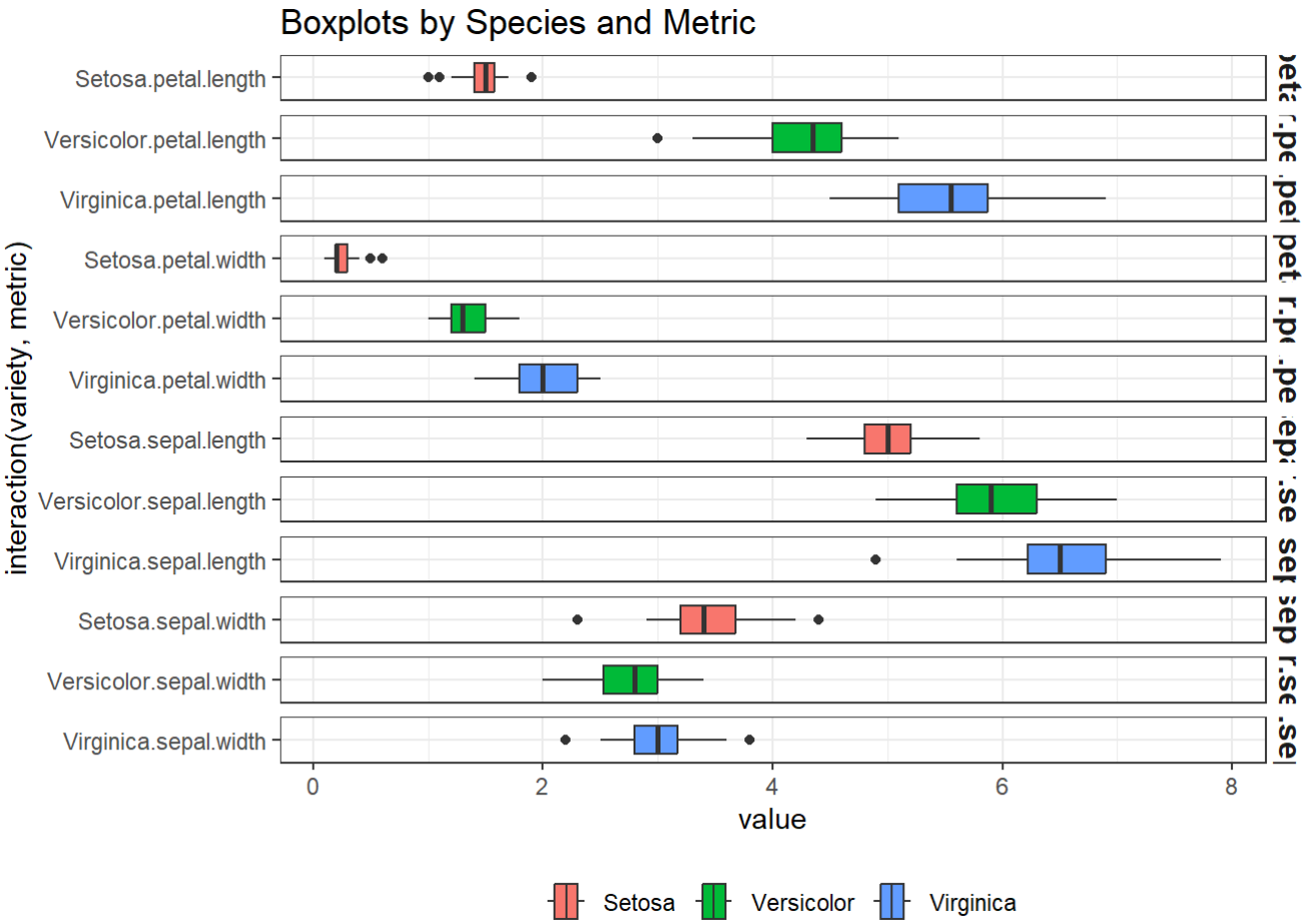

```
# Q1.5. Create facet 2 by 2 plot with histograms similar to previous but for each metric
# (2 points)
# hint:
#   following conversion to long format can be useful:
#   iris %>% gather(key = "metric", value = "value",-Species)
#
iris_long <- df %>% gather(key = "metric", value = "value", -variety)

# Build the plot using ggplot2
ggplot(iris_long, aes(x = value, fill = variety)) +
  geom_histogram(color = "black", binwidth = 0.2) +
  facet_wrap(~metric, nrow = 2, ncol = 2) +
  labs(title = "Histograms by Metric and Species", x = NULL, y = "Count") +
  theme_bw() +
  theme(strip.text = element_text(size = 12, face = "bold"))
```



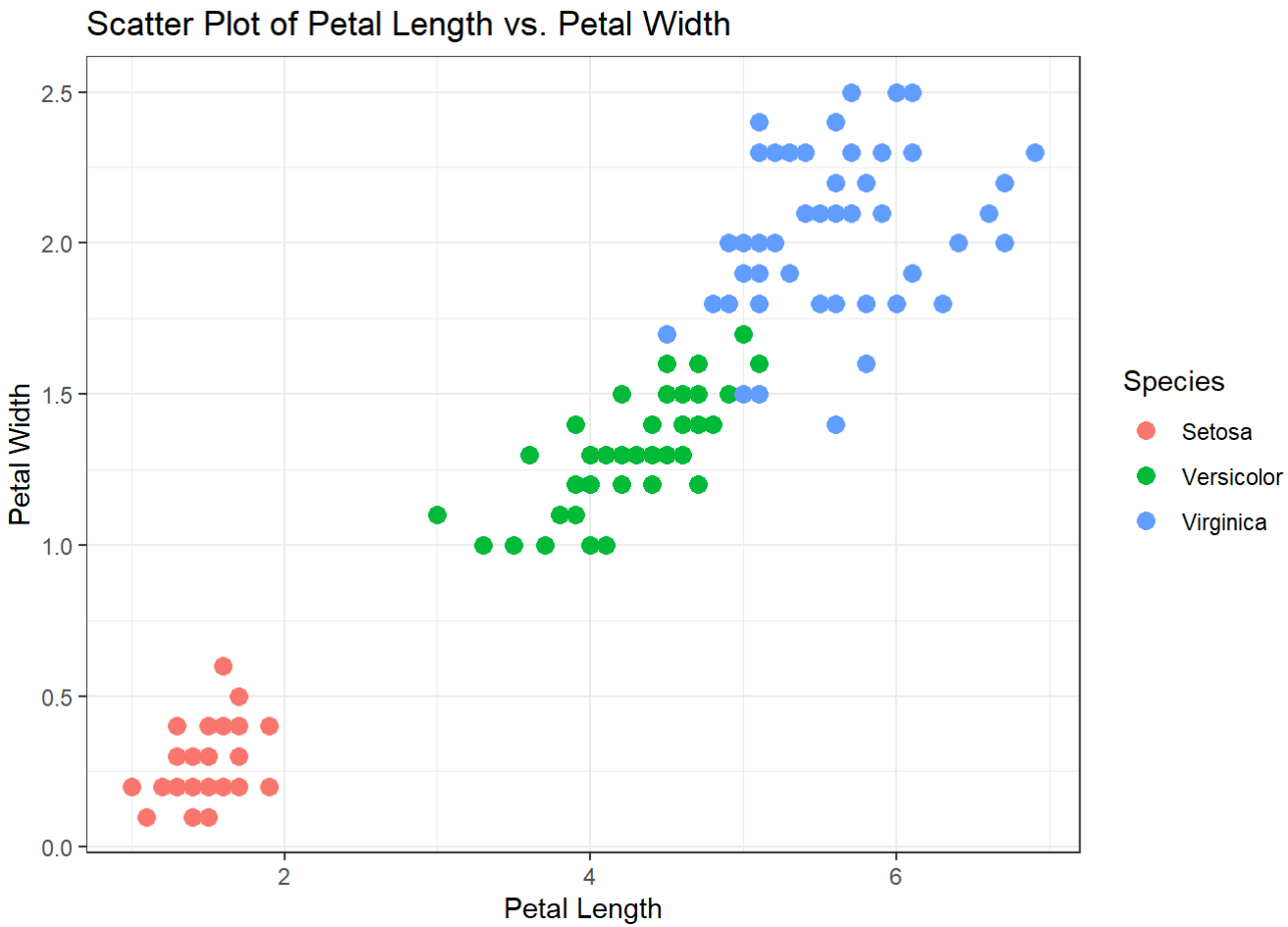
Q1.6. Which metrics has best species separations? (2 points)

–> From the above graph, it can be clearly seen that matric with column ‘petal.length’ has the best species separations.

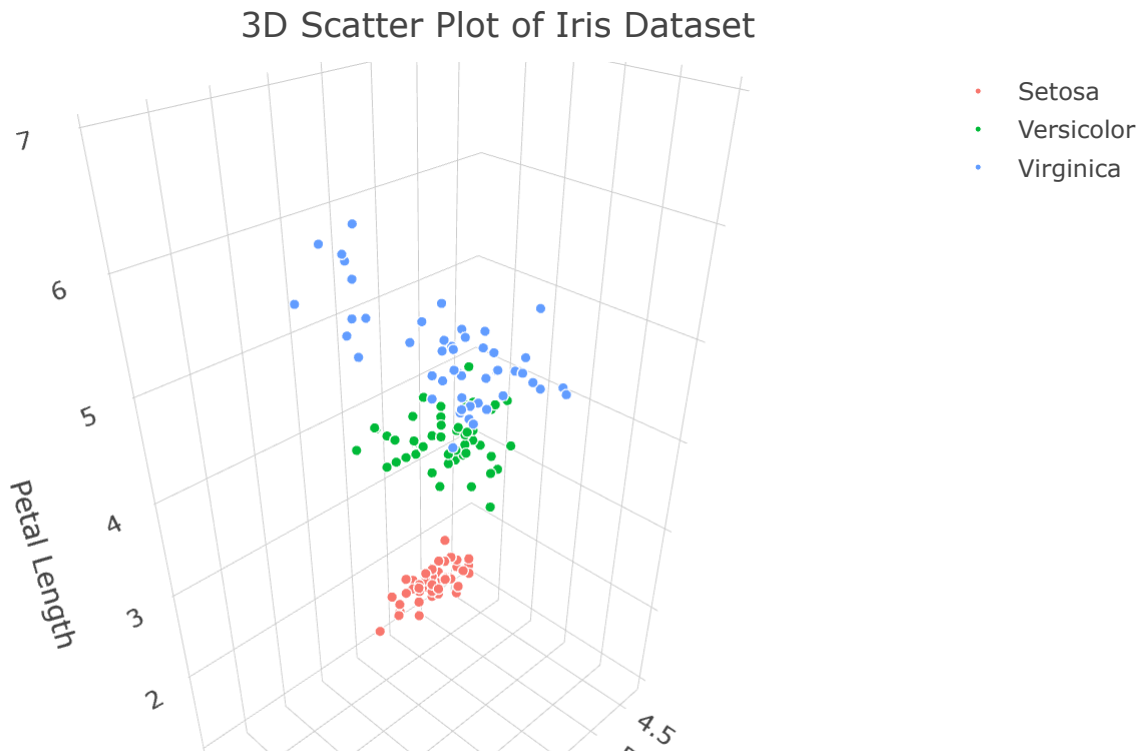
```
# Q1.7. Repeat above plot but using box plot (2 points)
iris_long <- df %>% gather(key = "metric", value = "value", -variety)

# Build the plot using ggplot2
ggplot(iris_long, aes(x = value, y = interaction(variety, metric), fill = variety)) +
  geom_boxplot() +
  facet_grid(interaction(variety, metric) ~ ., scales = "free_y") +
  labs(title = "Boxplots by Species and Metric") +
  theme_bw() +
  theme(strip.text = element_text(size = 12, face = "bold"),
        strip.background = element_blank(),
        legend.position = "bottom",
        legend.title = element_blank())
```




```
# Q1.8. Choose two metrics which separates species the most and use it to make scatter plot
# color points by species (2 points)
# Create a scatter plot of petal length and petal width, colored by species
ggplot(df, aes(x = petal.length, y = petal.width, color = variety)) +
  geom_point(size = 3) +
  labs(title = "Scatter Plot of Petal Length vs. Petal Width",
       x = "Petal Length", y = "Petal Width", color = "Species") +
  theme_bw()
```


Scatter Plot of Petal Length vs. Petal Width
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
2
4
6
Species
Setosa
Versicolor
Virginica

```
# Q1.9. Choose three metrics which separates species the most and use it to make 3d plot
# color points by species (2 points)

# Create a 3D scatter plot of sepal length, petal width, and petal length, colored by species
fig <- plot_ly(data = df, x = ~sepal.length, y = ~petal.width, z = ~petal.length,
               color = ~variety, colors = c("#F8766D", "#00BA38", "#619CFF"),
               type = "scatter3d", mode = "markers",
               marker = list(size = 3, symbol = "circle", line = list(width = 0.5, color = "white")))

# Add axis labels and title
fig <- fig %>% layout(scene = list(xaxis = list(title = "Sepal Length"),
                                   yaxis = list(title = "Petal Width"),
                                   zaxis = list(title = "Petal Length")),
                      title = list(text = "3D Scatter Plot of Iris Dataset"))

# Print the plot
fig
```



Q1.10. Comment on species separation (2 points): –> From the scatter plot of petal width vs. petal length shows a clear separation between the three species.In which Setosa has the smallest petals, Versicolor has medium-sized petals, and Virginica has the largest petals.

Similarly, a scatter plot of sepal length vs. petal length vs petal width also shows a clear separation between the species, with Setosa having the shortest petals and Virginica having the longest petals.

Part 2. Covid-19 Dataset. (20 points)

Download us-states.csv (https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv) (there is also a copy in homework assignment) from https://github.com/nytimes/covid-19-data/ (https://github.com/nytimes/covid-19-data/). README.md (https://github.com/nytimes/covid-19-data/blob/master/README.md) for details on file content.

```
# Q2.1. Read us-states.csv (2 points)
dframe <- fread('us-states.csv')
```

```
# Q2.2. Show some values from dataframe
head(dframe)
```

date	state	fips	cases	deaths
<IDate>	<chr>	<int>	<int>	<int>
2020-01-21	Washington	53	1	0
2020-01-22	Washington	53	1	0
2020-01-23	Washington	53	1	0
2020-01-24	Illinois	17	1	0
2020-01-24	Washington	53	1	0

date	state	fips	cases	deaths
<IDate>	<chr>	<int>	<int>	<int>
2020-01-25	California	6	1	0

6 rows

```
# Q2.3. Create new dataframe with new cases per month for each state (2 points)
# hint:
#    is cases column cumulative or not cumulative?

# Group the data by state and month
grouped_data <- dframe %>%
  group_by(state, month = format(date, "%Y-%m")) %>%
  summarize(cases = max(cases) - min(cases))
```

```
## `summarise()` has grouped output by 'state'. You can override using the
## `.groups` argument.
```

```
# Rename the state column to State
grouped_data <- rename(grouped_data, State = state)

# Reorder the columns as required
grouped_data <- select(grouped_data, month, State, cases)

# Print the final dataframe
head(grouped_data)
```

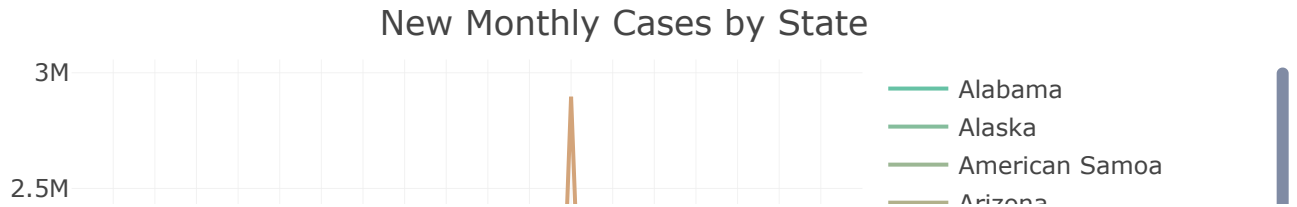
month	State	cases
<chr>	<chr>	<int>
2020-03	Alabama	993
2020-04	Alabama	5960
2020-05	Alabama	10658

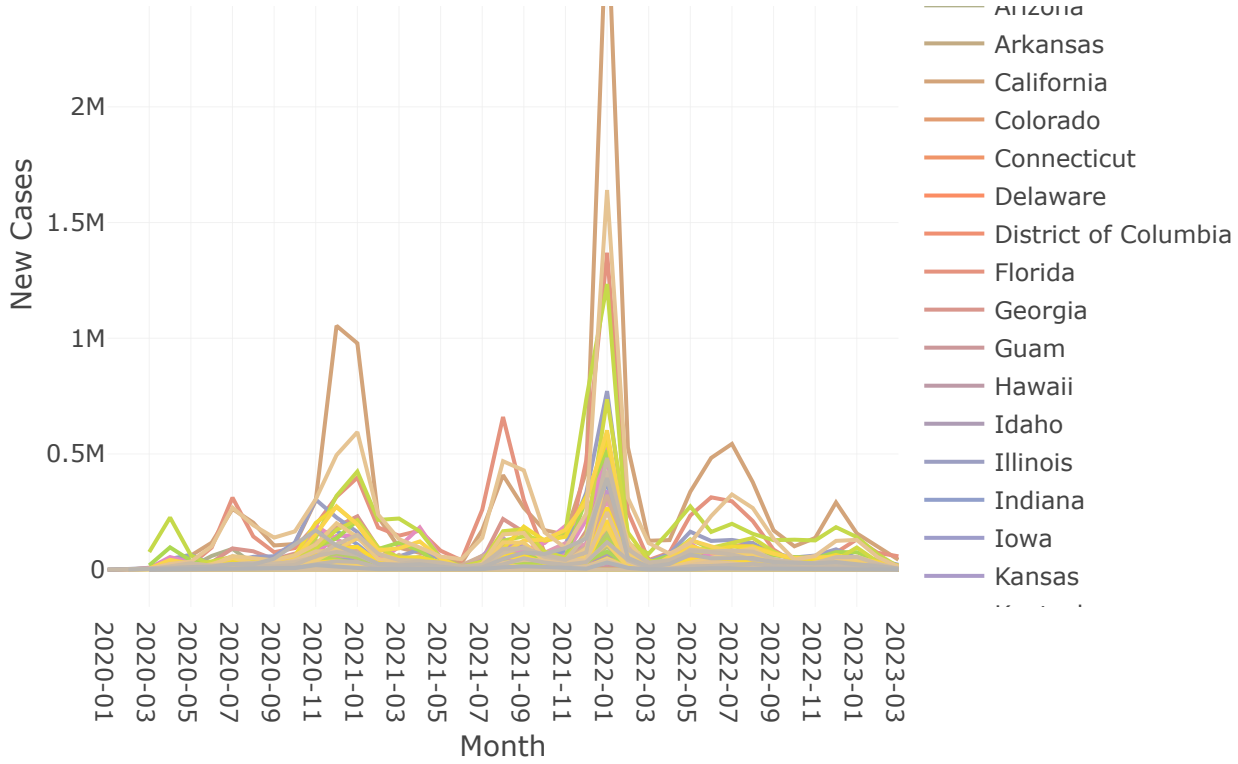
month <chr>	State <chr>	cases <int>
2020-06	Alabama	19511
2020-07	Alabama	48761
2020-08	Alabama	36709

6 rows

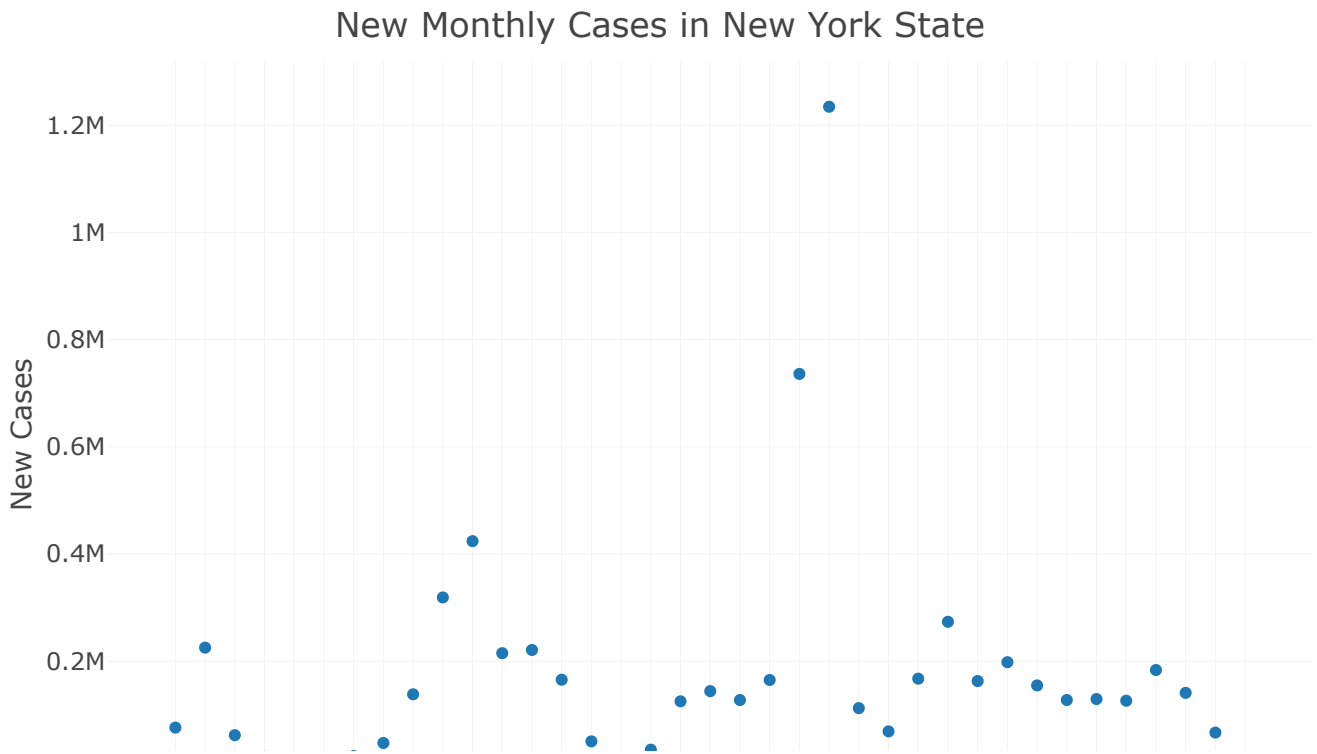
```
# Q2.4.Using previous dataframe plot new monthly cases in states, group by states
# The resulting plot is busy, use interactive plotly capabilities to limit number
# of displayed states
# (2 points)

# Plot the new monthly cases by state using Plotly
plot_data <- plot_ly(grouped_data, x = ~month, y = ~cases, color = ~State,
                     type = "scatter", mode = "lines") %>%
  layout(title = "New Monthly Cases by State", xaxis = list(title = "Month"),
         yaxis = list(title = "New Cases"))

# Limit the number of displayed states using Plotly's filter function
plot_data %>% filter(State %in% c("California", "Texas", "Alabama", "New York",
                                  "Pennsylvania", "Illinois", "Ohio")) %>%
  config(displayModeBar = FALSE)
```

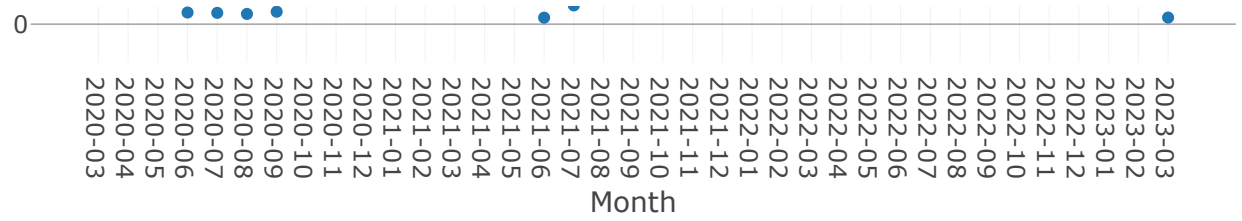
```
## Warning in RColorBrewer::brewer.pal(N, "Set2"): n too large, allowed maximum for palette Set2 is 8
## Returning the palette you asked for with that many colors

## Warning in RColorBrewer::brewer.pal(N, "Set2"): n too large, allowed maximum for palette Set2 is 8
## Returning the palette you asked for with that many colors
```




Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Guam
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
New Cases
2M
1.5M
1M
0.5M
0
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09
2022-11
2023-01
2023-03
Month

```
# Q2.5.Plot new monthly cases only in NY state
# (2 points)

# Filter the dframe for NY state only
ny_dframe <- dframe %>%
  filter(state == "New York") %>%
  group_by(month = format(date, "%Y-%m")) %>%
  summarize(cases = max(cases) - min(cases))

# Plot the new monthly cases for NY state using Plotly
plot_data <- plot_ly(ny_dframe, x = ~month, y = ~cases, type = "scatter",
                     mode = "markers") %>%
  layout(title = "New Monthly Cases in New York State", xaxis = list(title = "Month"),
         yaxis = list(title = "New Cases"))

# Display the interactive Plotly chart
plot_data
```






```
# Q2.6. Found the year-month with highest cases in NY state
# (2 points)

# Group rows by year-month and return the last date's value for each group
df_monthly <- dframe %>%
  filter(state == "New York") %>%
  group_by(year_month = format(date, "%Y-%m")) %>%
  summarize(date = max(date),
            state = unique(state),
            fips = unique(fips),
            cum_cases = last(cases),
            cum_deaths = last(deaths)) %>%
  ungroup()

# Calculate the highest_cases column
df_monthly$highest_cases <- sapply(seq_len(nrow(df_monthly)), function(i) {
  if (i == 1) {
    df_monthly$cum_cases[i]
  } else {
    max(df_monthly$cum_cases[i] - df_monthly$cum_cases[i-1])
  }
})

# Select and reorder columns
df_result <- df_monthly %>%
  select(state, date, fips, cum_cases, cum_deaths, highest_cases) %>%
  slice(which.max(highest_cases))

# Print the resulting dataframe
df_result
```

state <chr>	date <IDate>	fips <int>	cum_cases <int>	cum_deaths <int>	highest_cases <int>
New York	2022-01-31	36	4789532	64247	1315562

1 row

```
# Q2.7. Plot new cases in determined above year-month
# using USA state map, color each state by number of cases  (3 points)
# hint:
#   there two build in constants in R: state.abb and state.name
#   to convert full name to abbreviation


specific_month <- "2022-01"

grouped_data <- dframe %>%
  group_by(state, month = format(date, "%Y-%m")) %>%
  summarize(cases = max(cases) - min(cases))
```

```
## `summarise()` has grouped output by 'state'. You can override using the
## `.groups` argument.
```

```
df <- grouped_data %>% filter(month == specific_month)
head(df)
```

state <chr>	month <chr>	cases <int>
Alabama	2022-01	321643
Alaska	2022-01	59513
American Samoa	2022-01	7
Arizona	2022-01	480936

state <chr>	month <chr>	cases <int>
Arkansas	2022-01	206118
California	2022-01	2896206

6 rows

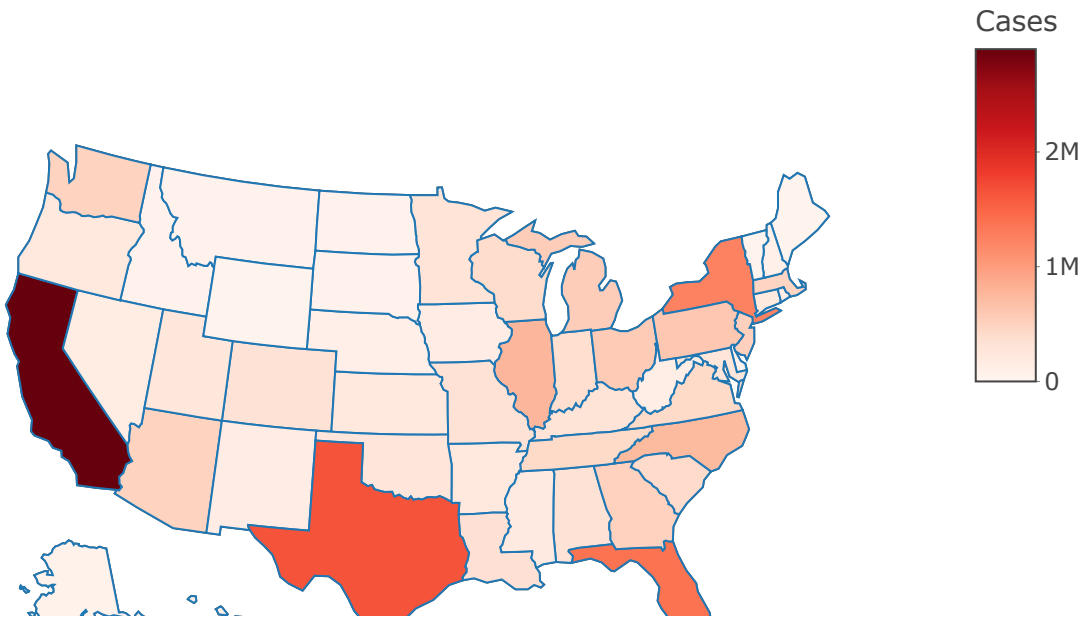
```
# convert full names to abbreviated names
abbreviated_states <- sapply(df$state, function(state_name) state.abb[match(state_name, state.name)])

# replace full names with abbreviated names in data frame
df$state <- abbreviated_states

# print data frame with abbreviated names
head(df)
```

state <chr>	month <chr>	cases <int>
AL	2022-01	321643
AK	2022-01	59513
NA	2022-01	7
AZ	2022-01	480936
AR	2022-01	206118
CA	2022-01	2896206

6 rows

```
plot_ly(locations = df$state,
        z = df$cases,
        type = "choropleth",
        locationmode = "USA-states",
        colorbar = list(title = "Cases"),
        colors = "Reds",
        scope = "usa")%>%
        layout(geo = list(scope = "usa"))
```

```
## Warning: 'choropleth' objects don't have these attributes: 'scope'
## Valid attributes include:
## 'autocolorscale', 'coloraxis', 'colorbar', 'colorscale', 'customdata', 'customdatasrc', 'featureidkey', 'geo', 'geojson',
'hoverinfo', 'hoverinfosrc', 'hoverlabel', 'hovertemplate', 'hovertemplatesrc', 'hovertext', 'hovertextsrc', 'ids', 'idssr
c', 'legendgroup', 'legendgrouptitle', 'legendrank', 'locationmode', 'locations', 'locationssrc', 'marker', 'meta', 'metasr
c', 'name', 'reversescale', 'selected', 'selectedpoints', 'showlegend', 'showscale', 'stream', 'text', 'textsrc', 'transform
s', 'type', 'uid', 'uirevision', 'unselected', 'visible', 'z', 'zauto', 'zmax', 'zmid', 'zmin', 'zsrc', 'key', 'set', 'fram
e', 'transforms', '_isNestedKey', '_isSimpleKey', '_isGraticule', '_bbox'
```



```
# Q2.8. Add animation capability (3 points)
# hint:
#     for variable frame you need either integer or character/factorial so
#     convert date to character or factorial

df_month <- dframe %>%
  group_by(state, year_month = format(date, "%Y-%m")) %>%
  summarize(year_month = min(date),
            state = unique(state),
            fips = unique(fips),
            cum_cases = last(cases),
            cum_deaths = last(deaths)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'state'. You can override using the
## `.groups` argument.
```

```
# Calculate the highest_cases column
df_month$highest_cases <- sapply(seq_len(nrow(df_month)), function(i) {
  if (i == 1) {
    df_month$cum_cases[i]
  } else {
    max(df_month$cum_cases[i] - df_month$cum_cases[i-1])
  }
})

# Add date column
df_month$date <- format(as.Date(paste0(df_month$year_month, "-01")), "%Y-%m")

# Add state_abb column
df_month$state_abb <- state.abb[match(df_month$state, state.name)]


# Select and reorder columns
df_resultant <- df_month %>%
  select(state, year_month, fips, cum_cases, cum_deaths, highest_cases, date, state_abb)

# Print the resulting dataframe

df_resultant
```

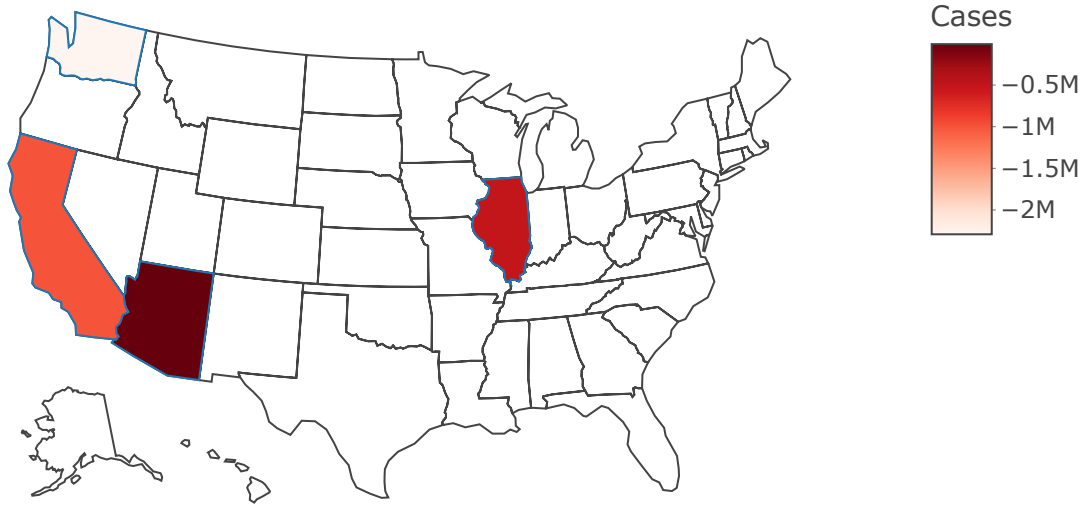
state <chr>	year_month <IDate>	fips <int>	cum_cases <int>	cum_deaths <int>	highest_cases <int>	date <chr>	state_abb <chr>
Alabama	2020-03-13	1	999	14	999	2020-03	AL
Alabama	2020-04-01	1	7068	272	6069	2020-04	AL
Alabama	2020-05-01	1	17952	630	10884	2020-05	AL
Alabama	2020-06-01	1	38045	950	20093	2020-06	AL
Alabama	2020-07-01	1	87723	1580	49678	2020-07	AL
Alabama	2020-08-01	1	126058	2182	38335	2020-08	AL

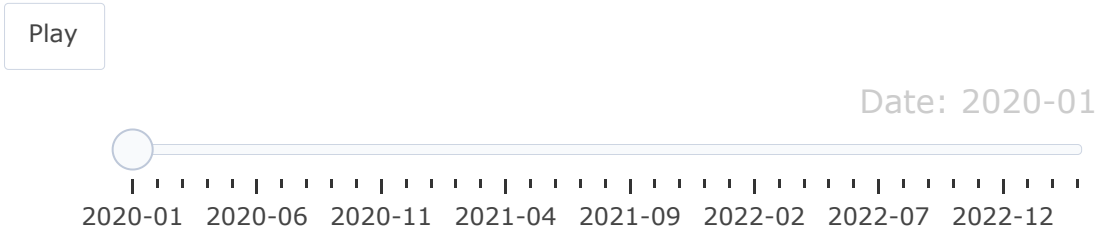
state <chr>	year_month <IDate>	fips <int>	cum_cases <int>	cum_deaths <int>	highest_cases <int>	date <chr>	state_abb <chr>
Alabama	2020-09-01	1	154701	2540	28643	2020-09	AL
Alabama	2020-10-01	1	192285	2967	37584	2020-10	AL
Alabama	2020-11-01	1	249524	3578	57239	2020-11	AL
Alabama	2020-12-01	1	361226	4827	111702	2020-12	AL

1-10 of 2,068 rows Previous **1** 2 3 4 5 6 … 207 Next

```
# Modify df_resultant dataframe to include a frame column
df_final <- df_resultant %>%
  select(state, year_month, fips, cum_cases, cum_deaths, highest_cases, date, state_abb) %>%
  mutate(frame = date)

# Create the plot
plot_ly(df_final, locations = ~state_abb,
        z = ~highest_cases, type = "choropleth",
        locationmode = "USA-states",
        colorbar = list(title = "Cases"),
        colors = "Reds",
        scope = "usa",
        frame = ~date) %>%
  layout(geo = list(scope = "usa"),
         xaxis = list(title = "Date"),
         yaxis = list(title = "Cumulative Cases")) %>%
  animation_slider(
    currentvalue = list(prefix = "Date: "),
    transition = list(duration = 500),
    x = 0, y = -0.2,
    len = 1
  )
```

```
## Warning: 'choropleth' objects don't have these attributes: 'scope'
## Valid attributes include:
## 'autocolorscale', 'coloraxis', 'colorbar', 'colorscale', 'customdata', 'customdatasrc', 'featureidkey', 'geo', 'geojson',
'hoverinfo', 'hoverinfosrc', 'hoverlabel', 'hovertemplate', 'hovertemplatesrc', 'hovertext', 'hovertextsrc', 'ids', 'idssr
c', 'legendgroup', 'legendgrouptitle', 'legendrank', 'locationmode', 'locations', 'locationssrc', 'marker', 'meta', 'metasr
c', 'name', 'reversescale', 'selected', 'selectedpoints', 'showlegend', 'showscale', 'stream', 'text', 'textsrc', 'transform
s', 'type', 'uid', 'uirevision', 'unselected', 'visible', 'z', 'zauto', 'zmax', 'zmid', 'zmin', 'zsrc', 'key', 'set', 'fram
e', 'transforms', '_isNestedKey', '_isSimpleKey', '_isGraticule', '_bbox'

## Warning: 'choropleth' objects don't have these attributes: 'scope'
## Valid attributes include:
## 'autocolorscale', 'coloraxis', 'colorbar', 'colorscale', 'customdata', 'customdatasrc', 'featureidkey', 'geo', 'geojson',
'hoverinfo', 'hoverinfosrc', 'hoverlabel', 'hovertemplate', 'hovertemplatesrc', 'hovertext', 'hovertextsrc', 'ids', 'idssr
c', 'legendgroup', 'legendgrouptitle', 'legendrank', 'locationmode', 'locations', 'locationssrc', 'marker', 'meta', 'metasr
c', 'name', 'reversescale', 'selected', 'selectedpoints', 'showlegend', 'showscale', 'stream', 'text', 'textsrc', 'transform
s', 'type', 'uid', 'uirevision', 'unselected', 'visible', 'z', 'zauto', 'zmax', 'zmid', 'zmin', 'zsrc', 'key', 'set', 'fram
e', 'transforms', '_isNestedKey', '_isSimpleKey', '_isGraticule', '_bbox'
```





Q2.9. Compare animated plot from Q2.8 to plots from Q2.4/Q2.5 (When you would prefer one or another?) (2 points) –> It can be seen from the all plots that plot in Q2.8 gives far more better visual impact as compared to Q2.4/Q2.5 graphs. Because analyzer can easily view changes in number of cases for every state. and moreover, animated effect make it more obvious to notice little changes from the plots. In earlier graphs, there was a lot of overlapping in data distribution, while this presentation makes it easier to visualize. So, I would prefer map plot of Q2.8 for bulky amount of data.