

Arkaraj Mukherjee: Worksheet 3

!!AI WAS USED TO FIGURE OUT "ARRANGE" AND "FILTER" COMMANDS ALONG WITH MOVING THE LEGEND TO THE BOTTOM AND PLOTTING THE GAUSSIAN ON THE GRAPH BUT ONLY SYNTAX!! Loading necessary libraries and setting the current working directory appropriately:

```
library("tidyverse");

## - Attaching core tidyverse packages ----- tidyverse 2.0.0 -
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.0
## v lubridate  1.9.4     v tidyrr    1.3.2
## v purrr     1.2.0

## - Conflicts -----
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
## to become errors

library("viridis");

## Loading required package: viridisLite

setwd("~/isi_bmath/sem_II/intro_to_statistics_and_computation_with_data");
```

1. (a) The command `readcsv` reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.
(b) `setwd` is used to set the working directory.
(c) `getwd` returns the current working directory.
2. (a) This will read the data from our file and due to the `header = T` argument, assume that the first row (in our case it is the line "Dice") contains the column name and not data itself.

```
DiceR = read.csv("Dice.csv", header = T)
```

This line renames the column in our file to "Sum"

```
names(DiceR) = "Sum"
```

This changes the data type of the column in our file named `Sum` to plain numbers.

```
DiceR$Sum = as.numeric(DiceR$Sum)
```

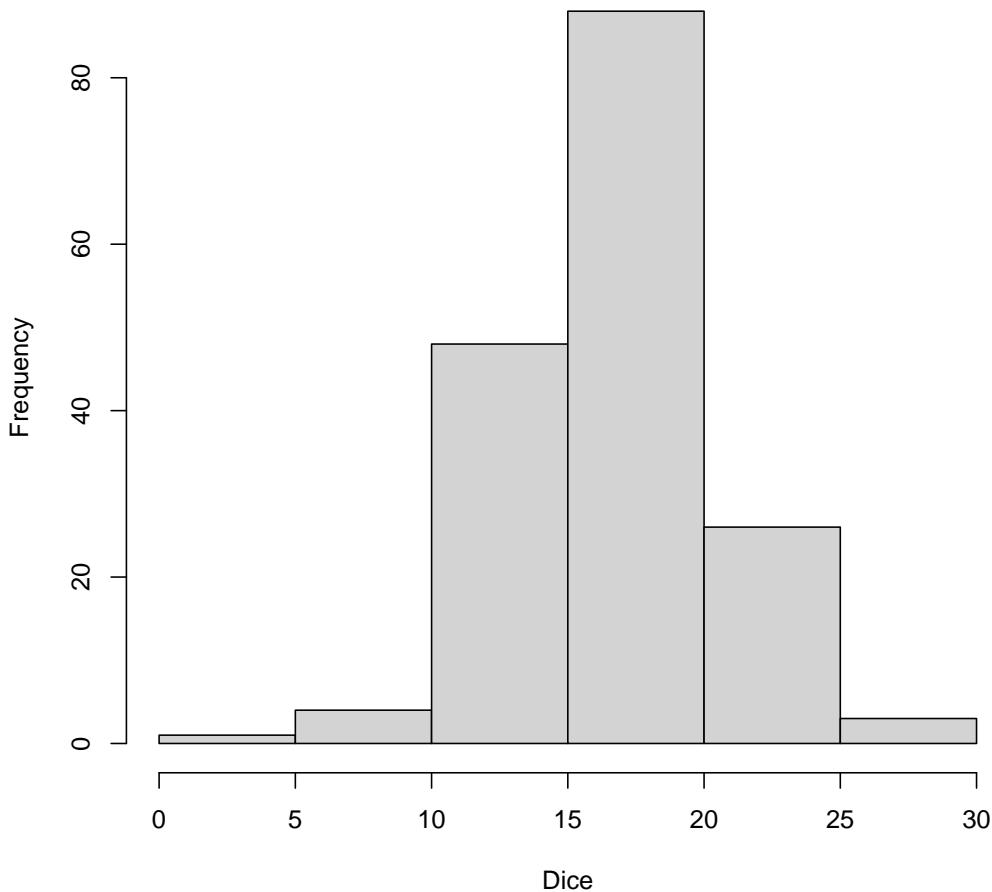
This effectively removes the empty rows which have no data (NA to be more precise) and puts the new clean data into the vector `Dice`

```
Dice = na.omit(DiceR$Sum)
```

This creates a histogram with this data

```
hist(Dice)
```

Histogram of Dice



This calculates and prints some basic statistics from this data like mean, median, mode, maximum, minimum and some quartiles.

```
summary(Dice)
```

```
##   Min. 1st Qu. Median    Mean 3rd Qu.    Max.
##   1.00 15.00 17.00 17.14 20.00 27.00
```

- (b) This calculates the Z-score of the data in our frame and puts it in the vector `cs`

```
cs = (Dice-mean(Dice))/(sd(Dice))
```

This calculates the mean (It will be 0 by linearity of expectation but theres some numerical error at play here) of these Z-scores

```
mean(cs)  
## [1] 1.34184e-16
```

This calculates the standard deviation of these Z-scores

```
sd(cs)  
## [1] 1
```

The first among these lines puts the scores that are within one standard deviation away from the mean in the vector `onesdcs`, the second line does this but for two standard deviations and puts them in `twosdcs` and the third line does it for three standard deviations and puts them in `threesdcs`.

```
onesdcs = cs[cs > -1 & cs < 1];  
twosdcs = cs[cs > -2 & cs < 2];  
threesdcs = cs[cs > -3 & cs < 3];
```

These are indeed approximated well by a normal distribution of appropriate variance and mean. This is because after standardizing this data we find that it is approximated well by the standard normal distribution due to

- The mean of `cs` is 0 and the sd is 1 which are the same as in a standard normal distribution.
- As the sums of these die follow a binomial distribution, upon standardization they become more and more approximable by the normal distribution due to the central limit theorem and the question here is *how good is the approximation*.
- We can use the empirical rule, also called 68-95-99.7 rule(these are the percentage of values that fall in within first, second and third standard deviations of the mean in a standard normal distribution) to say whether this is a good approximation.
- We will now calculate these percentages for our distribution and if they are close to these then our claim holds.

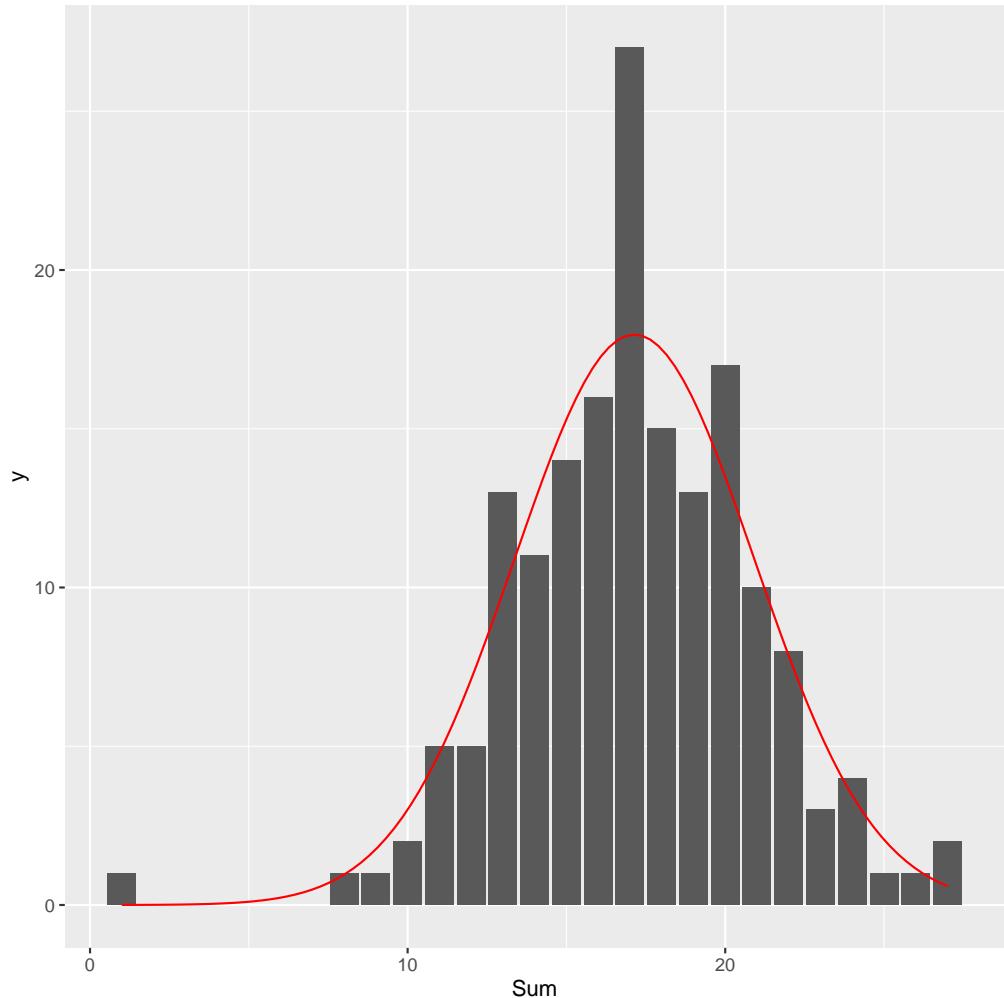
```
100*length(onesdcs)/length(cs)  
## [1] 66.47059  
100*length(twosdcs)/length(cs)  
## [1] 95.88235  
100*length(threesdcs)/length(cs)  
## [1] 99.41176
```

- We can use a bar plot to see the geomtric shape of the data too. The appropriate normal pdf is also graphed alongside, we see that it captures the shape of our data well at most points. `length(Dice) * 1` is multiplied to scale the graph, this is needed as it represents the probability and in our case the number of trials is `length(Dice)`.

```

ggplot(data = DiceR) +
  geom_bar(mapping = aes(x = Sum)) +
  stat_function(
    fun = function(x) dnorm(
      x,
      mean = mean(Dice),
      sd = sd(Dice)) * length(Dice) * 1,
    color = "red"
  )

```



3. (a) This dataset contains daily time-series records of COVID-19 statistics in India from January 30, 2020, onwards. It tracks cumulative counts of confirmed cases, recoveries, deceased individuals, "other" cases, and tested samples, categorized by specific states (like Kerala, Delhi, etc.) and the nation as a whole.

```

statedf = read.csv("states.csv", header = T)
head(statedf)

##           Date State Confirmed Recovered Deceased Other Tested

```

```

## 1 2020-01-30 India      1      0      0      0      0
## 2 2020-01-30 Kerala    1      0      0      0      0
## 3 2020-02-02 India      2      0      0      0      0
## 4 2020-02-02 Kerala    2      0      0      0      0
## 5 2020-02-03 India      3      0      0      0      0
## 6 2020-02-03 Kerala    3      0      0      0      0

```

(b) I pick **Uttar Pradesh** as my name is probably **Arkaraj Mukherjee**

(c) `mystatedf = subset(statedf, statedf$State == "Uttar Pradesh")
head(mystatedf)`

	Date	State	Confirmed	Recovered	Deceased	Other	Tested
## 24	2020-03-04	Uttar Pradesh	7	0	0	0	0
## 31	2020-03-05	Uttar Pradesh	8	0	0	0	0
## 38	2020-03-06	Uttar Pradesh	8	0	0	0	0
## 47	2020-03-07	Uttar Pradesh	8	0	0	0	0
## 56	2020-03-08	Uttar Pradesh	8	0	0	0	0
## 69	2020-03-09	Uttar Pradesh	10	0	0	0	0

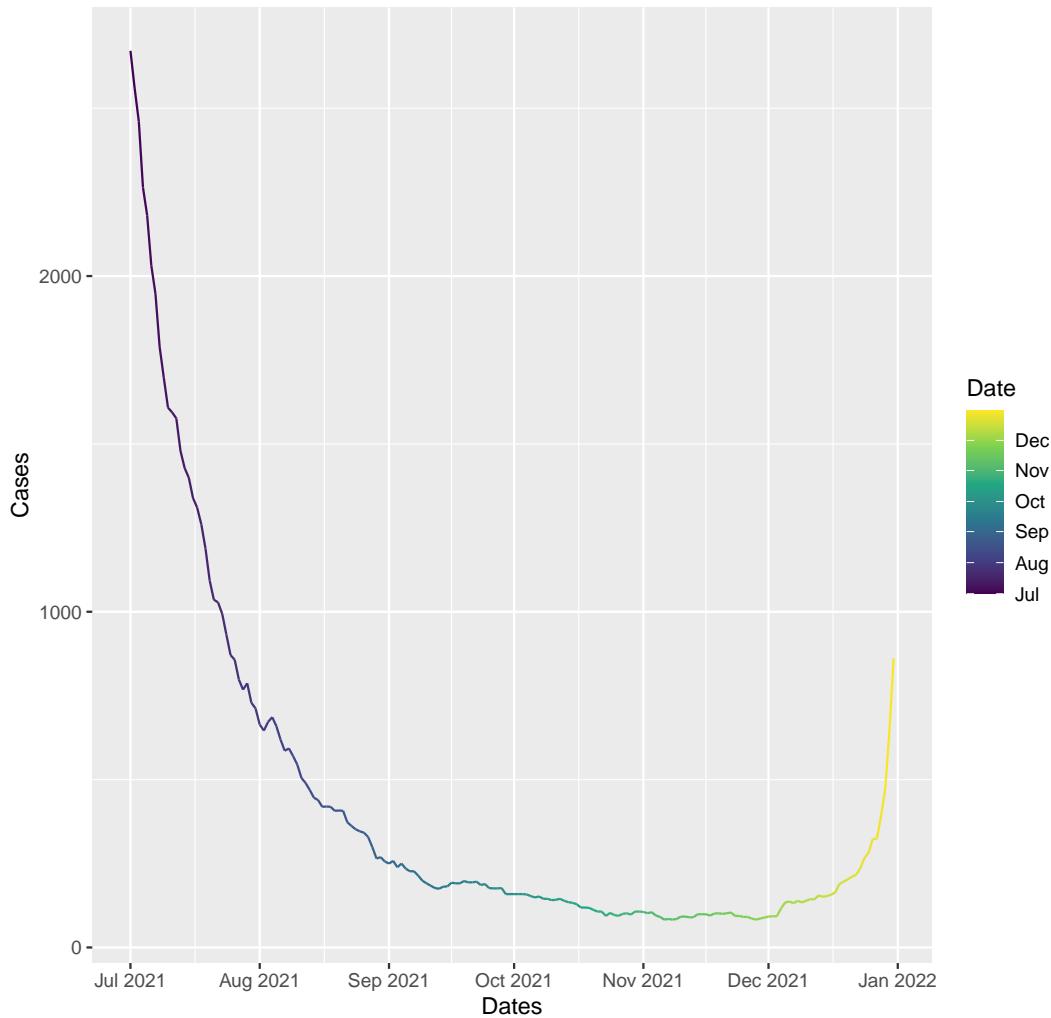
(d) We just have to subtract the number of confirmed cases from all the other closed cases. We put these new numbers in a seperate column called **Active**.

```

mystatedf$Active = mystatedf$Confirmed -
(mystatedf$Recovered + mystatedf$Deceased + mystatedf$Other)
mystatedf$date = as.Date(mystatedf$date)
to_plot = mystatedf %>%
filter(Date >= "2021-07-01" & Date <= "2021-12-31")
ggplot(data = to_plot) +
geom_line(mapping = aes(x = Date, y = Active, color = Date)) +
scale_color_viridis_c(option = "viridis", trans = "date") +
scale_x_date(date_labels = "%b %Y", date_breaks = "1 month") +
labs(title = "Active cases for Uttar Pradesh", x = "Dates", y = "Cases")

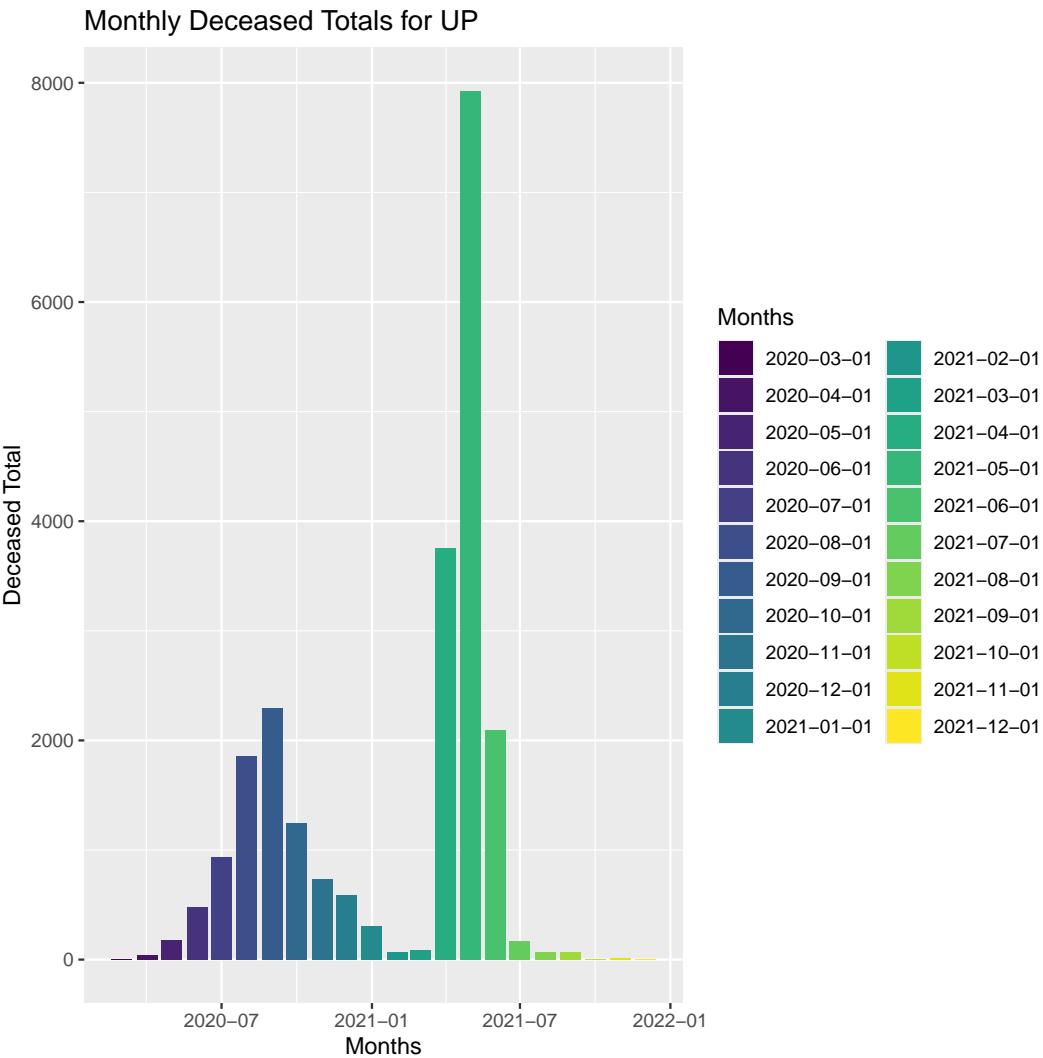
```

Active cases for Uttar Pradesh



- (e) First we get the daily new deceased cases to use them to compute new deaths per month.

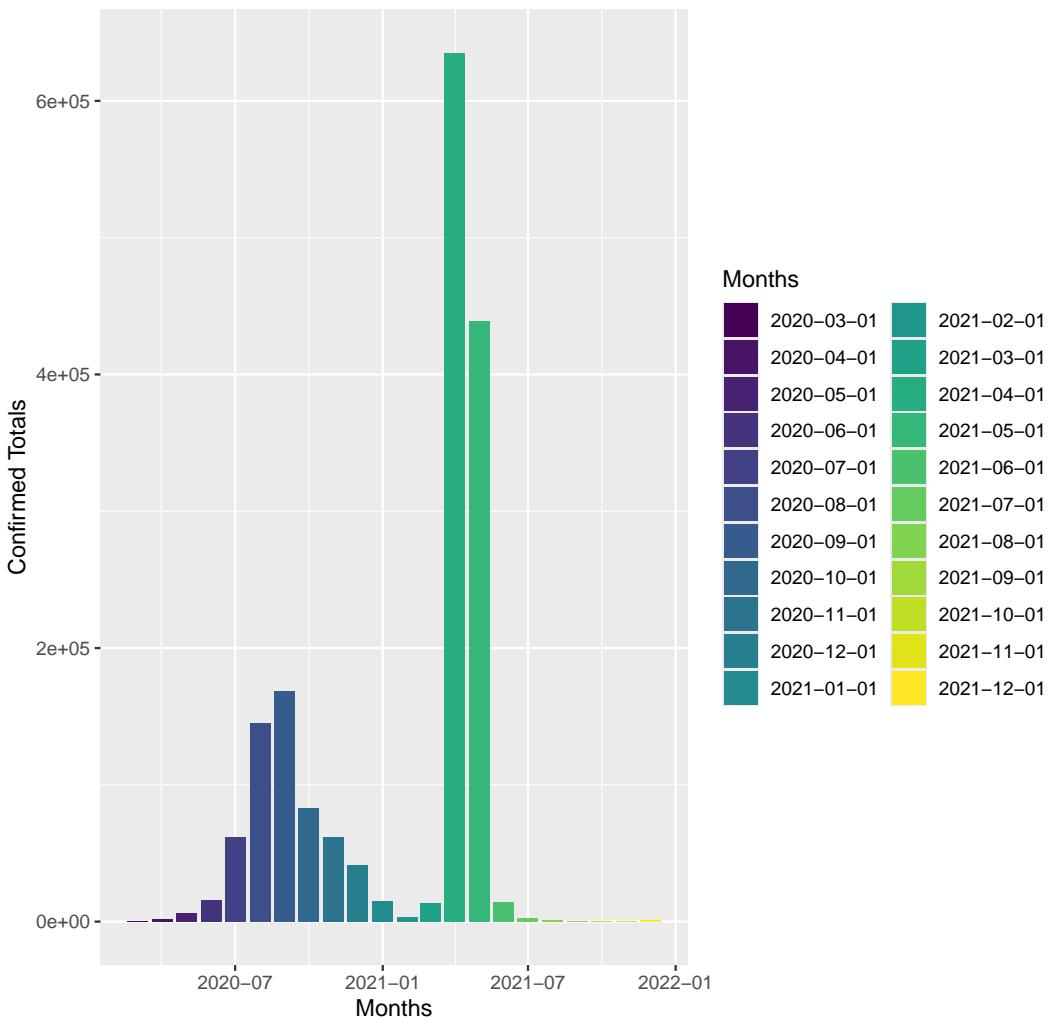
```
mystatedf$Daily_Deceased = c(0, diff(mystatedf$Deceased))
mystatedf$Month = as.Date(format(mystatedf>Date, "%Y-%m-01"))
to_plot_1 =
mystatedf[mystatedf>Date >= "2020-03-01" & mystatedf>Date <= "2021-12-31", ]
monthly_data = aggregate(Daily_Deceased ~ Month, data = to_plot_1, sum)
ggplot(data = monthly_data) +
  geom_bar(mapping = aes(x = Month, y = Daily_Deceased,
  fill = as.factor(Month)), stat = "identity") +
  scale_fill_viridis_d() +
  labs(title = "Monthly Deceased Totals for UP",
  x = "Months",
  y = "Deceased Total",
  fill = "Months")
```



(f) The process to do this is identical to that of the previous part.

```
mystatedf$Daily_Confirmed = c(0, diff(mystatedf$Confirmed))
mystatedf$Month = as.Date(format(mystatedf>Date, "%Y-%m-01"))
to_plot_1 =
mystatedf[mystatedf>Date >= "2020-03-01" & mystatedf>Date <= "2021-12-31", ]
monthly_data = aggregate(Daily_Confirmed ~ Month, data = to_plot_1, sum)
ggplot(data = monthly_data) +
  geom_bar(mapping = aes(x = Month, y = Daily_Confirmed,
  fill = as.factor(Month)), stat = "identity") +
  scale_fill_viridis_d() +
  labs(title = "Monthly Confirmed Totals for UP",
  x = "Months",
  y = "Confirmed Totals",
  fill = "Months")
```

Monthly Confirmed Totals for UP



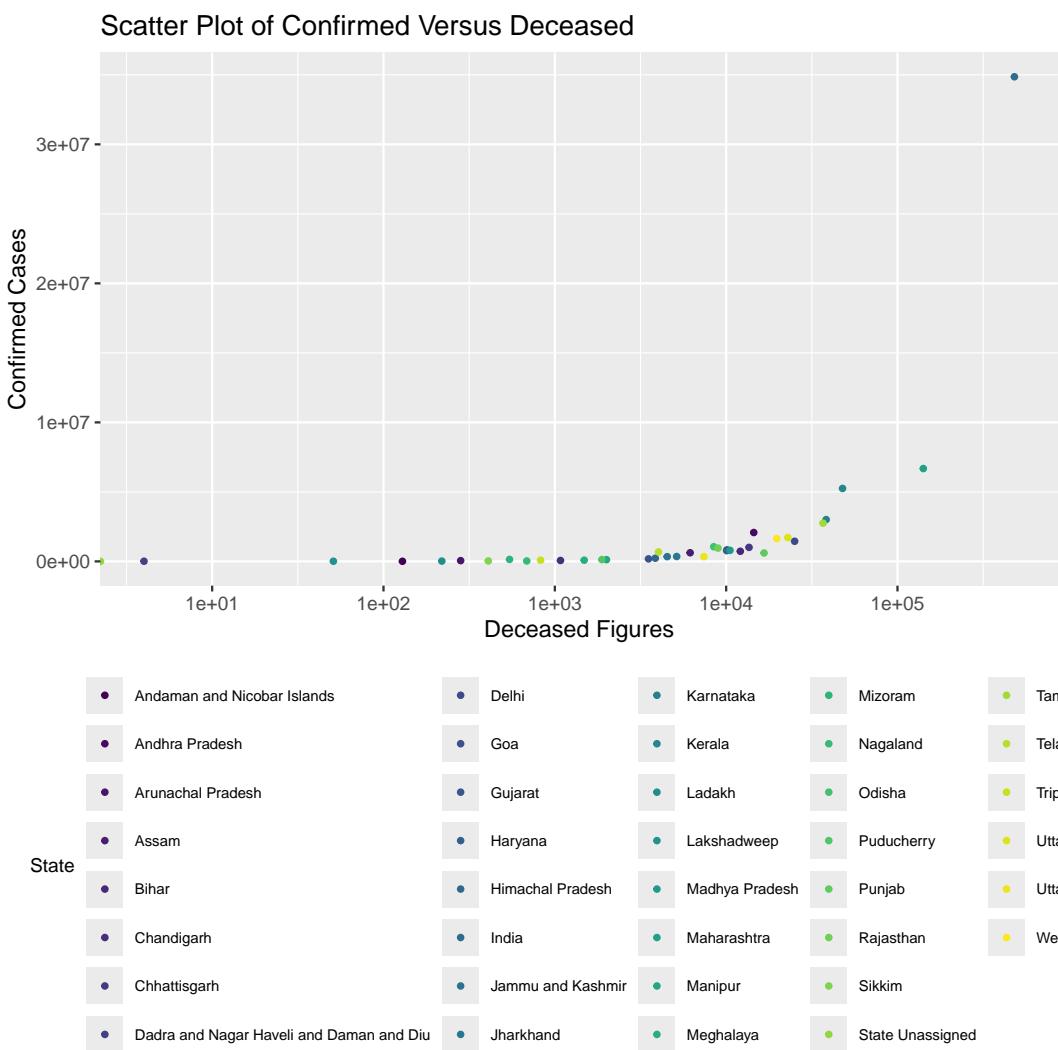
```
(g) statedf = statedf %>%
  arrange(State, Date) %>%
  group_by(State) %>%
  mutate(
    Daily_Confirmed = c(0, diff(Confirmed)),
    Daily_Deceased = c(0, diff(Deceased))
  ) %>%
  ungroup()

state_totals = statedf %>%
  filter(Date >= "2020-03-01" & Date <= "2021-12-31") %>%
  group_by(State) %>%
  summarise(
    Total_Confirmed = sum(Daily_Confirmed, na.rm = TRUE),
    Total_Deceased = sum(Daily_Deceased, na.rm = TRUE)
)
```

```

ggplot(data = state_totals, aes(x = Total_Deceased, y = Total_Confirmed)) +
  scale_x_log10() +
  geom_point(aes(color = State), size = 1) +
  #geom_text(mapping = aes(label = State)) +
  scale_color_viridis_d(option = "viridis") +
  labs(title = "Scatter Plot of Confirmed Versus Deceased",
       x = "Deceased Figures",
       y = "Confirmed Cases") +
  theme(legend.position = "bottom",
        legend.text = element_text(size = 7),
        legend.title = element_text(size = 9))
## Warning in scale_x_log10(): log-10 transformation introduced infinite
values.

```



log scale is used as otherwise the points are concentrated at the bottom right and immense overplotting takes. Names can be added with `geom_text` but the points get covered by the text itself.

```

ggplot(data = state_totals, aes(x = Total_Deceased, y = Total_Confirmed)) +
  scale_x_log10() +
  geom_point(aes(color = State), size = 1) +
  geom_text(mapping = aes(label = State), size = 2) +
  scale_color_viridis_d(option = "viridis") +
  labs(title = "Scatter Plot of Confirmed Versus Deceased",
       x = "Deceased Figures",
       y = "Confirmed Cases") +
  theme(legend.position = "bottom",
        legend.text = element_text(size = 7),
        legend.title = element_text(size = 9))

## Warning in scale_x_log10(): log-10 transformation introduced infinite
values.
## log-10 transformation introduced infinite values.

```

Scatter Plot of Confirmed Versus Deceased

