Community Contribution: Video Tutorial

Exploratory Data Analysis and Visualization

Intuition behind the y-axis of a HexMap

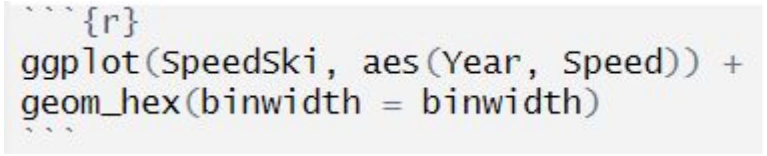
By:-
Anusree Mondal Rakhi (ar4636)
Tushar Prasad (tp2802)

What is Data Visualization?

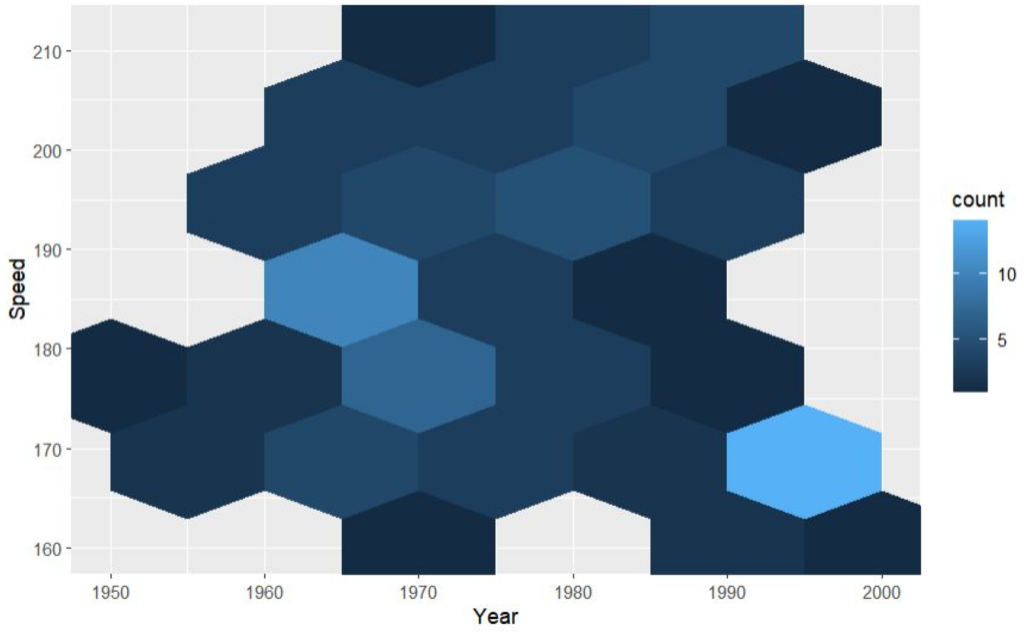
Wikipedia tells us that Data visualization or information visualization (data viz or info viz) is the practice of designing and creating **easy-to-communicate** and **easy-to-understand** graphic or visual representations of a large amount of complex quantitative and qualitative data and information with the help of static, dynamic or interactive visual items.

Easy-to-communicate and easy-to-understand, are some very core concepts without which having a visualization doesn’t really make a lot of sense.

Since this project is more about Hexmaps, let's take examples using those plots. Lets, see if we can iteratively add more information for the user.

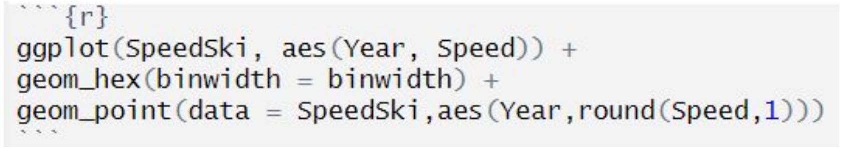


```{r}
ggplot(SpeedSki, aes(Year, Speed)) +
geom_hex(binwidth = binwidth)
```

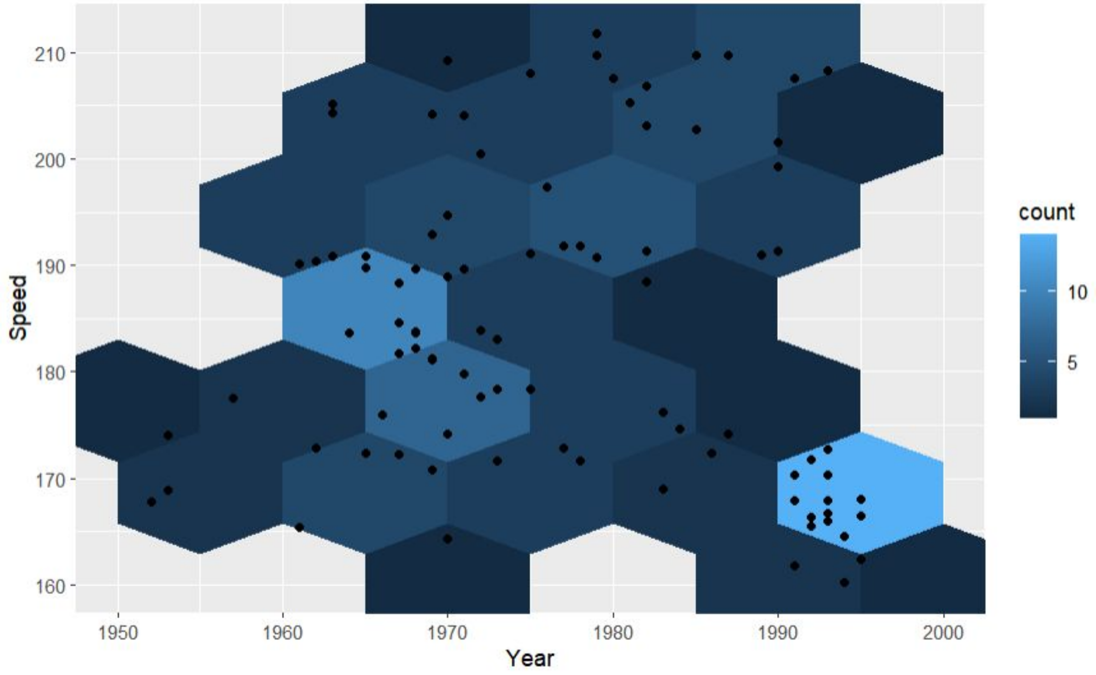


With this graph, we can see the density of skis with different speeds running in different years along with the counts. This is a good starting point to understand this dataset into some detail.

But, immediately after seeing this graph, I am curious to see where do the actual points lie in the Hexmaps. Lets see if we can add more information to it.

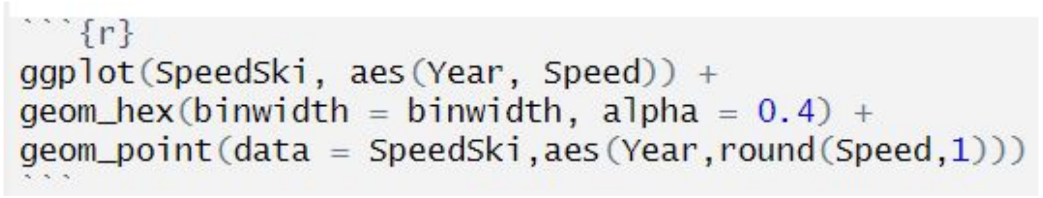


~~~
```{r}
ggplot(SpeedSki, aes(Year, Speed)) +
geom_hex(binwidth = binwidth) +
geom_point(data = SpeedSki,aes(Year,round(Speed,1)))
```
~~~

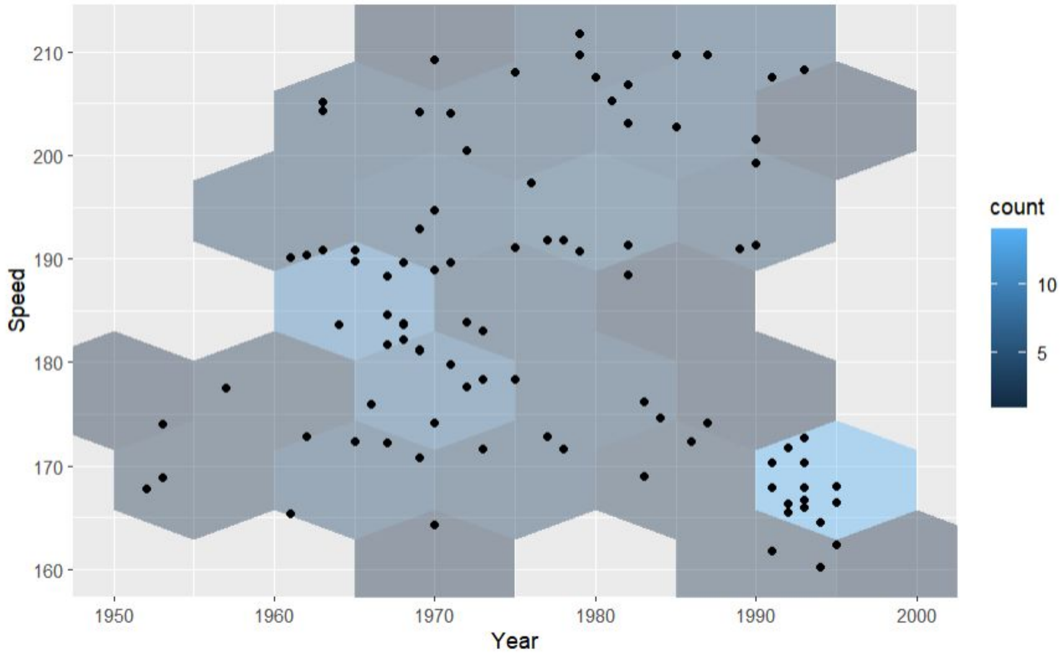


Now, we can exactly see where do our points lie within the HexMaps.

But despite this change, it's hard to exactly spot where the data point is located since the color is overlapping a lot. Let’s see if we can do something about it



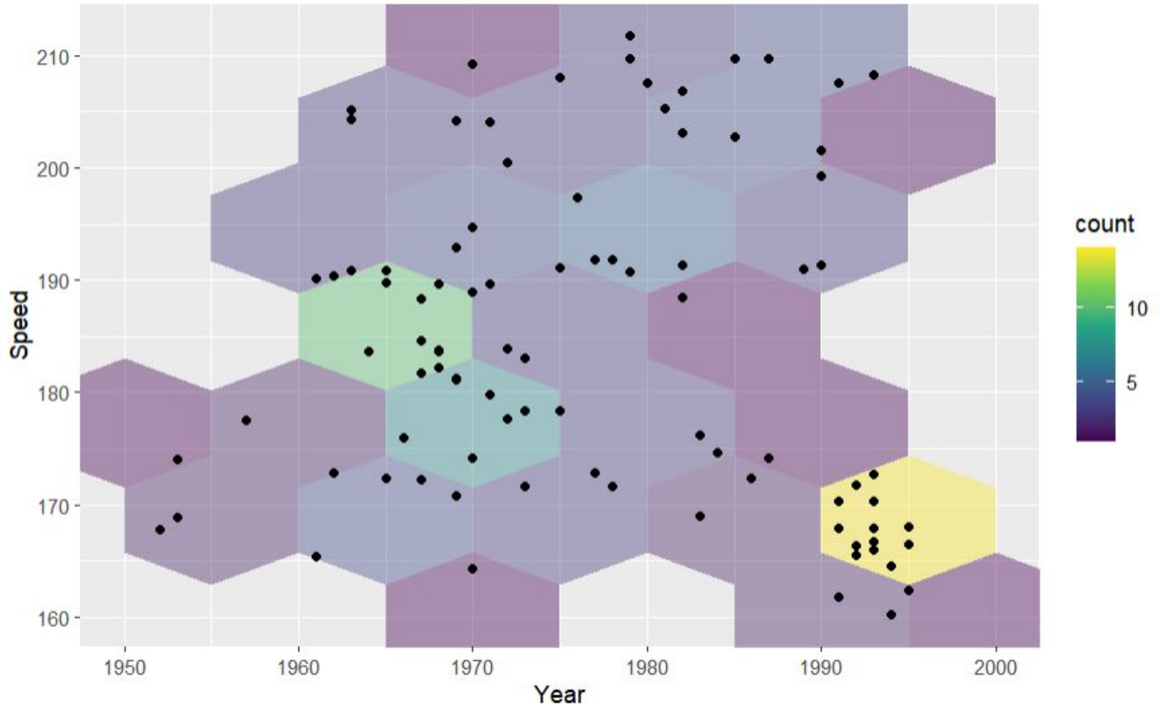
~~~
```{r}
ggplot(SpeedSki, aes(Year, Speed)) +
geom_hex(binwidth = binwidth, alpha = 0.4) +
geom_point(data = SpeedSki,aes(Year,round(Speed,1)))
```
~~~



By adjusting the alpha value of the Hexmap, the data points become much more evident than before.

Despite these changes, we can make the density even more intuitive by changing the color of the gradients to something different.

```{r}
ggplot(SpeedSki, aes(Year, Speed)) +
geom_hex(binwidth = binwidth, alpha = 0.4) +
geom_point(data = SpeedSki,aes(Year,round(Speed,1))) +
scale_fill_viridis_c()
```



By changing the theme of the fill of the HexMaps, it's much easier to find out visually, which hexagon has more data points in it. Now if you look carefully, you can easily figure out the x-axis and the x coordinates of the hexagons, since they lie exactly on the grid lines. But it is not very intuitive to find the y-coordinates of the hexagons, since they don't lie on the grid lines. Unfortunately, there is no inbuilt function to find this out easily.

This is where we come in!

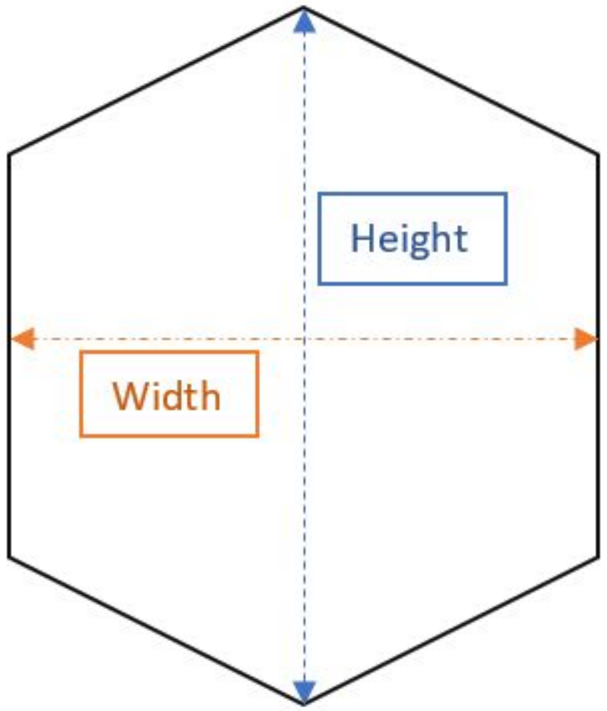
Problem Statement.

The geom_hex() function in ggplot2 is used to create a hexagonal binning plot, which is a way to represent the distribution of data points in a two-dimensional space by grouping them into hexagonal bins.

But inside the geom_hex() function, even after specifying the width and height of the x and y axis, e.g. geom_hex(binwidth = c(10, 10)). It's hard to interpret the height of the Hexagons since both vertices of height don’t lie on the grid line most of the time. Also, the width and height are different in a Hexagon.

In our community contribution assignment, we will talk about how to determine the height of a hexagon.

Here is a visual representation of what I was explaining in the previous slide

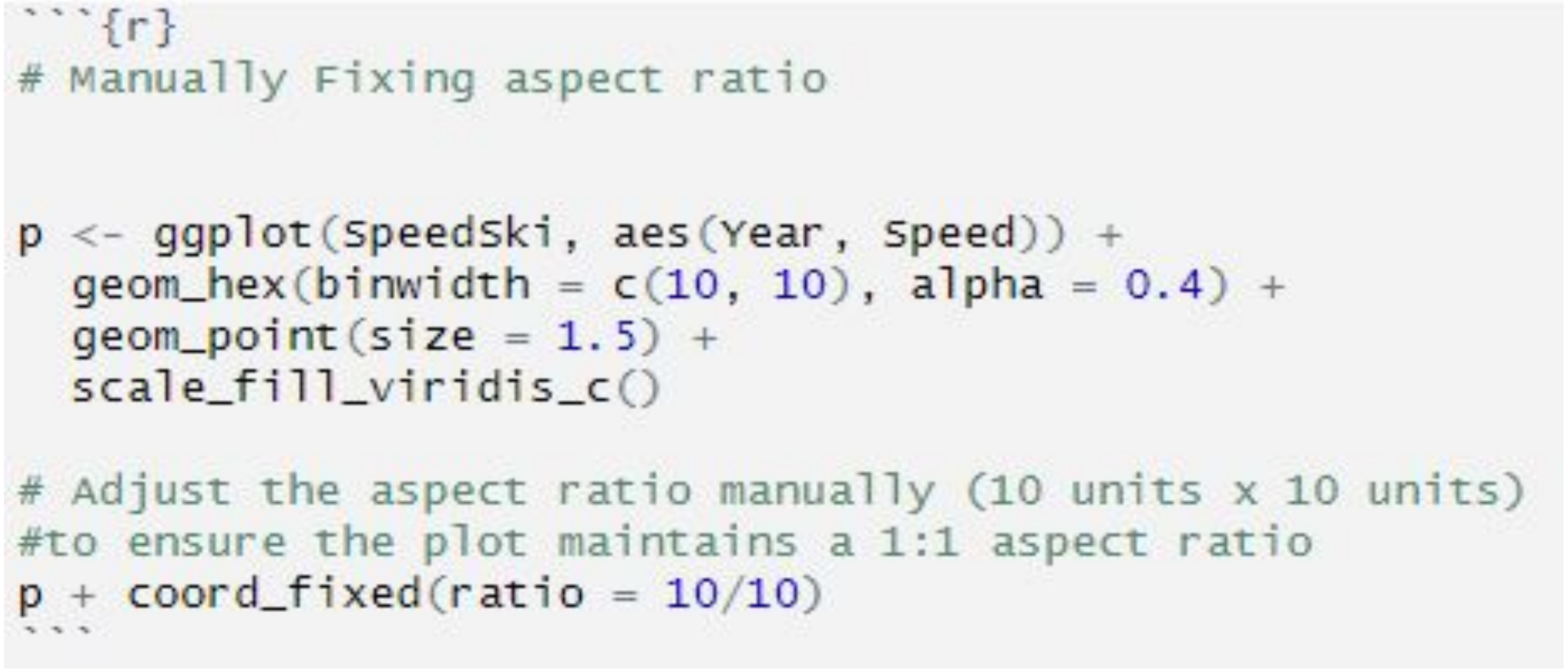


Aspect Ratio

First, we thought this was happening because of the issue with the aspect ratio. Though ggplot2 automatically adjusted the 1:1 aspect ratio.

But to ensure the discrepancy in the graph is not happening due to the aspect ratio, we manually adjusted the aspect ratio to ensure the plot maintains a 1:1 aspect ratio.

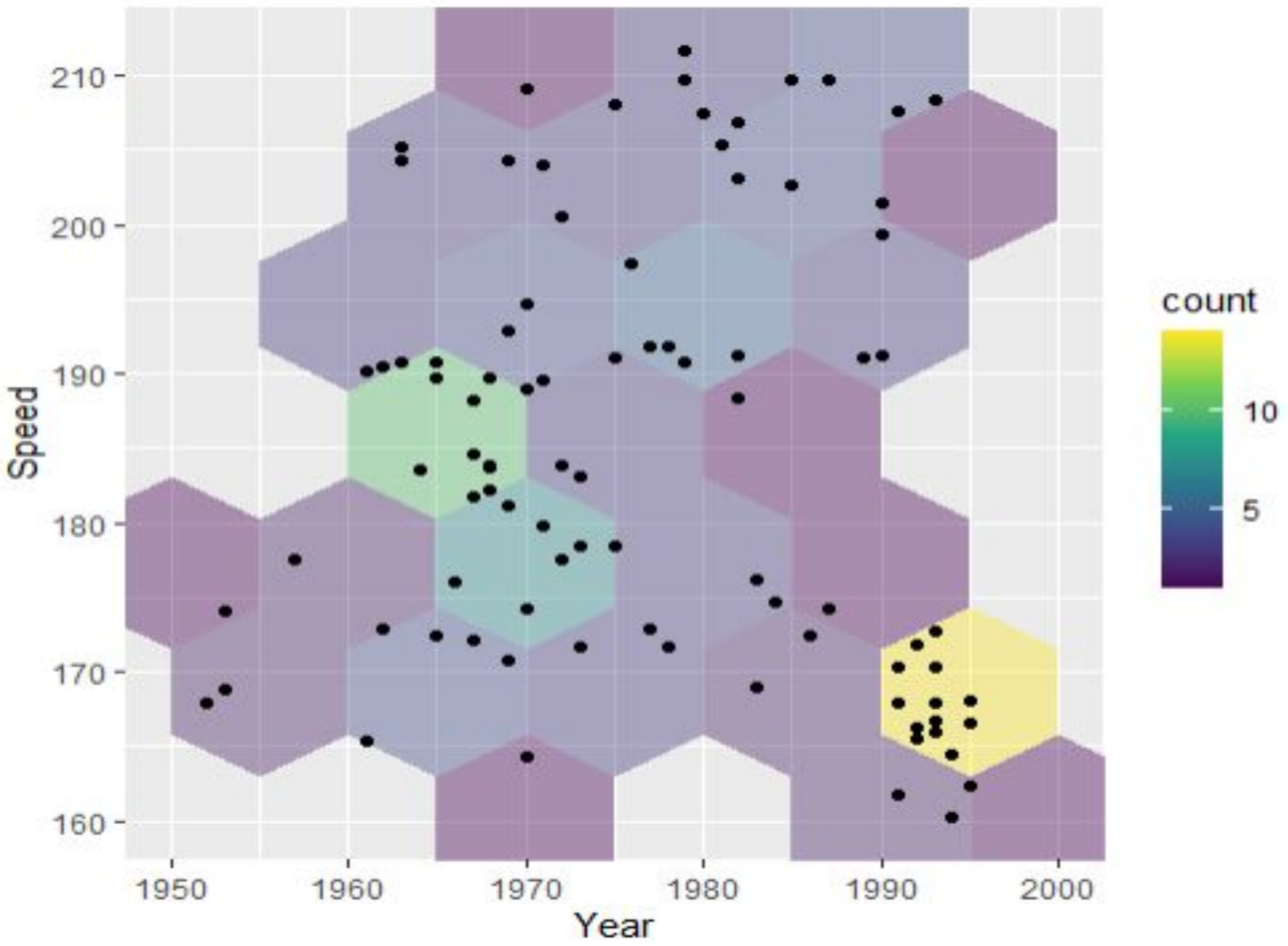
Adjusting Aspect Ratio Manually



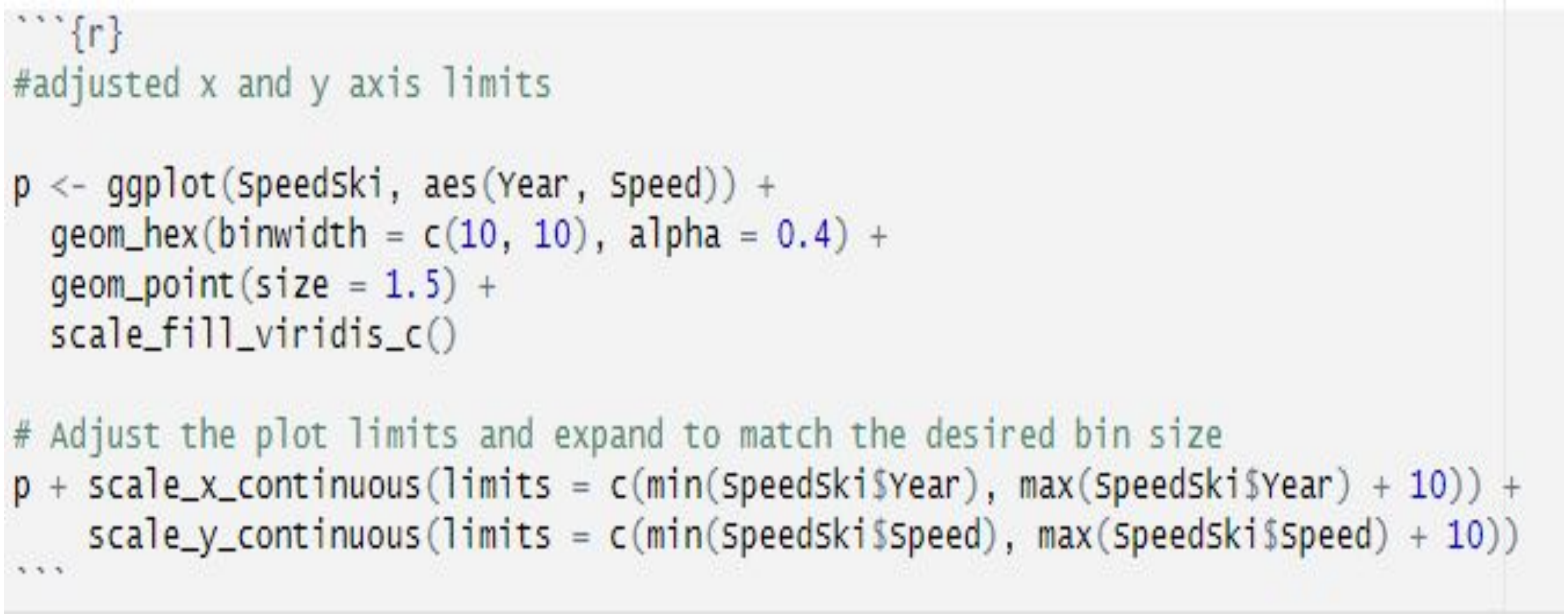
~~~
```{r}
# Manually Fixing aspect ratio


p <- ggplot(SpeedSki, aes(Year, Speed)) +
  geom_hex(binwidth = c(10, 10), alpha = 0.4) +
  geom_point(size = 1.5) +
  scale_fill_viridis_c()

# Adjust the aspect ratio manually (10 units x 10 units)
#to ensure the plot maintains a 1:1 aspect ratio
p + coord_fixed(ratio = 10/10)
```
~~~


count
10
5
210
200
190
180
170
160
Speed
1950
1960
1970
1980
1990
2000
Year

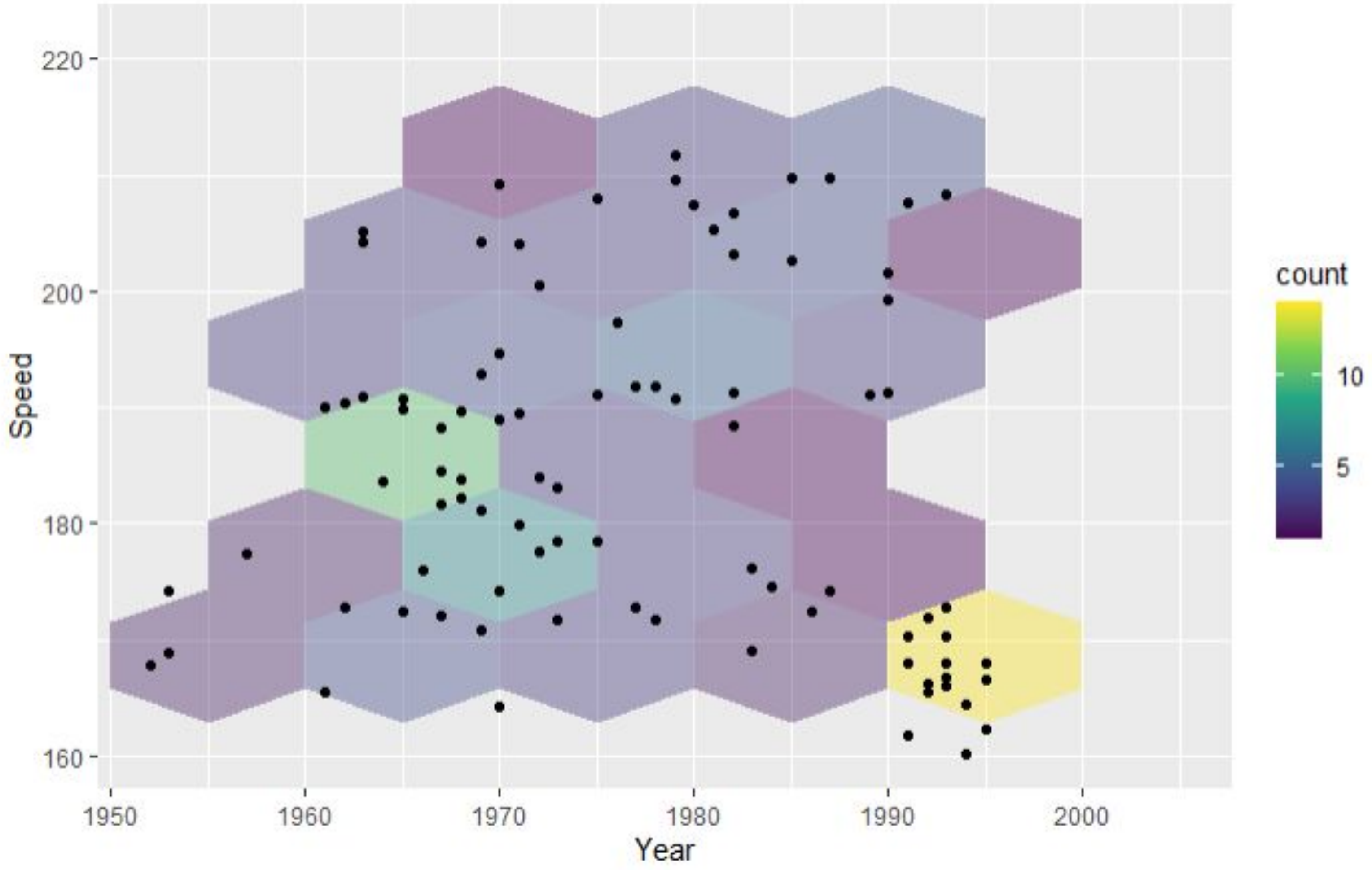
Ensuring that the plot is not distorted!



~~~
```{r}
#adjusted x and y axis limits

p <- ggplot(SpeedSki, aes(Year, Speed)) +
  geom_hex(binwidth = c(10, 10), alpha = 0.4) +
  geom_point(size = 1.5) +
  scale_fill_viridis_c()

# Adjust the plot limits and expand to match the desired bin size
p + scale_x_continuous(limits = c(min(SpeedSki$Year), max(SpeedSki$Year) + 10)) +
    scale_y_continuous(limits = c(min(SpeedSki$Speed), max(SpeedSki$Speed) + 10))
```
~~~


220
200
180
160
Speed
1950
1960
1970
1980
1990
2000
Year
count
10
5

Then, we also experimented with **different values for the binwidth** to manage to fall the hexagon's vertices on the grid line to know the exact height of the hexagon. However, we failed to do so.

So, none of the methods helped us to get the information about exact height of a hexagon from y axis.

So, time to change the strategy.

Solution Time!!!

We can find the height of a regular hexagon by using some simple formulas as shown below.

The width of the hexagon we can be determined from the graph because there is no discrepancy in the input width hexagonal bin size(x-axis) and the unit we can visualize from the graph.

If the width is **10** (In our case).

Width = 10 = **f * a**, where: **f = 1.732** and **a = length of the side**.

From the formula, we can get the a, which is **5.77** units.

We know for a Regular Hexagon, the height = 2 * a. By doing the calculations, **height = 11.6**.

But the previous solution, will only work with the Regular Hexagon.

Non Regular and Regular Hexagon (Generalized Solution)

For a non-regular hexagon, there is no direct formulas like regular hexagon.

So, we came up with the easiest and generalized method, which will work with both regular and non-regular hexagons. After researching the following issue, We learned that we need to find some method to calculate all the vertices of a hexagon.

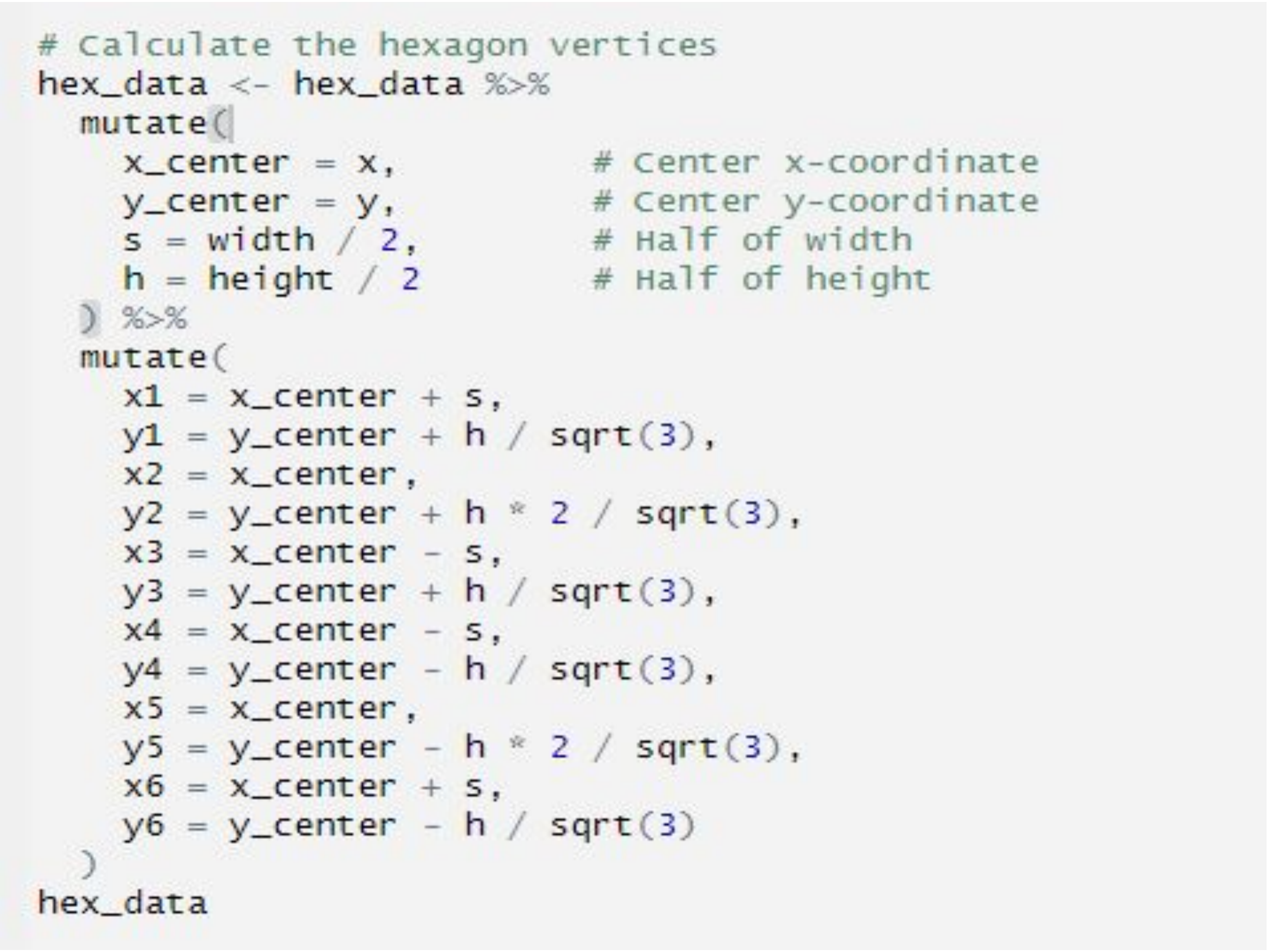
To do so, we used the built-in function, ggplot_build(). This function retrieves information about the hexagonal bins, their positions, and attributes.

```r
# Extract the hexagon data
hex_data <- ggplot_build(p)$data[[1]]
hex_data
```

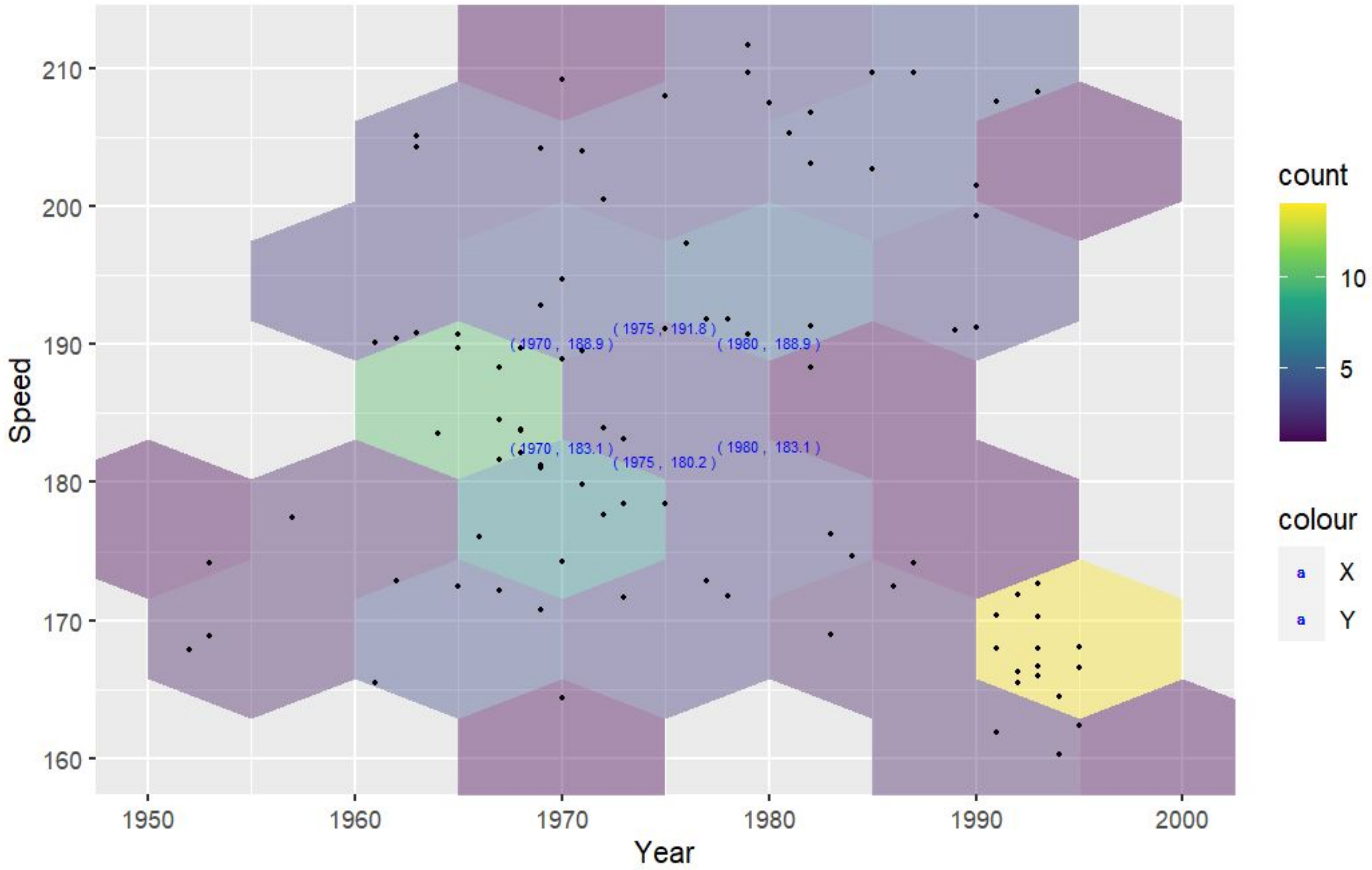
fill <chr>	x <dbl>	y <dbl>	width <dbl>	height <dbl>	density <dbl>	ndensity <dbl>	count <int>	ncount <dbl>	PANEL <fctr>
#440154	1970	160.0000	10	10	0.01098901	0.07142857	1	0.07142857	1
#462167	1990	160.0000	10	10	0.02197802	0.14285714	2	0.14285714	1
#440154	2000	160.0000	10	10	0.01098901	0.07142857	1	0.07142857	1
#462167	1955	168.6603	10	10	0.02197802	0.14285714	2	0.14285714	1
#404C88	1965	168.6603	10	10	0.04395604	0.28571429	4	0.28571429	1
#44377B	1975	168.6603	10	10	0.03296703	0.21428571	3	0.21428571	1
#462167	1985	168.6603	10	10	0.02197802	0.14285714	2	0.14285714	1
#FDE725	1995	168.6603	10	10	0.15384615	1.00000000	14	1.00000000	1
#440154	1950	177.3205	10	10	0.01098901	0.07142857	1	0.07142857	1
#462167	1960	177.3205	10	10	0.02197802	0.14285714	2	0.14285714	1

1-10 of 27 rows | 1-10 of 15 columns

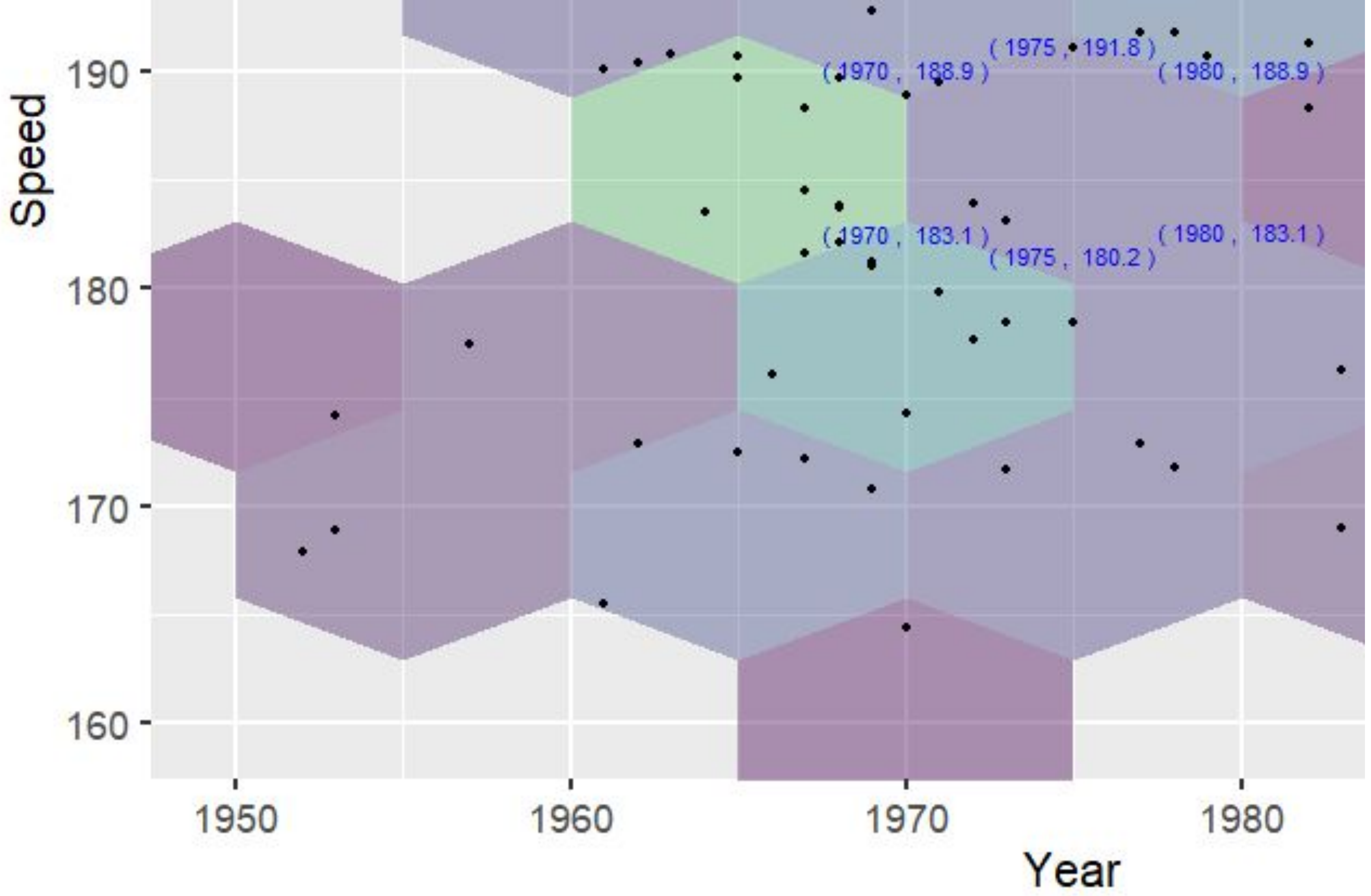
Previous 1 2 3 Next



```r
# Calculate the hexagon vertices
hex_data <- hex_data %>%
  mutate(
    x_center = x,          # Center x-coordinate
    y_center = y,          # Center y-coordinate
    s = width / 2,         # Half of width
    h = height / 2         # Half of height
  ) %>%
  mutate(
    x1 = x_center + s,
    y1 = y_center + h / sqrt(3),
    x2 = x_center,
    y2 = y_center + h * 2 / sqrt(3),
    x3 = x_center - s,
    y3 = y_center + h / sqrt(3),
    x4 = x_center - s,
    y4 = y_center - h / sqrt(3),
    x5 = x_center,
    y5 = y_center - h * 2 / sqrt(3),
    x6 = x_center + s,
    y6 = y_center - h / sqrt(3)
  )
hex_data
```



Speed
Year
210
200
190
180
170
160
1950
1960
1970
1980
1990
2000
(1975 , 191.8)
(1970 , 188.9)
(1980 , 188.9)
(1970 , 183.1)
(1980 , 183.1)
(1975 , 180.2)
count
10
5
colour
a X
a Y


(1975 , 191.8)
(1970 , 188.9)
(1980 , 188.9)
(1970 , 183.1)
(1980 , 183.1)
(1975 , 180.2)
190
180
170
160
Speed
1950
1960
1970
1980
Year

Final Generalized Solution

Like this, we have solved estimating the height of a hexagon in a hexmap. Also, the solution we came up with is extracting all the 12 points of 6 vertices in a hexagon. Now, we can not only get information on the height (corner distance) of a hexagon (regular or non-regular), but also we can get any values related to the particular hexagon.

Now what if we want to add the corner points to all the hexagons and interact with it. Let's try to interact with it in real time.

In conclusion, we are able to find out all the coordinates of the HexMap at the end of the project. We feel this particular contribution can help the community understand the HexMaps in a more intuitive way and answer some curiosities.

What are the coordinates of each hexagon?

What's the representation of each hexagon on the Y-axis?

Why is there a discrepancy between the binwidth of the hexagon and the height of the hexagon?

Hopefully this can help answer these questions and can eventually be a part of the geom_hex or ggplot package to make things more **easy-to-communicate** and **easy-to-understand** since that is what Data Visualization is all about.

References

- https://github.com/tidyverse/ggplot2/blob/f74dbbe53fb0ed288c580f0d762a38a755ec8caf/R/geom-hex.R
- https://github.com/tidyverse/ggplot2/blob/main/R/hexbin.R
- https://github.com/tidyverse/ggplot2/blob/HEAD/R/stat-binhex.R
- https://en.wikipedia.org/wiki/Data_and_information_visualization
- https://edav.info/

THANK YOU