Learning Objectives: Composition Charts

- **Create a pie chart**
- **Create a stacked column chart**
- **Create an area chart**
- **Add labels to pie, stacked column, and area charts**
- **Determine the best composition chart to use based on the data provided**

definition

Assumptions

- Learners are comfortable reading and importing CSV data sets, extracting relevant data into data frames, and printing that data to the console.

Limitations

- This section will cover composition charts in brief details only and will offer practical visualization functions for learners to start creating charts right away.

Pie Charts

Composition Charts vs. Comparison Charts

The first composition chart you will create is a **pie** chart. The **main** difference between a composition chart and a comparison chart is that the data (particularly on the y-axis) within a composition chart is usually divided into several sub-categories. This is why you typically see a **key** or **legend** specifying the labels for those sub-categories. This isn't to say that composition charts cannot be used to *compare* data, they are just better at showing how the data is comprised of several sub-categories.

Creating Pie Charts

Follow the directions below to open up the `pie.r` file in RStudio.

info

Open the `pie.r` file

Within RStudio, open the `pie.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `pie.r`

Data Import

```
Cities <- c("New York", "Los Angeles", "Chicago", "Houston",
        "Phoenix",
            "Philadelphia", "San Antonio", "San Diego",
        "Dallas", "San Jose")
Population <- c(8.60, 4.06, 2.68, 2.40, 2.71, 1.58, 1.57, 1.45,
        1.40, 1.03)
top_10 <- data.frame(Cities, Population)
print(top_10)
```

Source

Result:

```
          Cities Population
1       New York       8.60
2    Los Angeles       4.06
3        Chicago       2.68
4        Houston       2.40
5        Phoenix       2.71
6   Philadelphia       1.58
7    San Antonio       1.57
8      San Diego       1.45
9         Dallas       1.40
10      San Jose       1.03
```

The basic syntax is:

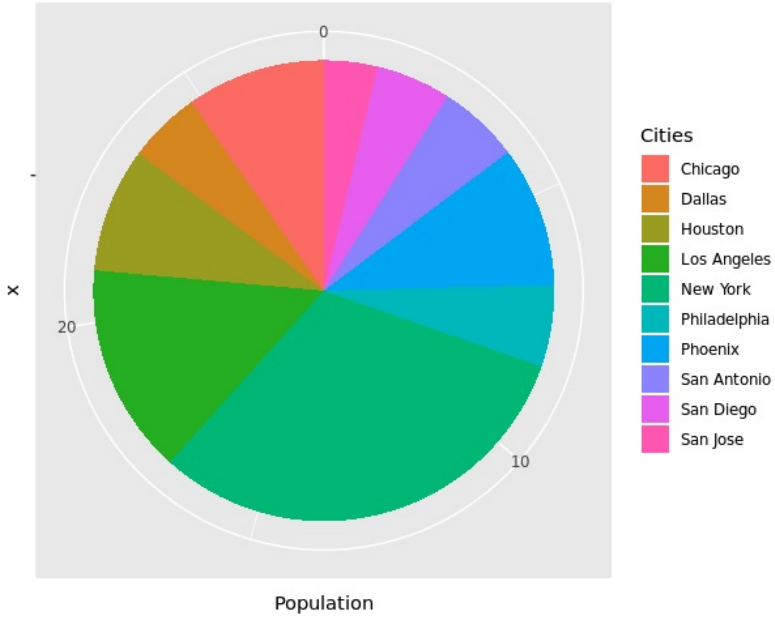
```
ggplot($data, aes(x = "", y = $counts, fill = $categories)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0)
```

Where:
* `$data` represents the data frame
* `$counts` represents the counts
* `$categories` represents the categories

Add on the following code into the text editor and then click the `Source` button to see the result.

```
chart <- ggplot(data = top_10, aes(x = "", y = Population, fill
        = Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0)
print(chart)
```

Result:



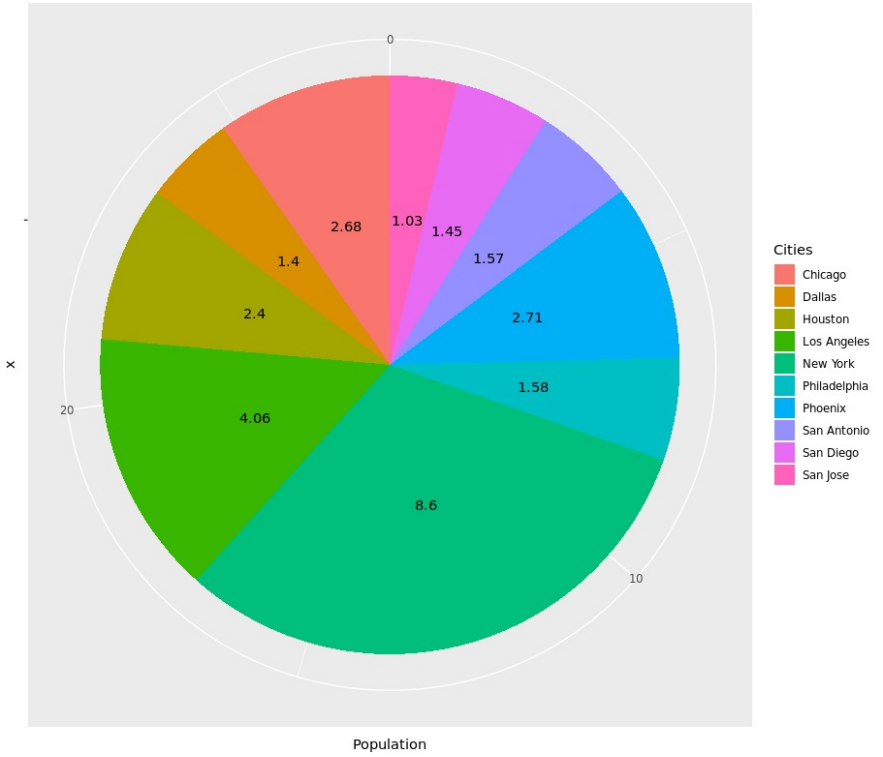
Click this link to enlarge the image.

Notice how the orientation of the chart is similar to that of a column or bar chart. However, we don't fill out the `x` attribute. Instead, we use the `fill` attribute to get the desired pie chart look.

Labels & Aesthetics

To add numerical values to each pie slice, we can use the `geom_text()` function like so:

```
chart <- ggplot(data = top_10, aes(x = "", y = Population, fill
        = Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = Population), position =
        position_stack(vjust = 0.5))
print(chart)
```



Click this link to enlarge the image.

The `label` attribute allows you to choose what to use to label the pie slices. The `position` attribute allows you to adjust the positioning of those labels.

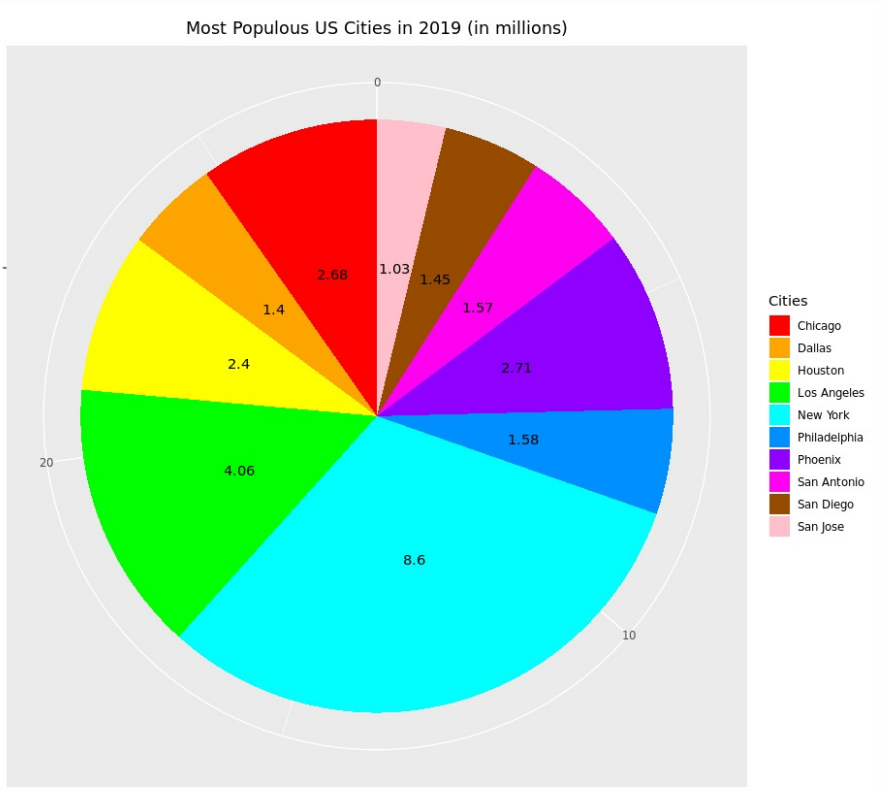
To choose your own colors for the each pie slice, you can use the `scale_fill_manual()` function. And within it, choose the hex code that corresponds with the color you want. For example, the hex code `#FF0000` represents the color red and `#FFA500` represents orange, etc. Use this site here to generate hex codes based on the colors you want.

```
chart <- ggplot(data = top_10, aes(x = "", y = Population, fill
        = Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = Population), position =
        position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                               "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB"))
print(chart)
```

Next we'll use the `labs()` function to add our `title`. However, we'll also use it to remove the `x` and `y` labels by setting them to `NULL` since they are not necessary for pie charts.

```
chart <- ggplot(data = top_10, aes(x = "", y = Population, fill
        = Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = Population), position =
        position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                                "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB")) +
  labs(title = "Most Populous US Cities in 2019 (in millions)",
        x = NULL, y = NULL)
print(chart)
```

Result:

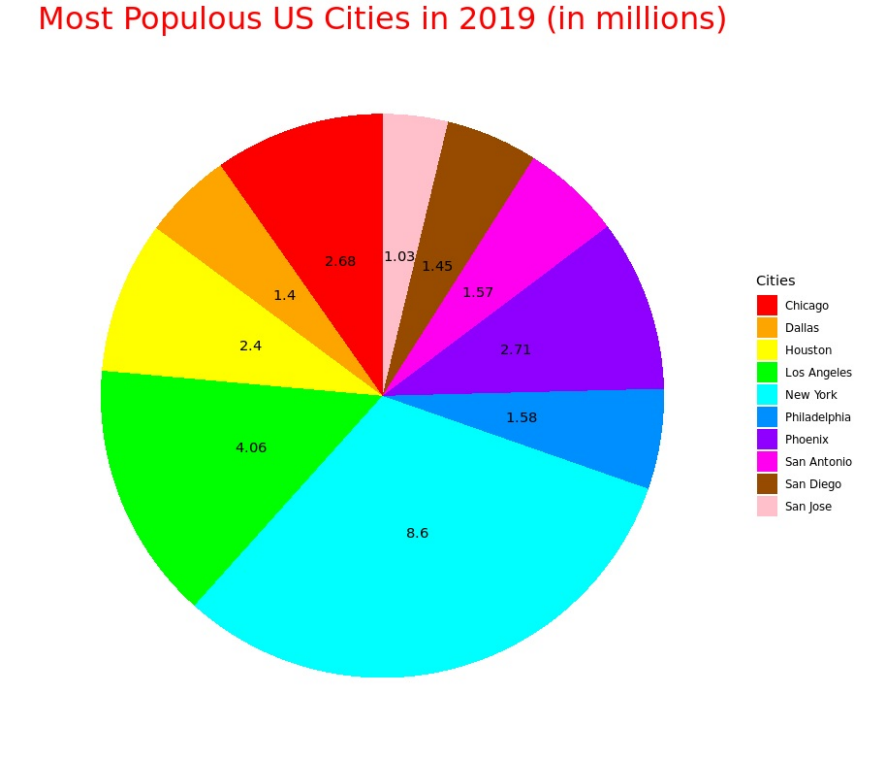


Click this link to enlarge the image.

Next, let's remove the ticks and lines that are present within the pie chart at the moment. In addition, let's clear the background color so that it's white instead of light gray. We can do all of this by using the `theme_classic()` and `theme()` functions.

```r
chart <- ggplot(data = top_10, aes(x = "", y = Population, fill
        = Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = Population), position =
        position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                                "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB")) +
  labs(title = "Most Populous US Cities in 2019 (in millions)",
        x = NULL, y = NULL) +
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(size = 24, hjust = 0.5, color
        = "#FF0000"))
print(chart)
```

Result:



Click this link to enlarge the image.

Notice how we set `axis.line`, `axis.text`, and `axis.ticks` to `element_blank()` to remove the unnecessary marks from the pie chart. Additionally, we used `plot.title = element_text(size = 24, hjust = 0.5, color = "#FF0000")` to set the font size of the title to `size = 24`, centered the title with `hjust = 0.5` and changed the color of the title to red `color = "#FF0000"`.

Percentages

To show percentages instead of counts of the population, modify the data import so that it looks like this:

```
Cities <- c("New York", "Los Angeles", "Chicago", "Houston",
        "Phoenix",
            "Philadelphia", "San Antonio", "San Diego",
        "Dallas", "San Jose")
Population <- c(8.60, 4.06, 2.68, 2.40, 2.71, 1.58, 1.57, 1.45,
        1.40, 1.03)
Percents <- round((Population/sum(Population)) * 100, 2)


top_10 <- data.frame(Cities, Percents)
print(top_10)
```

The `round()` function takes the percentage calculations `(Population/sum(Population)) * 100` and rounds them to two decimal places (`2`).

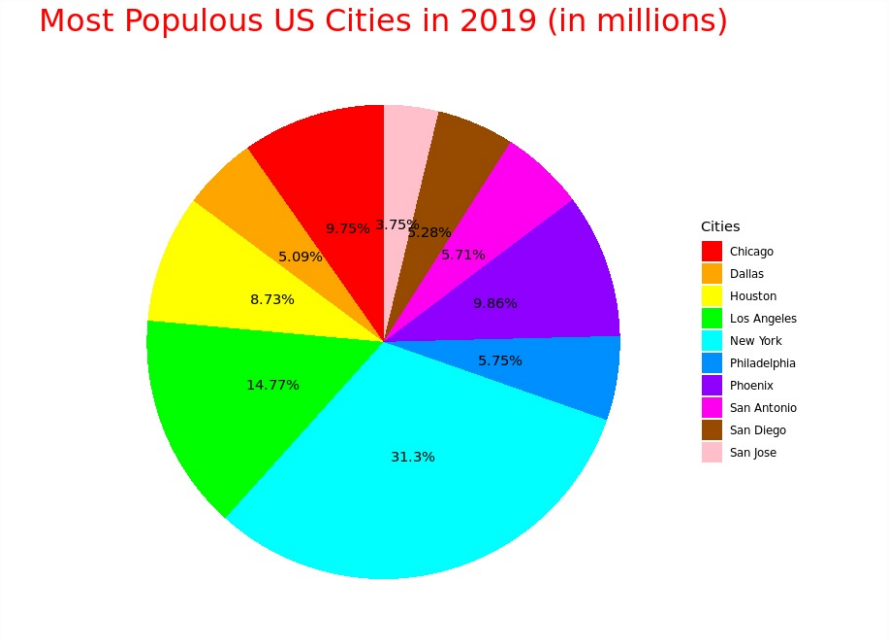
Next, swap `Population` with `Percents` when plotting the chart.

```
chart <- ggplot(data = top_10, aes(x = "", y = Percents, fill =
        Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = Percents), position =
        position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                               "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB")) +
  labs(title = "Most Populous US Cities in 2019 (in millions)",
        x = NULL, y = NULL) +
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(size = 24, hjust = 0.5, color
        = "#FF0000"))
print(chart)
```

Then, change `aes(label = Percents)` to `aes(label = paste0(Percents, "%"))` so that the pie pieces show the `%` symbol.

```
chart <- ggplot(data = top_10, aes(x = "", y = Percents, fill =
        Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = paste0(Percents, "%")), position =
        position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                               "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB")) +
  labs(title = "Most Populous US Cities in 2019 (in millions)",
        x = NULL, y = NULL) +
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(size = 24, hjust = 0.5, color
        = "#FF0000"))
print(chart)
```

Result:

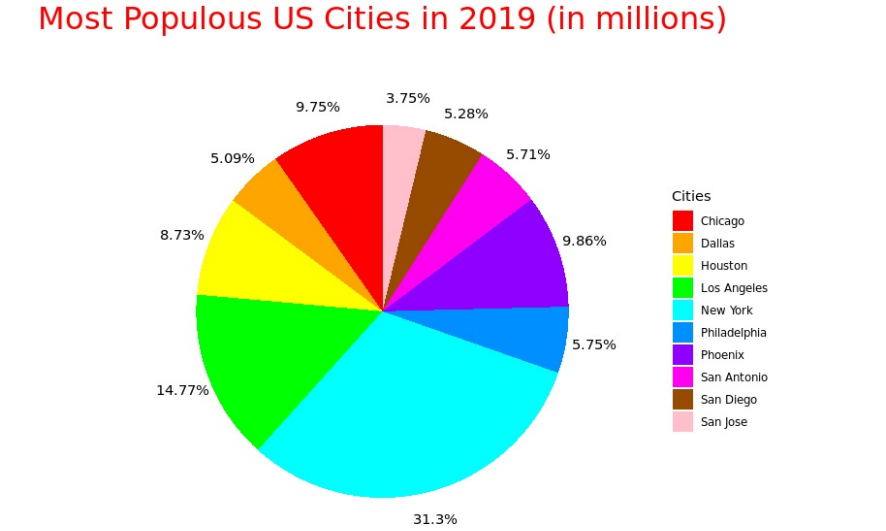


Click this link to enlarge the image.

Unfortunately, the labels are now colliding with each other, to spread the pie percentage labels farther apart, tweak the `aes()` function again from `aes(label = paste0(Percents, "%"))` to `aes(x = 1.65, label = paste0(Percents, "%"))`. The `x` attribute in `aes()` determines how far away from the center of the pie the labels will be. The larger the x value is, the farther away the labels are from the center.

```
chart <- ggplot(data = top_10, aes(x = "", y = Percents, fill =
        Cities)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(x = 1.65, label = paste0(Percents, "%")),
        position = position_stack(vjust = 0.5)) +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                               "#008FFF", "#8F00FF", "#FF00F0",
        "#964B00","#FFC0CB")) +
  labs(title = "Most Populous US Cities in 2019 (in millions)",
        x = NULL, y = NULL) +
  theme_classic() +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        plot.title = element_text(size = 24, hjust = 0.5, color
        = "#FF0000"))
print(chart)
```

Result:



Click this link to enlarge the image.

Stacked Column Charts

Creating Stacked Column Charts

Follow the directions below to open up the `stack.r` file in RStudio.

info

Open the `stack.r` file

Within RStudio, open the `stack.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `stack.r`

Data Import

The data below showcases four different groups of species and their fictional measured condition values for when they are under `normal` or `stress` condition as well as what their `Nitrogen` level is.

```
specie <- c(rep("sorgho", 3) , rep("poacee", 3) , rep("banana",
        3) , rep("triticum", 3))
condition <- rep(c("normal", "stress", "Nitrogen"), 4)
value <- c(3.139308, 1.209113, 5.995753, 24.688453, 22.441730,
        3.236758,
           12.003071, 2.853204, 18.298194, 23.998054, 6.609334,
        6.226210)
df <- data.frame(specie, condition, value)
print(df)
```

Result:

```
     specie condition     value
1    sorgho    normal  3.139308
2    sorgho    stress  1.209113
3    sorgho  Nitrogen  5.995753
4    poacee    normal 24.688453
5    poacee    stress 22.441730
6    poacee  Nitrogen  3.236758
7    banana    normal 12.003071
8    banana    stress  2.853204
9    banana  Nitrogen 18.298194
10 triticum    normal 23.998054
11 triticum    stress  6.609334
12 triticum  Nitrogen  6.226210
```

Source

Note that the function `rep()` stands for **replicate** and that causes the system to create additional copies of a specified vector element. For example:

```
specie <- c(rep("sorgho", 3) , rep("poacee", 3) , rep("banana",
        3) , rep("triticum", 3))
```

means create **three** copies of the elements `"sorgho"`, `"poacee"`, `"banana"`, and `"triticum"` respectively. Printing `specie` will result in:

```
 [1] "sorgho"   "sorgho"   "sorgho"   "poacee"   "poacee"
       "poacee"   "banana"   "banana"   "banana"
[10] "triticum" "triticum" "triticum"
```

Alternatively, you can instruct the system to create a **set** of columns a specified number of times by using the `c()` function **within** the `rep()` function:

```
condition <- rep(c("normal", "stress", "Nitrogen"), 4)
```

Which results in the following when printed:

```
 [1] "normal"   "stress"   "Nitrogen" "normal"   "stress"
       "Nitrogen" "normal"   "stress"   "Nitrogen" "normal"
[11] "stress"   "Nitrogen"
```

Notice how the elements `"normal"`, `"stress"`, and `"Nitrogen"` are duplicated four times as a set.

The basic syntax for a **stacked column** chart is:

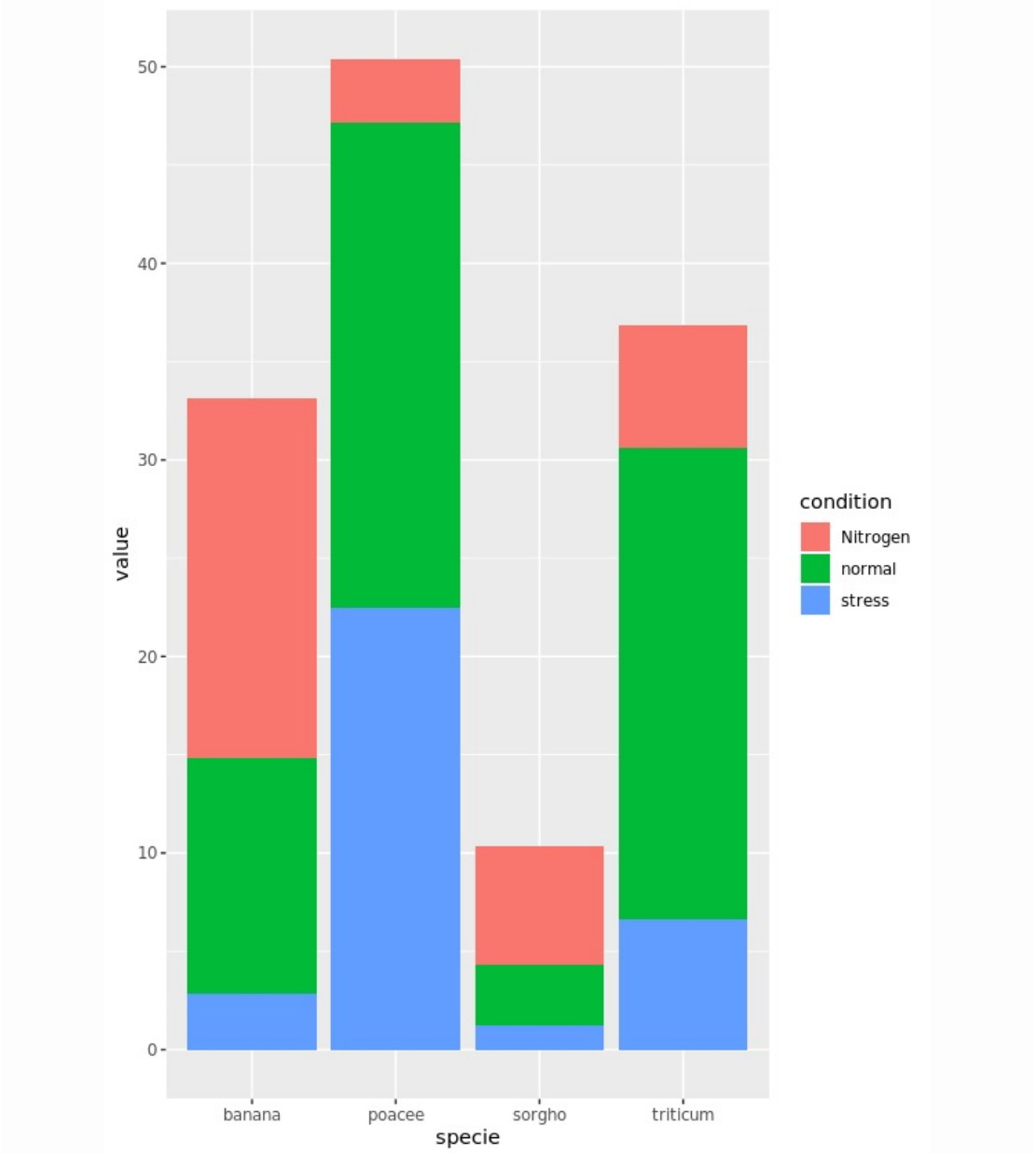
```
ggplot($data, aes(x = $x, y = $y, fill = $fill)) +
  geom_bar(position = "stack", stat = "identity")
```

Where:
* `$x` represents the label on the x-axis
* `$y` represents the label on the y-axis
* `$fill` represents the sub-categories in which each column is broken into

Add on the following code into the text editor and then click the `Source` button to see the result.

```
chart <- ggplot(df, aes(x = specie, y = value, fill =
        condition)) +
  geom_bar(position = "stack", stat = "identity")
print(chart)
```

Result:



Click this link to enlarge the image.

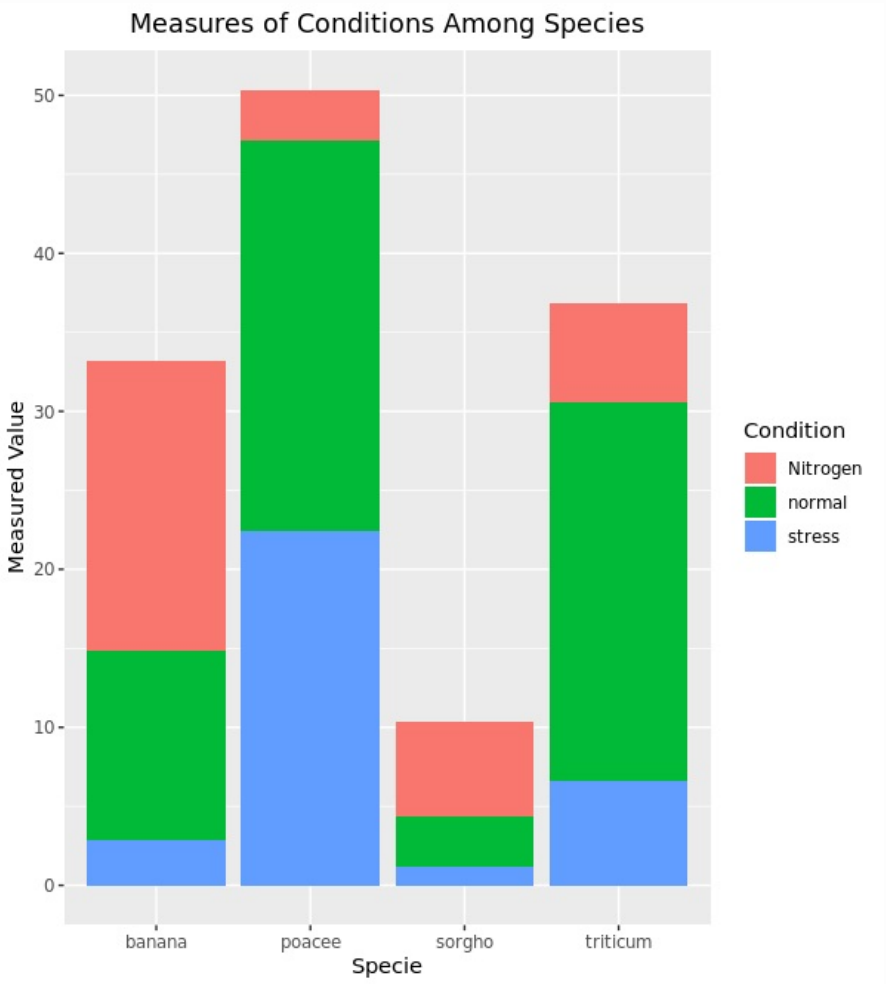
Labels & Aesthetics

As with many charts, we can use `labs()` to label our chart:

```
chart <- ggplot(df, aes(x = specie, y = value, fill =
        condition)) +
  geom_bar(position = "stack", stat = "identity") +
  labs(title = "Measures of Conditions Among Species",
       x = "Specie",
       y = "Measured Value",
       fill = "Condition")
print(chart)
```

Note that `fill` represents the label for the legend or key signifying what each color represents.

Result:

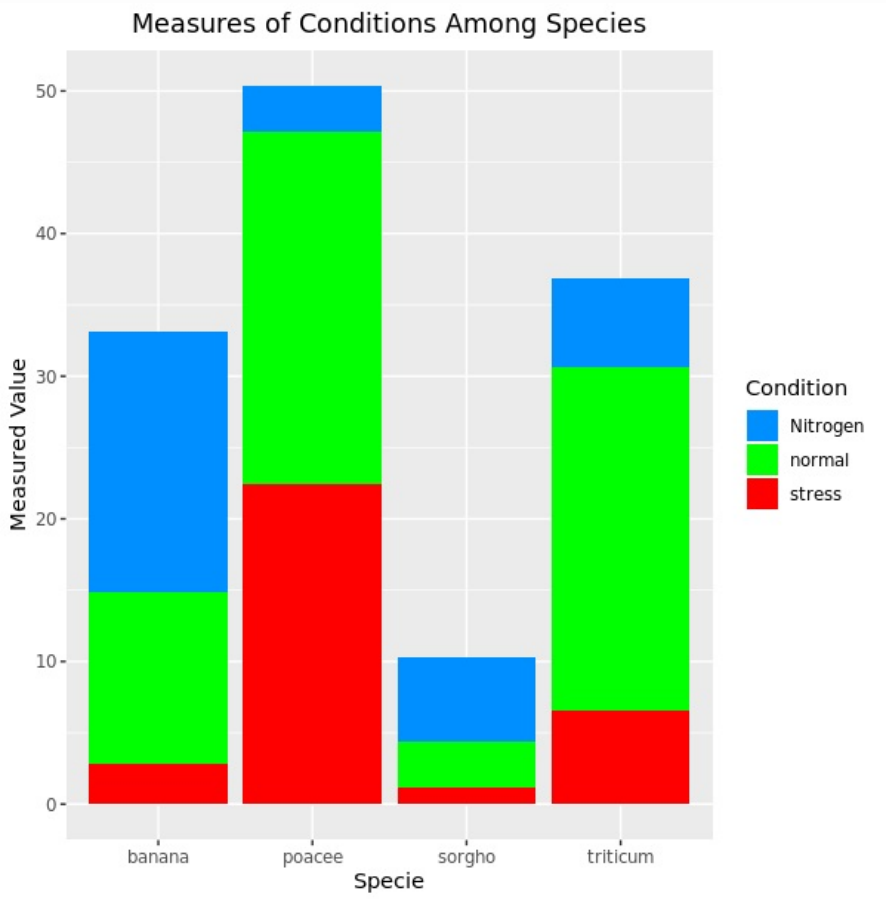


Click this link to enlarge the image.

Use `scale_fill_manual()` to set the color schemes for how each category is filled.

```
chart <- ggplot(df, aes(x = specie, y = value, fill =
        condition)) +
  geom_bar(position = "stack", stat = "identity") +
  labs(title = "Measures of Conditions Among Species",
       x = "Specie",
       y = "Measured Value",
       fill = "Condition") +
  scale_fill_manual(values = c("#008FFF", "#00FF00", "#FF0000"))
print(chart)
```

Result:

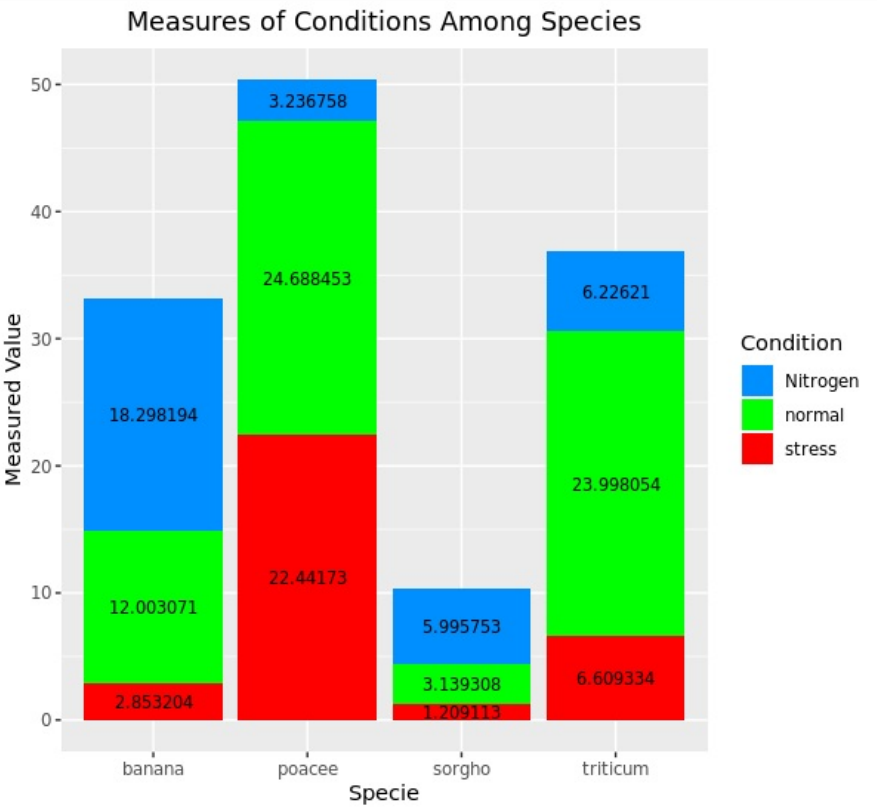


Click this link to enlarge the image.

Add `geom_text()` if you would like to see the value of each piece of the stacks labeled. The `label` attribute represents the values you want the pieces to be labeled as. `size` determines the font size of the labels. And `position` can situate the labels according to your needs.

```
chart <- ggplot(df, aes(x = specie, y = value, fill =
        condition)) +
  geom_bar(position = "stack", stat = "identity") +
  labs(title = "Measures of Conditions Among Species",
       x = "Specie",
       y = "Measured Value",
       fill = "Condition") +
  scale_fill_manual(values = c("#008FFF", "#00FF00", "#FF0000"))
       +
  geom_text(aes(label = value), size = 3, position =
        position_stack(vjust = 0.5))
print(chart)
```

Result:



Click this link to enlarge the image.

Area Charts

Creating Area Charts

Follow the directions below to open up the `area.r` file in RStudio.

info

Open the `area.r` file

Within RStudio, open the `area.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `area.r`

Data Import

```
df <- read.csv("data/uspopage.csv")
```

The data above represents the age distribution of the U.S. population from 1900 to 2002. This data was published in 2003: Source

The basic syntax for an **area** chart is:

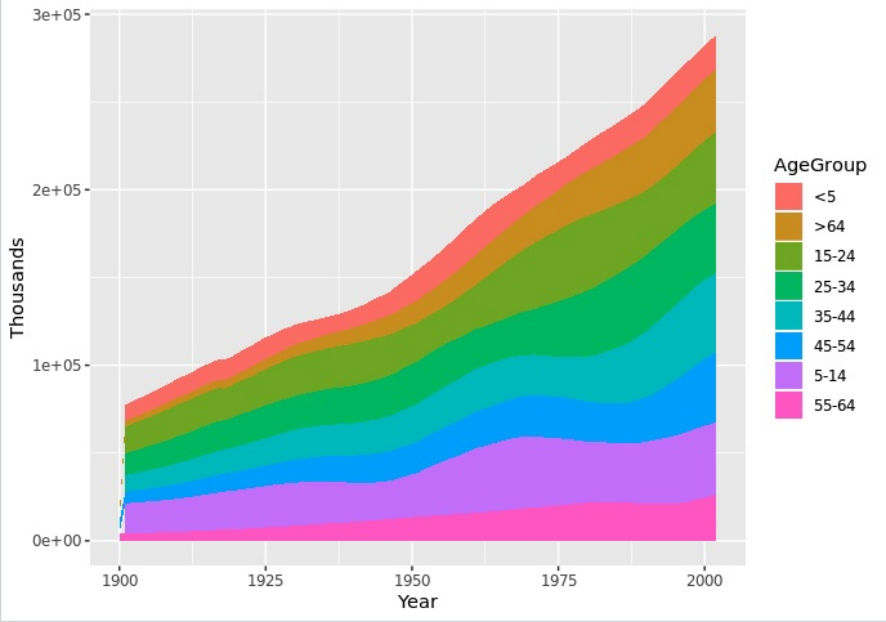
```
ggplot($data, aes(x = $x, y = $y, fill = $fill)) +
  geom_area()
```

Where:
* `$x` represents the label on the x-axis
* `$y` represents the label on the y-axis
* `$fill` represents the sub-categories in which the entire population is broken into

Add on the following code into the text editor and then click the `Source` button to see the result.

```
chart <- ggplot(df, aes(x = Year, y = Thousands, fill =
        AgeGroup)) +
  geom_area()
print(chart)
```

Result:



Click this link to enlarge the image.

You'll notice from the chart that the y-axis `Thousands` has labels that are exponential. For example, `2e+05` is exponential form for `200,000`. However, this can be confusing to understand so we will convert the `Thousands` column from the original data set to `Millions`. This means that instead of having `200,000 Thousand`, which is equivalent to `200 Million`, we'll simply have `200` and our units will be in millions.

Modify your data import so that it looks like this:

```
df <- read.csv("data/uspopage.csv")
Millions <- df$Thousands/1000
df <- cbind(df, Millions)
print(df)
```

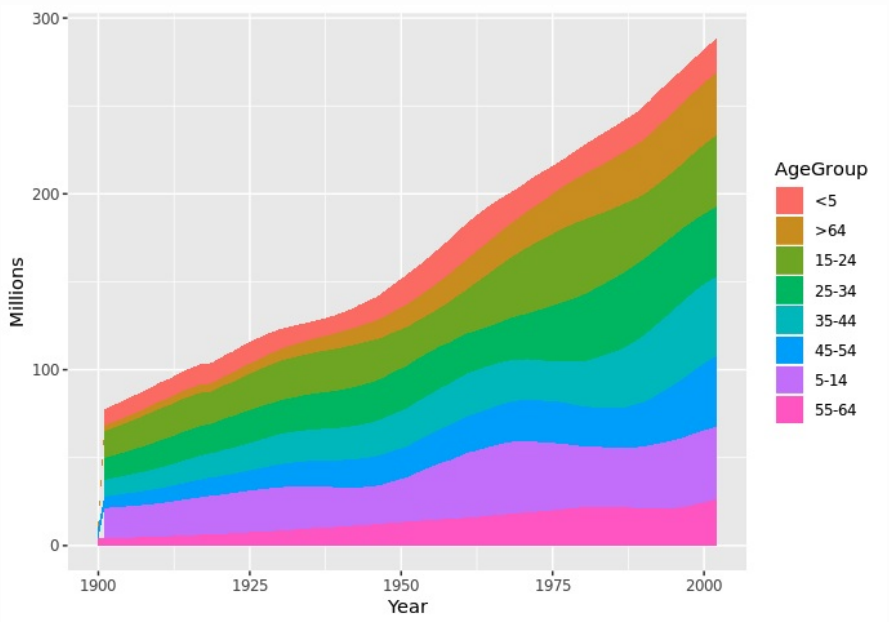
What is `cbind()`?

`cbind()` is a function that merges a particular vector as a last column to a data frame. For example, `df <- cbind(df, Millions)` merges the `Millions` vector with the `df` data frame and then stores all of that information back into `df`. When you print `df` now, you'll see `Millions` as the final column within the data frame.

Next, modify `chart` by substituting `Thousands` with `Millions`:

```
chart <- ggplot(df, aes(x = Year, y = Millions, fill =
        AgeGroup)) +
  geom_area()
print(chart)
```

Result:



Click this link to enlarge the image.

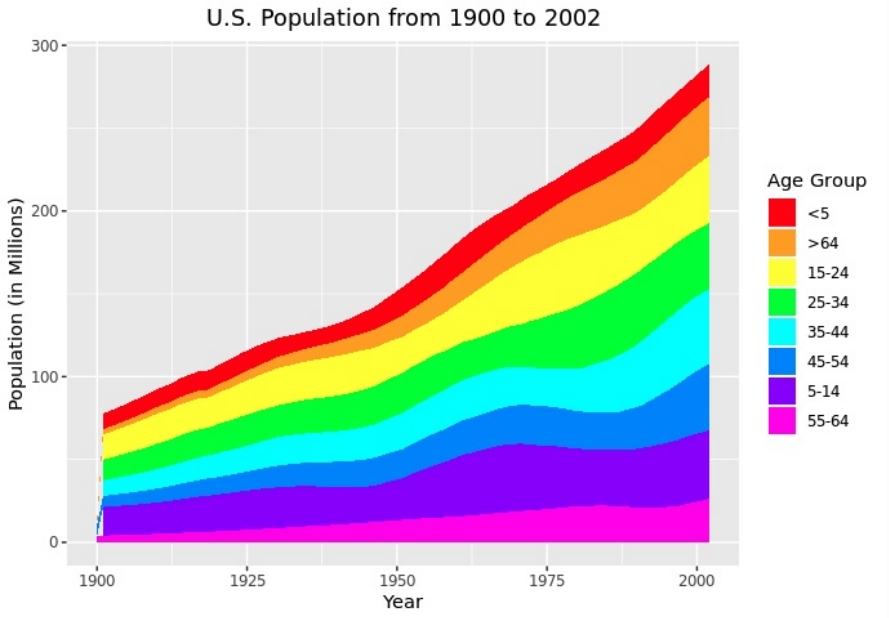
The chart looks much better now and is easier to understand.

Labels & Aesthetics

We can continue to use `labs()` and `scale_fill_manual()` to set our labels and colors.

```
chart <- ggplot(df, aes(x = Year, y = Millions, fill =
        AgeGroup)) +
  geom_area() +
  labs(title = "U.S. Population from 1900 to 2002",
       x = "Year",
       y = "Population (in Millions)",
       fill = "Age Group") +
  scale_fill_manual(values = c("#FF0000", "#FFA500", "#FFFF00",
        "#00FF00", "#00FFFF",
                               "#008FFF", "#8F00FF", "#FF00F0"))
print(chart)
```

Result:



Click this link to enlarge the image.

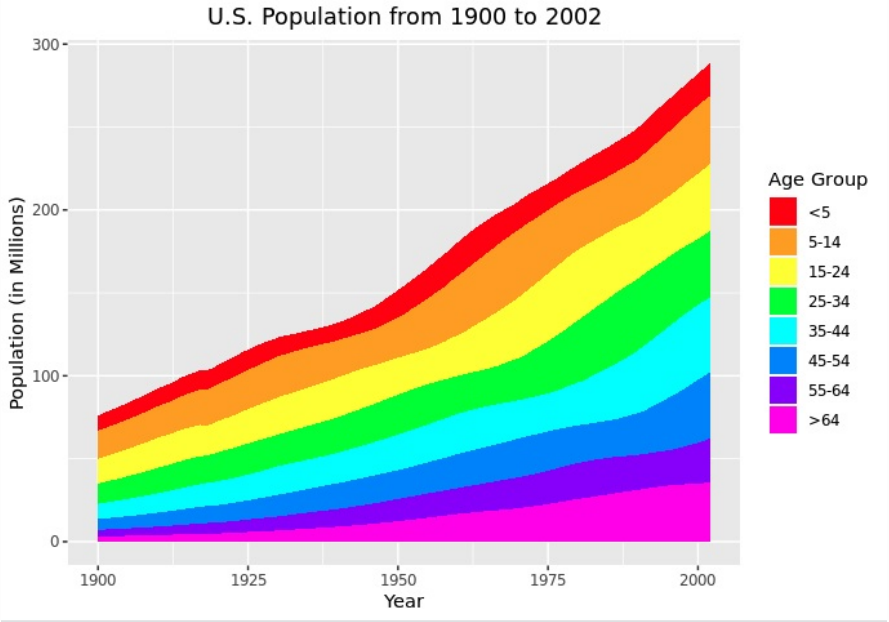
However, one thing that stands out is the order in which the age groups are presented. Notice how the age group `<5` is shown as the top-most layer followed by `>64`. This isn't necessarily bad, but usually we want to maintain some sort of order when it comes to age groups. To maintain the original order in which the age groups were presented, we can use the `factor()` function and modify the **data import** section so that it looks like this:

```
df <- read.csv("data/uspopage.csv")
Millions <- df$Thousands/1000
df$AgeGroup <- factor(df$AgeGroup,
                      levels = c("<5", "5-14", "15-24", "25-34",
                                 "35-44", "45-54", "55-64",
        ">64"))
df <- cbind(df, Millions)
```

Note that due to the `AgeGroup` labels being repeated multiple times, we needed to specify the maintenance order using `levels = c()`. The specified order goes within the parentheses. Due to the labels repeating, we are not able to use `df$AgeGroup <- factor(df$AgeGroup, levels = df$AgeGroup)` like we did similarly in a previous section.

Keep `chart` the same and you will see the following result when you print.

Result:



Click this link to enlarge the image.

Area Chart Versus Stacked Column Chart

You might notice that an area chart looks very similar to a stacked column chart. This is true because both charts show how each column of data is broken up into sub-categories. However, the “columns” in an area chart are continuous meaning the data on the x-axis looks like they “blend” into each other which is not necessarily the case for stacked column charts. This makes area charts great when showing data through **time** periods. For example, the area chart above makes it very easy to see that over time each sub-population has increased in number causing the overall population to grow going from the year 1900 to 2002.