Summary Data Visualization - 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Summary data visualization: topic introduction

In this part of the course, we will cover the following concepts:

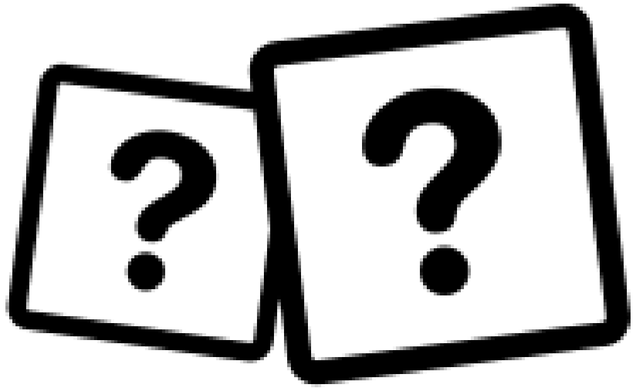
- Transform and summarize data for visualization
- Visualize the summary data

Warm-up activity: scavenger hunt

- Take 5 minutes to ***explore the plot options in the Highchart API documentation***
- Select one of the plotting needs below and determine what `plotOptions` **argument** can help you meet it:
  i. Suppose you want to **change the whisker colors** for a boxplot
  ii. Suppose you want to **stack the values of each series** in a column plot
  iii. Suppose you want to **add a hover state** to a particular series
- **Hint:** Find the chart type **first** under `plotOptions`, then look for the particular argument
  - For the **hover** example, you may want to look under a menu beginning with the letter `s`...
- Copy the URL to the relevant piece of documentation, and be prepared to share in the chat

Task 1

- How might you **change the whisker colors** for a boxplot?



Answer 1

- Use the `whiskerColor` argument to change the color of the whiskers
  - ***Read the complete documentation***
  - ***Try a demonstration*** by using the links in the documentation
- What happens when you substitute a different hex code for `whiskerColor`?

Since 3.0.0

`whiskerColor: Highcharts.ColorString, Highcharts.GradientColorObject, Highcharts.PatternObject`

The color of the whiskers, the horizontal lines marking low and high values. When `undefined`, the general series color is used.

In **styled mode**, the whisker stroke can be set with the `.highcharts-boxplot-whisker` class .

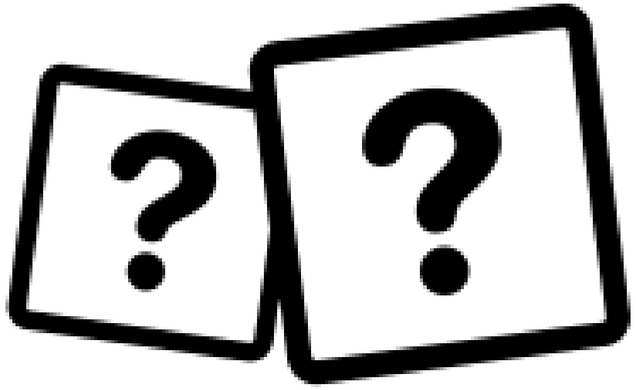
Defaults to `undefined`.

Try it

- **Box plot styling**
- **Box plot in styled mode**

Task 2

- How might you **stack the values of each series** in a column plot?



Answer 2

- Use the `stacking` argument to stack series values alongside one another
  - ***Read the complete documentation***
  - ***Try a demonstration*** by using the links in the documentation
- What happens when you change the `stacking` argument from `percent` to `normal`?

`stacking: string`

Whether to stack the values of each series on top of each other. Possible values are `undefined` to disable, `"normal"` to stack by value or `"percent"`.

When stacking is enabled, data must be sorted in ascending X order.

Some stacking options are related to specific series types. In the streamgraph series type, the stacking option is set to `"stream"`. The second one is `"overlap"`, which only applies to waterfall series.

Defaults to `undefined`.

Try it

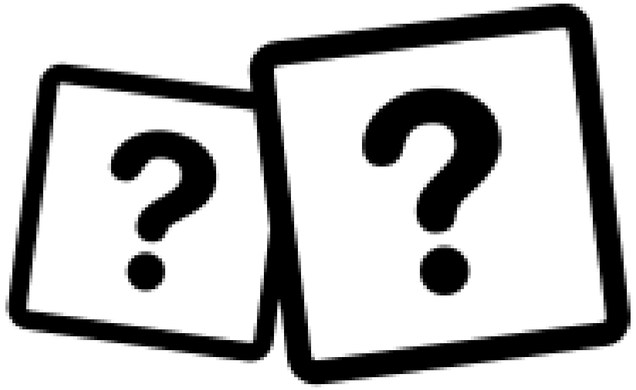
- Line
- Column
- Bar
- Area
- Line
- Column
- Bar
- Area
- Waterfall with normal stacking
- Waterfall with overlap stacking

See also

- yAxis.reversedStacks

Task 3

- How might you **add a hover state** to a particular series?



Answer 3

- Enable the `hover` argument to create a hover effect for a particular series' elements
  - ***Read the complete documentation***
  - ***Try a demonstration*** by using the links in the documentation
- What happens when you change the `enabled` argument from `false` to `true`?

`plotOptions.series.states.hover`

Options for the hovered series. These settings override the normal state options when a series is moused over or touched.

Since 5.0.8

animation

Animation setting for hovering the graph in line-type series.

See also

- Partial.<Highcharts.AnimationOptionsObject>

Since 1.2.0

enabled: boolean

Enable separate styles for the hovered series to visualize that the user hovers either the series itself or the legend.

Defaults to `true`.

Try it

- Line
- Column
- Pie

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	
Create interactive maps utilizing JSON files	

Building compound plots

- The charts we covered so far can be layered to bring out insights about the interaction of different variables
- We will create a compound density plot to highlight this functionality to compare a small subset of variables from our dataset
- Since those variables are at different scales, we will also need to normalize them for easy comparison and overlay

```
highchart() %>%                #<- main plot
  hc_chart( ... )  %>%         #<- global chart options to apply to all layers
  hc_add_series( ... )  %>% #<- plot an independent layer of data
  hc_add_series( ... )  %>% #<- plot another independent layer of data
  ...
  hc_xAxis( ... ) %>%          #<- adjust x-axis options (if necessary)
  hc_yAxis( ... ) %>%          #<- adjust y-axis options (if necessary)
  hc_tooltip( ... ) %>%        #<- adjust tooltip (if necessary)
  hc_plotOptions( ... ) %>% #<- adjust other plot options (if necessary)
  hc_legend( ... )  %>%        #<- adjust legend (if necessary)
  hc_title( ... )              #<- add/edit title (if necessary)
```

Directory settings

- In order to maximize the efficiency of your workflow, use the `box` package and encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your materials folder

```
# Set `main_dir` to the location of your materials folder.

path = box::file()
main_dir = dirname(dirname(path))
```

Directory settings (cont'd)

- We will store all datasets in the `data` directory inside the materials folder in your environment; hence we will save their path to a `data_dir` variable
- We will save all the plots in the `plots` directory corresponding to `plot_dir` variable
- To append one string to another, use `paste0` command and pass the strings you would like to paste together

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Load packages

- Let's load the packages we will need to use in this module

```
library(htmlwidgets)
library(tidyverse)
library(highcharter)
library(broom)
library(dplyr)
library(visNetwork)
```

Loading HDS dataset

- Let's load the HDS dataset from our `data_dir` into R's environment

```
# Read CSV file called "healthcare-dataset-stroke-data.csv"
HDS = read.csv(file = file.path(data_dir,"/healthcare-dataset-stroke-data.csv"), #<- provide file
path
               header = TRUE,            #<- if file has header set to TRUE
               stringsAsFactors = FALSE) #<- read strings as characters, not as factors
```

- We'll now remove the NA's in the dataset followed by creating a subset of the columns: `age, bmi, and avg_glucose_level`

```
# we can use unique() and sum(is.na()) function to know about NA values
HDS$bmi <- as.numeric(as.character(HDS$bmi))
# NA imputation
HDS$bmi[is.na(HDS$bmi)]<-mean(HDS$bmi,na.rm=TRUE)
# Let's make a vector of column indices we would like to save.
column_ids= select(HDS, age,bmi,avg_glucose_level)
HDS_subset = column_ids
```

Building compound plots

- Before building the layered plot of `age`, `bmi`, and `avg_glucose_level`, we want to avoid having different scales of data
- Normalization is the process used to convert the unevenly distributed data or the skewed data into a common scale without distorting differences in the ranges of values
- There are various normalization techniques like min max scaling, standard scaling, etc.; available in R
- In this module, we will normalize the variables between 0 and 1 so that their density plots can be layered and compared meaningfully

```
# Function to normalize data between 0 and 1.
normalize <- function(x)
{return ((x - min(x, na.rm = TRUE))/
            (max(x, na.rm = TRUE) -
               min(x, na.rm = TRUE)))}

HDS_subset$age <- normalize(HDS_subset$age)
HDS_subset$bmi <- normalize(HDS_subset$bmi)
HDS_subset$avg_glucose_level <- normalize(HDS_subset$avg_glucose_level)
```

Compound plots: density + lines example

- With the data normalized, we can build the density plot layer by layer

```
layered_density_interactive = highchart() %>%
  hc_chart(type = "area") %>%
  hc_add_series(data = density(HDS_subset$age, na.rm = TRUE),
                name = "Age") %>%
  hc_add_series(data = density(HDS_subset$bmi, na.rm = TRUE),
                name = "BMI") %>%
  hc_add_series(data = density(HDS_subset$avg_glucose_level, na.rm = TRUE),
                name = "Average_Glucose_Level") %>%
```

Compound plots: density + lines example (cont'd)

- We will now add the plotlines

```
hc_xAxis(plotLines = list(
  list(label = list(text = "Mean Age"),
       width = 2,
       color = "red",
       value = mean(HDS_subset$age)),
  list(label = list(text = "Mean BMI"),
       width = 2,
       color = "red",
       value = mean(HDS_subset$bmi)),
  list(label = list(text = "Mean Avg_Glucose_Level"),
       width = 2,
       color = "red",
       value = mean(HDS_subset$avg_glucose_level, na.rm = TRUE)))) %>%
```

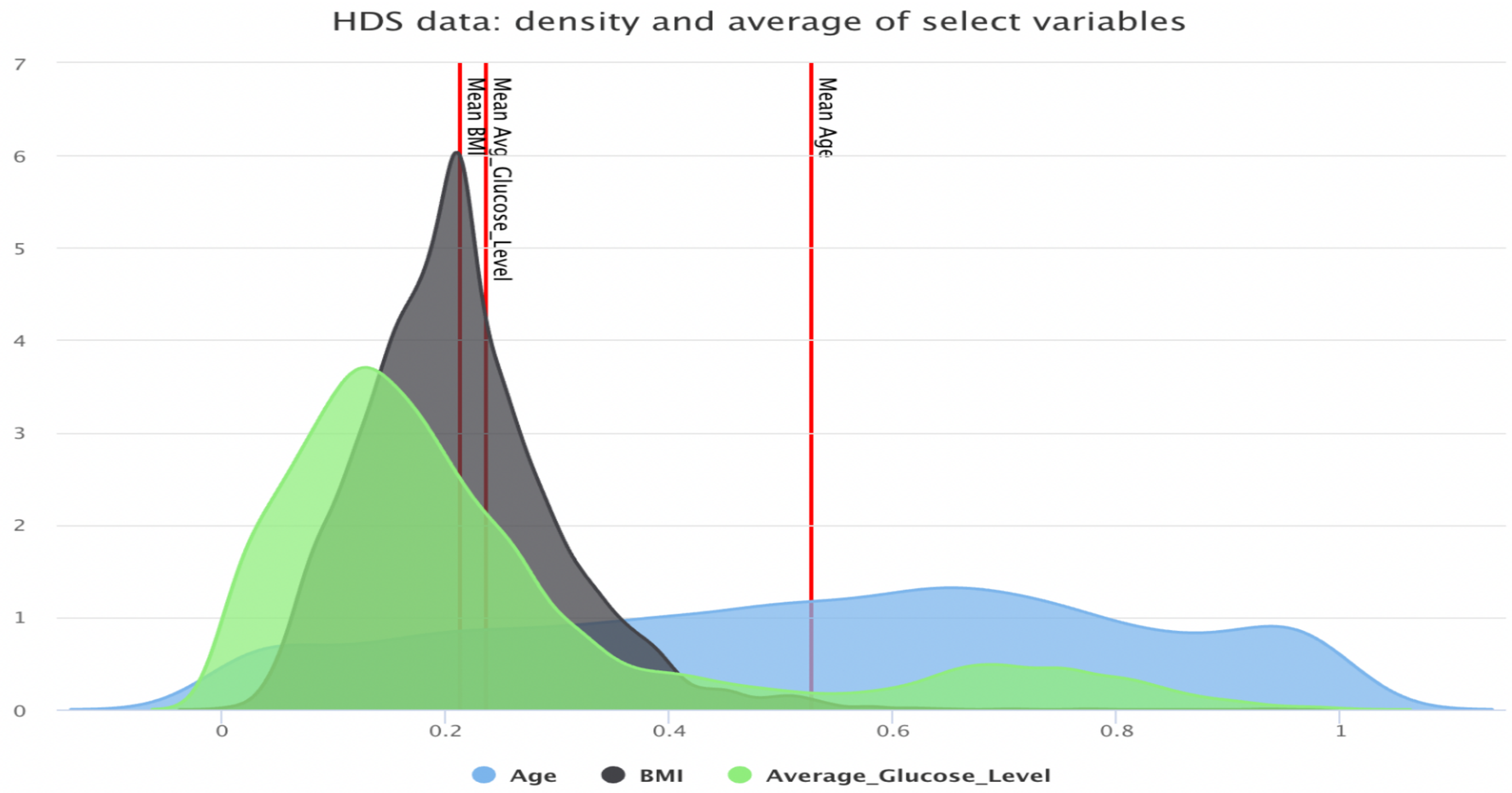
Compound plots: density + lines example (cont'd)

- We'll finally add the tooltip and title to the plot

```
hc_tooltip(crosshairs = TRUE) %>%
hc_title(text = "HDS data: density and average of select variables")
```

Layered density plot

```
layered_density_interactive
```



- What insights can you glean from this plot?

Layered density plot results

- The values of `BMI` and `Avg_Glucose_Level` seem to have a similar distribution, which helps us predict the level of `stroke`
- Layering different charts and variables is handy for uncovering variable interactions during exploratory data analysis

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	

Knowledge check

Exercise

You are now ready to try tasks 1-2 in the Exercise for this topic

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	✔

Summary data visualization: topic summary

In this part of the course, we have covered:

- Transforming and summarizing data for visualization
- Visualizing summary data

Congratulations on completing this module!