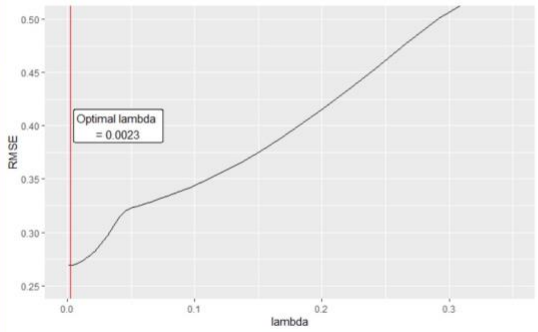
The goal of this presentation is to give you examples of a wide variety of graphs that are good for presenting the results of machine learning analysis to a non-technical audience, for example, in the executive summary of your midterm and final projects for this course. This is not an exhaustive list of all the graphs you could possibly make. But it should be a good overview to get you thinking in the right direction. As you're watching this presentation, be sure to check out the R Markdown document that accompanies this presentation to view the R code that I used to create each of these graphs.

Important note: Transcripts are **not** substitutes for textbook assignments.

Communicating the relative performance of different models/combinations of tuning parameters

Line or bar graph of performance



One frequent use of graphs is to communicate the relative performance of different models or different combinations of tuning parameters, so you can show which model did the best. You can do this with a line graph or a bar graph of the performance of the models. For example, in the line graph shown here, I'm showing the root mean squared error of a lasso model with different values of lambda displayed on the x-axis. And I'm using a red line to highlight the lowest value of the RMSE, which corresponds to the optimal lambda value, which was 0.0023.

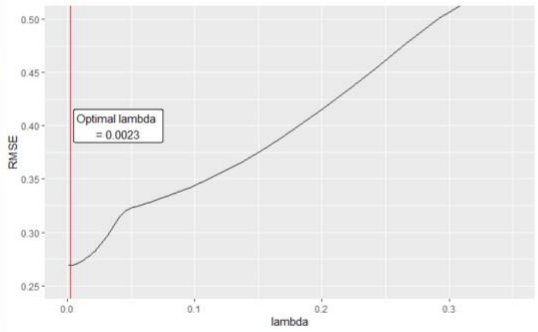
Notes:

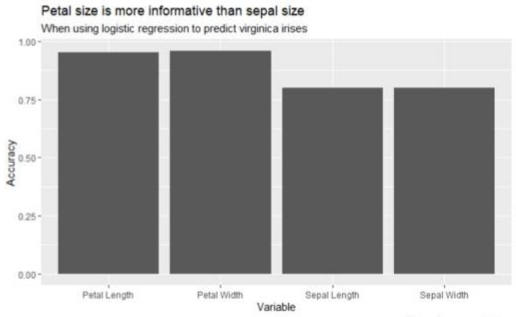
```
lambdalist <- 10^seq(-3, 1, length.out = 100)

set.seed(400)
ctrl = trainControl(method = "cv", number = 10)
fit_lasso = train(Petal.Length ~ .,
             data = iris,
             method = "glmnet",
             na.action = na.exclude, # this data set
doesn't actually have any NAs
             trControl = ctrl,
             tuneGrid = expand.grid(alpha = 1,
                                    lambda = lambdalist))

fit_lasso

gf_line(RMSE ~ lambda, data = fit_lasso$results) %>%
  gf_refine(coord_cartesian(xlim = c(0, 0.35), ylim =
c(0.25, 0.5))) %>%
```

```
  gf_vline(xintercept =~ fit_lasso$finalModel$lambdaOpt,
color = "red") %>%
  gf_label(0.4 ~ 0.04, label = paste("Optimal lambda \n=",
round(fit_lasso$finalModel$lambdaOpt, 4)))



iris2 <- iris %>%
  mutate(is_virginica = factor(ifelse(Species ==
"virginica", 1, 0)))

ctrl = trainControl(method = "cv", number = 10)
fit_petal_length = train(is_virginica ~ Petal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_sepal_length = train(is_virginica ~ Sepal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_petal_width = train(is_virginica ~ Petal.Width,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_sepal_width = train(is_virginica ~ Sepal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)

fit_is_virginica = rbind(fit_petal_length$results,
fit_sepal_length$results, fit_petal_width$results,
fit_sepal_width$results)
fit_is_virginica = data.frame(Variable = c("Petal Length",
"Sepal Length", "Petal Width", "Sepal Width"),
fit_is_virginica)
fit_is_virginica

fit_is_virginica %>%
  gf_col(Accuracy ~ Variable) %>%
  gf_labs(title = "Petal size is more informative than
sepal size",
          subtitle = "When using logistic regression to
predict virginica irises")
```

Communicating the relative performance of different models/combinations of tuning parameters

Line or bar graph of performance





A bar graph is more often used when we want to compare different models, rather than different values of a tuning parameter. So it makes more sense to have the x-axis be a categorical variable, which model was it, rather than a quantitative variable, such as the value of lambda.

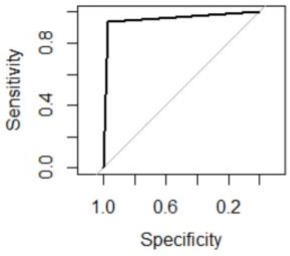
Notes:

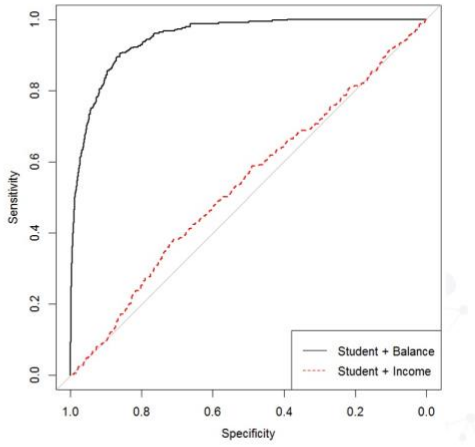
```
lambdalist <- 10^seq(-3, 1, length.out = 100)

set.seed(400)
ctrl = trainControl(method = "cv", number = 10)
fit_lasso = train(Petal.Length ~ .,
             data = iris,
             method = "glmnet",
             na.action = na.exclude, # this data set
doesn't actually have any NAs
             trControl = ctrl,
             tuneGrid = expand.grid(alpha = 1,
                                    lambda = lambdalist))

fit_lasso

gf_line(RMSE ~ lambda, data = fit_lasso$results) %>%
  gf_refine(coord_cartesian(xlim = c(0, 0.35), ylim =
c(0.25, 0.5))) %>%
  gf_vline(xintercept =~ fit_lasso$finalModel$lambdaOpt,
color = "red") %>%
  gf_label(0.4 ~ 0.04, label = paste("Optimal lambda \n=",
round(fit_lasso$finalModel$lambdaOpt, 4)))
```

```
iris2 <- iris %>%
  mutate(is_virginica = factor(ifelse(Species ==
"virginica", 1, 0)))

ctrl = trainControl(method = "cv", number = 10)
fit_petal_length = train(is_virginica ~ Petal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_sepal_length = train(is_virginica ~ Sepal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_petal_width = train(is_virginica ~ Petal.Width,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)
fit_sepal_width = train(is_virginica ~ Sepal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)

fit_is_virginica = rbind(fit_petal_length$results,
fit_sepal_length$results, fit_petal_width$results,
fit_sepal_width$results)
fit_is_virginica = data.frame(Variable = c("Petal Length",
"Sepal Length", "Petal Width", "Sepal Width"),
fit_is_virginica)
fit_is_virginica

fit_is_virginica %>%
  gf_col(Accuracy ~ Variable) %>%
  gf_labs(title = "Petal size is more informative than
sepal size",
          subtitle = "When using logistic regression to
predict virginica irises")
```

ROC curves

- Compare multiple models for a binary response
- Use predicted probabilities, not predicted classes
- Nice touch: Add AUCs using *text()* or *gf_label()*





When your response variable is binary, you can use a ROC curve to compare models. Remember that ROC curves are most effective when they're used to compare multiple models, not just showing the performance of a single model, and that they can be used with any model that's predicting a binary response. They don't have to be used with logistic regression.

One important thing to keep in mind when you're computing a ROC curve is to use the predicted probabilities of belonging to one of the classes, not the predicted classes themselves. This is a common mistake when you're working with multiple modeling types and going back and forth between using caret and not using caret. If you end up with a ROC curve that looks like this, that basically looks like a couple of straight lines stitched together, think carefully about whether you created the ROC curve correctly.

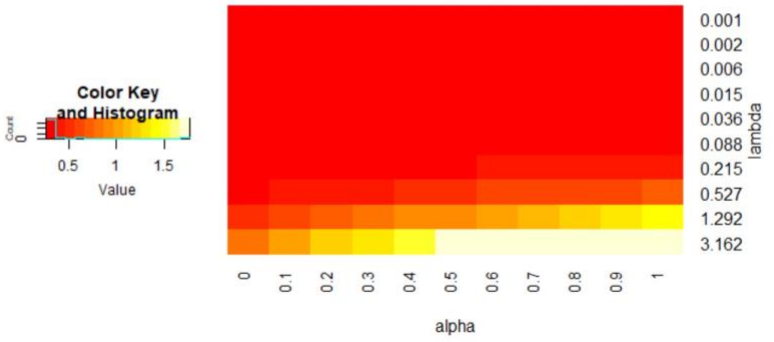
You could get a ROC curve that looks like this if you made a single decision tree that doesn't have very many branches in it. But, otherwise, it's very common for this type of ROC curve to occur because you accidentally created the ROC curve using the predicted classes, that is, zeros and ones, rather than the probability that each observation belongs to class one. Another thing that can be a nice touch to add to your ROC curve is to add the area under the curve to the graph using either the text function or the gf_label function.

Notes:

```
library(pROC)
myroc = roc(response=Default$default,
predictor=predictvals)
myroc2 = roc(response=Default$default,
predictor=predictvals2)
plot.roc(myroc)
plot.roc(myroc2, add=T, col="red", lty=2)
legend("bottomright", legend=c("Student + Balance",
"Student + Income"),lty=c(1,2), col=c("black","red"))
```

Heat map

- Shows the performance of a model across all combinations of 2 tuning parameters



If you want to show the performance of a model across combinations of two different tuning parameters, such as alpha and lambda for an elastic net model, a heat map can be a good way to do that. In this example, we're graphing the root mean squared error of an elastic net model. Red corresponds to lower values. So it's looking like we want a lower value of lambda.

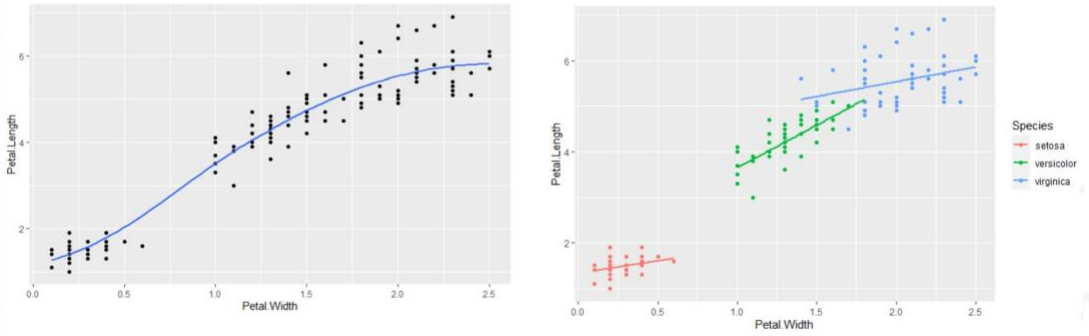
Notes:

```
set.seed(400)
lambdalist <- 10^seq(-3, 0.5, length.out = 10)
alphalist <- seq(0, 1, by = 0.1)

ctrl = trainControl(method = "cv", number = 10)
fit_enet = train(Petal.Length ~ .,
             data = iris,
             method = "glmnet",
             na.action = na.exclude, # this data set
doesn't actually have any NAs
             trControl = ctrl,
             tuneGrid = expand.grid(alpha = alphalist,
                                    lambda = lambdalist))

enet_mat <- fit_enet$results %>%
  select(alpha, lambda, RMSE) %>%
  pivot_wider(names_from = alpha, values_from = RMSE)

enet_mat = as.matrix(enet_mat)
row.names(enet_mat) = round(enet_mat[ ,1],3)
```

```
enet_mat = enet_mat[ , -1]

library(gplots)
heatmap.2(enet_mat, Rowv = FALSE, Colv = FALSE,
          dendrogram = "none", trace = "none",
          xlab = "alpha", ylab = "lambda")
```

Communicating the relationship between the response and one or more of the predictors

- Scatterplot with smoothing line or curve



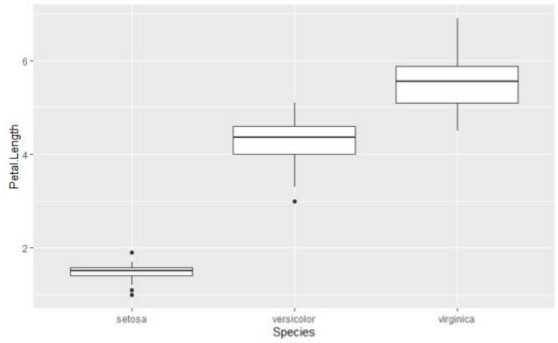
When you're interpreting your model, it can be helpful to use graphs that communicate the relationship between the response variable and one or more of the important predictor variables. If your response variable and one or more of your predictor variables are quantitative, then a scatterplot can be a good way to do this. We always graph the response variable on the y-axis.

So, here, the points are showing the actual observed values of petal length and petal width. And the smoothing curve is showing an approximate predicted value of petal length for each given value of petal width. But this is just based on a smoothing curve not directly based on the model that we built. If we want to incorporate a categorical variable as an additional predictor, we can do that using colors and creating a separate smoothing line for each of the three species.

Notes:

```
iris %>%
  gf_point(Petal.Length ~ Petal.Width) %>%
  gf_smooth(Petal.Length ~ Petal.Width)

iris %>%
  gf_point(Petal.Length ~ Petal.Width, col =~ Species) %>%
  gf_smooth(Petal.Length ~ Petal.Width, col =~ Species,
method = "lm")
```

Boxplots

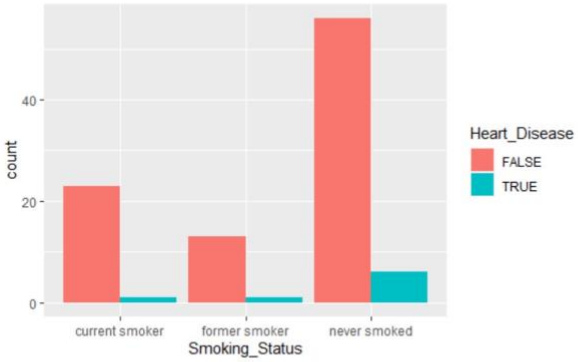


If either the response or predictor variable is categorical and the other one is quantitative, then a box plot can be a good way to show their relationship. You could also use a violin plot here.

Notes:

```
iris %>%
  gf_boxplot(Petal.Length ~ Species)
```

Conditional bar graphs



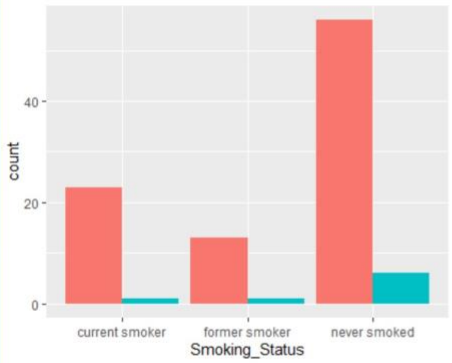
If both your response and predictor variables are categorical, then a bar graph can be useful for showing their relationship. But I find that bar graphs of two variables can be difficult to interpret when one of the groups is much more common than the others. For example, in this graph, the group never smoked is much more common than the groups current smoker or former smoker. And it's easy to see that the bars for heart disease false and heart disease true are both taller in the never smoked group than they are in the other two groups. But it's not really clear whether people who have never smoked are more or less likely to have heart disease than the other two groups.

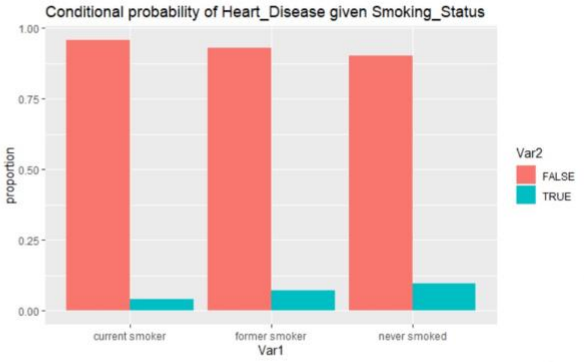
Notes:

```
library(simexaft) # for BHS data
data(BHS)

BHS <- BHS %>%
  mutate(Heart_Disease = ifelse(CHID == 1, TRUE, FALSE),
         Smoking_Status = case_when(
           SMOKE2 == 1 ~ "current smoker",
           SMOKE1 == 1 ~ "former smoker",
           TRUE ~ "never smoked") )

BHS %>%
  gf_bar(~ Smoking_Status, fill =~ Heart_Disease,
         position = position_dodge())
```

```
# Download conditional_bar.R from Canvas, then run the
following.
source("conditional_bar.R")

conditional_bar("Smoking_Status", "Heart_Disease", BHS)
```

Conditional bar graphs





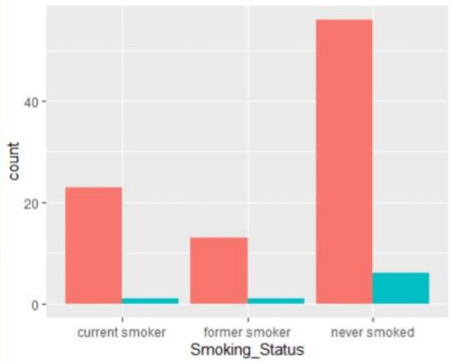
One solution to this is to use a conditional bar graph. That is, instead of having the y-axis show the count or the actual number of people in each category, we instead let the y-axis show the proportion of people out of each group, current smoker, former smoker, and never smoked, who have heart disease equal to false or equal to true. Now, when the heights of bars in each group add up to 1 or 100%, we can more easily see that the never smoked group has the highest bar for heart disease equal to true, showing us that people who have never smoked are more likely to have heart disease than current smokers or former smokers.

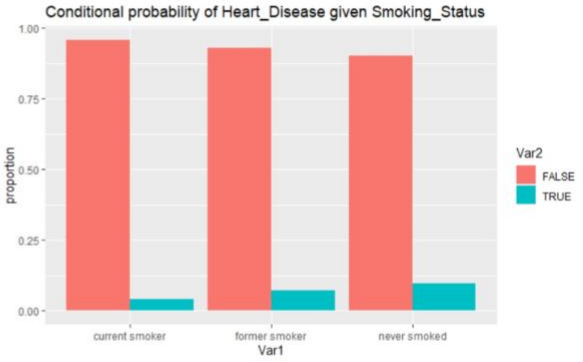
Notes:

```
library(simexaft) # for BHS data
data(BHS)

BHS <- BHS %>%
  mutate(Heart_Disease = ifelse(CHID == 1, TRUE, FALSE),
         Smoking_Status = case_when(
           SMOKE2 == 1 ~ "current smoker",
           SMOKE1 == 1 ~ "former smoker",
           TRUE ~ "never smoked") )

BHS %>%
  gf_bar(~ Smoking_Status, fill =~ Heart_Disease,
         position = position_dodge())
```

```
# Download conditional_bar.R from Canvas, then run the
following.
source("conditional_bar.R")

conditional_bar("Smoking_Status", "Heart_Disease", BHS)
```

Conditional bar graphs





Does this result surprise you? If so, maybe it's because this graph is only looking at the relationship between heart disease and smoking status. It's not accounting for any of the many other variables in the data that might be different between people with different smoking statuses and might influence the probability of heart disease. In other words, the results of this graph might not actually agree with the way our model is using smoking status at all.

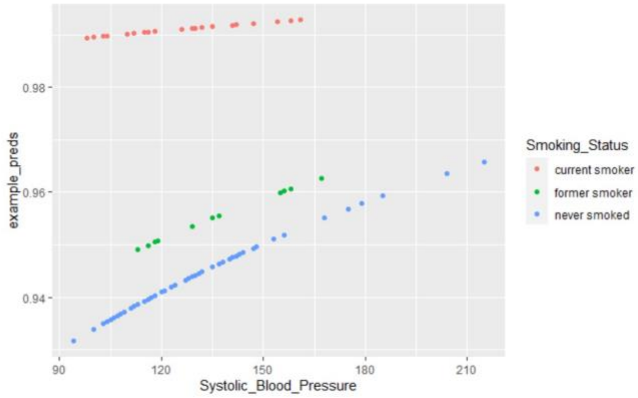
Notes:

```
library(simexaft) # for BHS data
data(BHS)

BHS <- BHS %>%
  mutate(Heart_Disease = ifelse(CHID == 1, TRUE, FALSE),
         Smoking_Status = case_when(
           SMOKE2 == 1 ~ "current smoker",
           SMOKE1 == 1 ~ "former smoker",
           TRUE ~ "never smoked") )

BHS %>%
  gf_bar(~ Smoking_Status, fill =~ Heart_Disease,
         position = position_dodge())


# Download conditional_bar.R from Canvas, then run the
following.
source("conditional_bar.R")
```

```
conditional_bar("Smoking_Status", "Heart_Disease", BHS)
```

Communicating the relationship between the predicted response and one or more of the predictors

- Graph predictions as a function of one or two variables, holding other variables constant



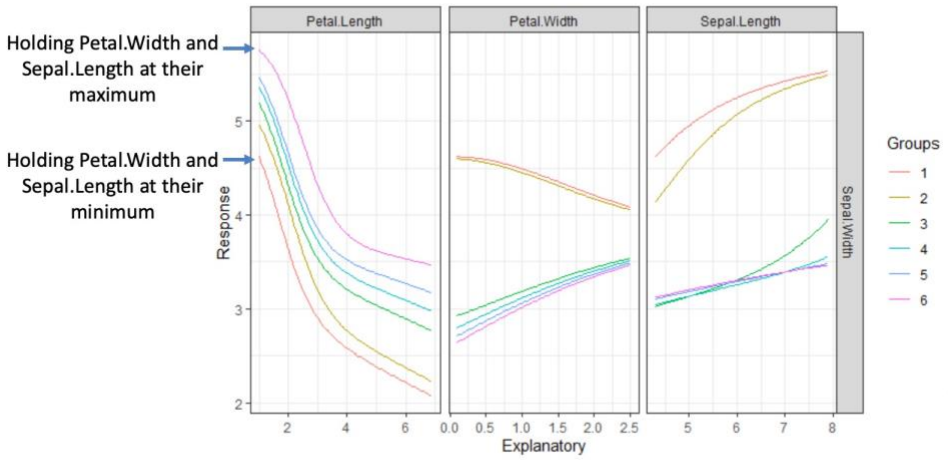
To demonstrate how your model is actually using the predictor variable, such as smoking status, it can be useful to have a graph that communicates the relationship between the predicted response and one or more of the predictor variables. A good way to do this is to graph the predictions as a function of one or two variables, holding all the other predictor variables constant. For example, here we're graphing the predicted probability of heart disease as a function of systolic blood pressure on the x-axis and smoking status shown by the different colors. And all of the other predictor variables are being held constant at their median for a quantitative variable or at their mode for a categorical variable. And, here, we can see that as we move from never smoked, to former smoker, to current smoker, the predicted probability of heart disease does indeed increase.

Notes:

```
library(simexaft)
data(BHS)
BHS2 <- BHS %>%
  mutate(Diastolic_Blood_Pressure = DBP,
         Systolic_Blood_Pressure = SBP,
         Heart_Disease = factor(ifelse(CHID == 1, TRUE,
FALSE)),
         Smoking_Status = factor(case_when(
           SMOKE2 == 1 ~ "current smoker",
           SMOKE1 == 1 ~ "former smoker",
           TRUE ~ "never smoked") )) %>%
  mutate(DRINKING = factor(DRINKING),
```

```
        ) %>%
  select(-c(PAIR, DBP, SBP, CHID,
            SMOKE1, SMOKE2, SMOKE,
            DTHCENS, CHDCENS, CVDCENS,
            DIABETES, RXHYPER))

set.seed(17)
ctrl = trainControl(method = "cv", number = 10)
fit_logistic = train(Heart_Disease ~ .,
             data = BHS2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)

# Thanks to
https://stackoverflow.com/questions/2547402/how-to-find-
the-statistical-mode for this elegant solution.
find_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

example_data <- BHS2 %>%
  mutate(across(where(is.numeric) & c(-
Systolic_Blood_Pressure), median)) %>%
  mutate(across(where(is.factor) & c(-Smoking_Status),
find_mode))

example_data <- example_data %>%
  mutate(example_preds = predict(fit_logistic,
example_data, type = "prob")[,1])

example_data %>%
  gf_point(example_preds ~ Systolic_Blood_Pressure, col =~
Smoking_Status)
```

Lek Profile



A Lek profile for artificial neural networks is a fancy way of doing exactly what we saw on the previous slide, graphing the predicted response as a function of one predictor variable at a time, holding all other predictor variables constant. If we focus on just the leftmost panel of this graph, we see that the response variable is sepal width as shown on the y-axis on the right-hand side. And the predictor variable that we're changing is petal length as shown at the top. Then each of the curves of different colors is holding the other two predictor variables constant at a different value.

The pink curve for group 6 is holding petal width and sepal length constant at their maximum values from this data set. And the peach curve for group 1 is holding them constant at their minimum. So this looks more complicated than what we did before because we're testing out multiple different values where we're holding the variables constant, not just their median.

And we're showing the effects of petal length, petal width, and sepal length all in the same illustration. So this is sort of like 18 different graphs in one. But, really, it's doing exactly the same thing as what we saw before. We're graphing the predicted response as a function of one variable, holding the other predictor variables constant.

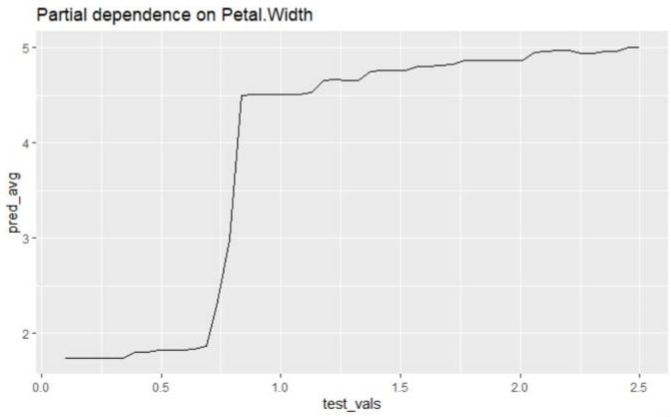
Notes:

```
library(nnet)
library(NeuralNetTools)

set.seed(100)
ctrl = trainControl(method = "cv", number = 5)
```

```
fit_sepal = train(Sepal.Width ~ .-Species,
                  data = iris,
                  method = "nnet",
                  tuneGrid = expand.grid(size = 1:5,
                  decay = c(0, .5, 10^(-c(1:7)))),
                  preProc = c("center", "scale"),
                  linout = TRUE,
                  maxit = 500,
                  trace = FALSE,
                  trControl = ctrl)

lekprofile(fit_sepal)
```

Partial Dependence Plot

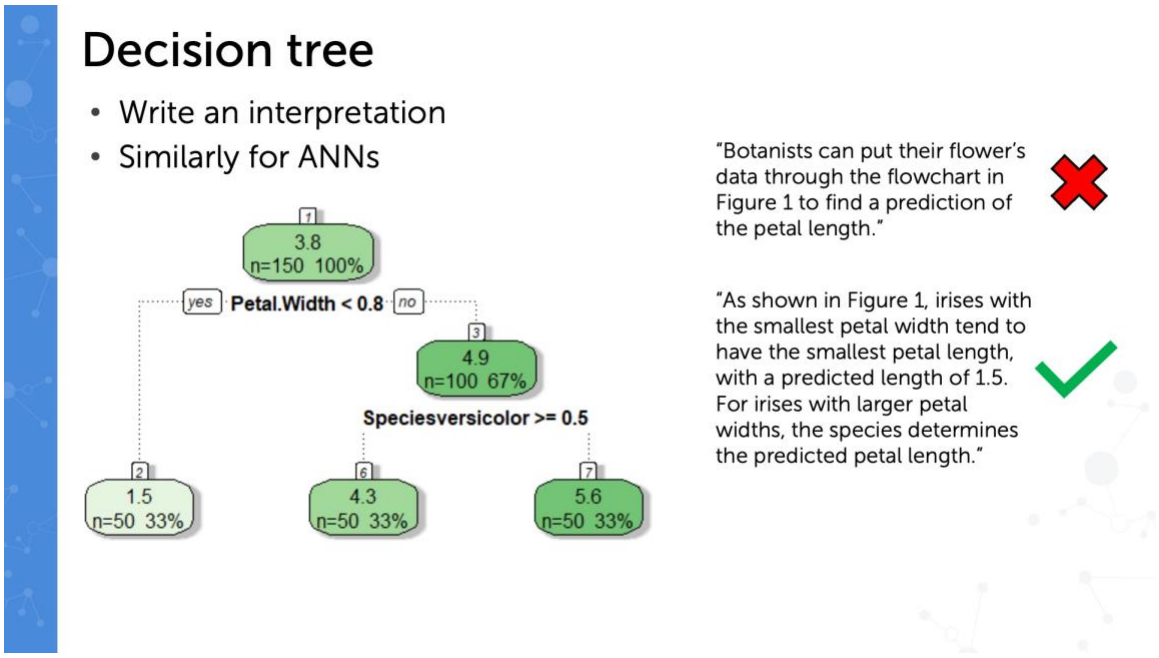
- Computes predicted response for possible values of other variables
- Takes average



A partial dependence plot for random forests is similar to the graphs that we just saw in that they're looking at the predicted response as a function of one predictor variable at a time. But instead of holding all the other predictors constant at one specific value, such as the median, it instead computes the predicted response for a range of possible values of the other variables and then takes the average response across those possible values. In this graph, we see that as the petal width of an iris flower increases, the predicted petal length also increases. But it doesn't do so in a linear fashion. Instead, it's relatively flat for a ways and then has a steep increase right around when the petal width is changing from say 0.6 to 0.8.

Notes:

```
set.seed(800)
ctrl = trainControl(method = "cv", number = 5)
fit_rf = train(Petal.Length ~ .,
               data = iris,
               method = "rf",
               tuneGrid = expand.grid(mtry = c(1:4)),
               trControl = ctrl)

fit_rf

source("gf_partialPlot.R")
gf_partialPlot(fit_rf, iris, x.var = "Petal.Width")
```



For a few modeling types, you can communicate the relationship between the predicted response variable and the predictor variables by showing the model. This is especially true if you've made a single decision tree. You can show the decision tree. When you do this, be sure to write an interpretation of the model.

For example, don't just write botanists can put their flowers data through the flowchart in figure 1 to find a prediction of the petal length. That's true. But it doesn't really help a non-technical audience understand what the graph is showing them about which variables are most important and in what ways.

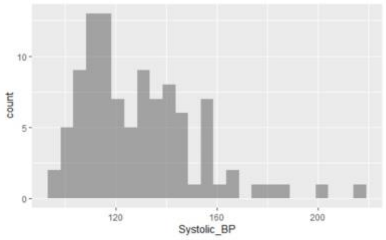
Instead, for this graph, you could write something like as shown in figure 1, irises with the smallest petal width tend to have the smallest petal length with a predicted length of 1.5. For irises with larger petal width, the species determines the predicted petal length. This is also true for artificial neural networks if you are displaying a graph of the nodes in that neural network.

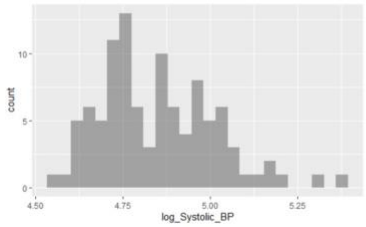
Notes:

```
set.seed(789)
ctrl = trainControl(method = "cv", number = 5)
fit_tree = train(Petal.Length ~ .,
                 data = iris,
                 method = "rpart",
                 na.action = na.exclude,
                 tuneGrid = expand.grid(cp = seq(.05, .1, by =
.01)),
                 trControl = ctrl)
```

```
fit_tree

rattle::fancyRpartPlot(fit_tree$finalModel)
```

Communicating why you chose the data cleaning that you did





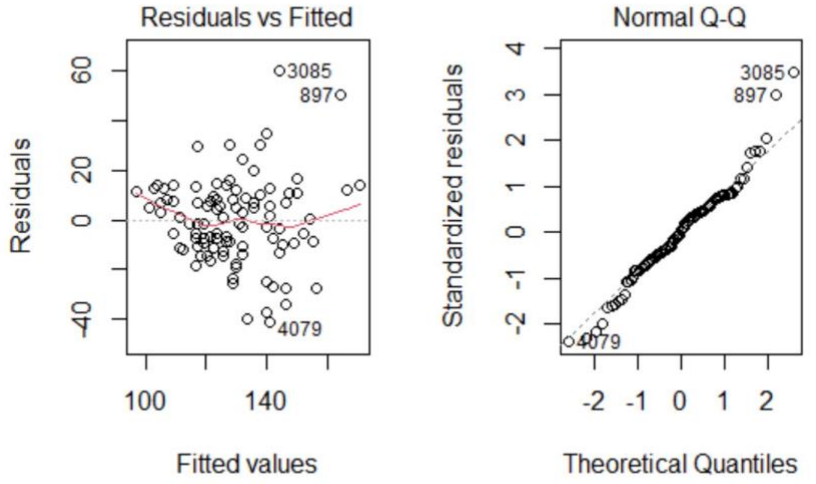
- Dealing with skew
  - Before-and-after histograms
    - Counts as 1 graph

Another use of graphs is to communicate why you chose the data cleaning that you chose. If you want to show why you dealt with skew in the way that you did, one way you can do this is with before and after histograms, showing a particular variable before you did the cleaning and after you did the transformation. For purposes of the projects in this class, a before and after histogram will count as a single graph.

Notes:

```
library(simexaft) # for BHS data
data(BHS)
BHS <- BHS %>%
  mutate(Systolic_BP = SBP,
         log_Systolic_BP = log(SBP))

BHS %>%
  gf_histogram(~Systolic_BP)
BHS %>%
  gf_histogram(~log_Systolic_BP)
```

Residual plots



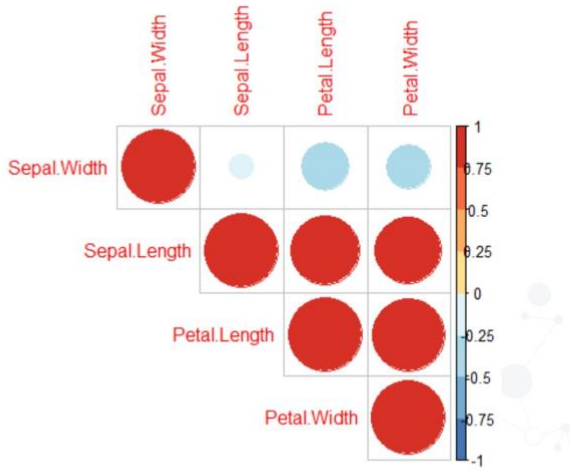
Showing the residual plots from linear regression can also be helpful for communicating why you chose to transform a variable to reduce skew or why robust regression is appropriate for a particular data set.

Notes:

```
library(simexaft) # for BHS data
data(BHS)
BHS <- BHS %>%
  mutate(Systolic_BP = SBP,
         Diastolic_BP = DBP)

fit_lm_BP = lm(Systolic_BP ~ Diastolic_BP, data = BHS)
par(mfrow = c(1,2))
plot(fit_lm_BP)
```

De-correlating predictors

- Correlation plots
  - Remember that strong correlations with the response are good

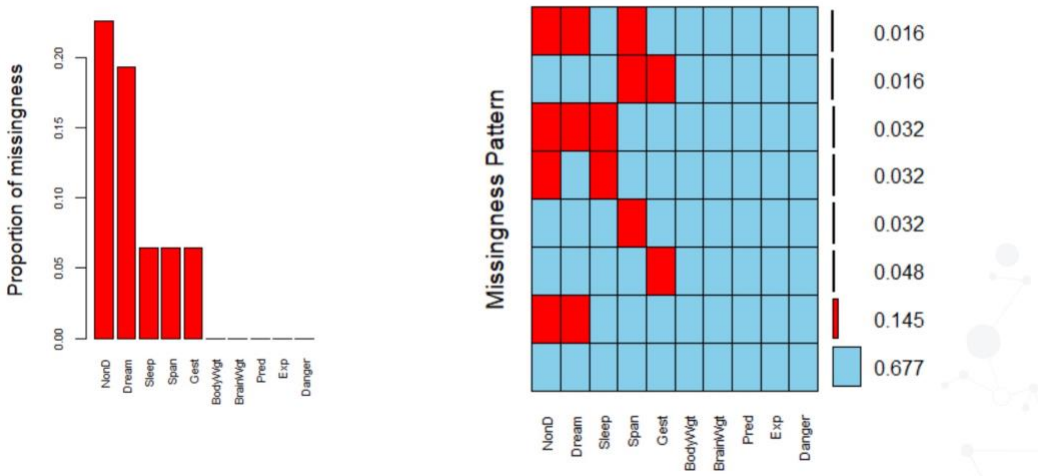


A correlation plot can be useful for showing why you removed a highly correlated predictor variable or why you replaced it by its residuals. When you're interpreting the correlation plots, remember that strong correlations between a predictor variable and the response variable are not a sign of multicollinearity. Instead, they're a good sign that that predictor variable is going to be highly informative for making good predictions about the response. In this correlation graph, I've reversed the color scheme so that red would correspond to a correlation of one and blue would correspond to a correlation of negative 1. For me, it's more intuitive to think of warm colors as being positive.

Notes:

```
library(corrplot)
library(RColorBrewer) # for the Brewer color pallette
iris_numeric = select_if(iris, is.numeric)
correlations <-cor(iris_numeric, use =
"pairwise.complete.obs")
corrplot(correlations, type="upper", order="hclust",
         col=rev(brewer.pal(n=8, name="RdYlBu")))
```

Dealing with missingness



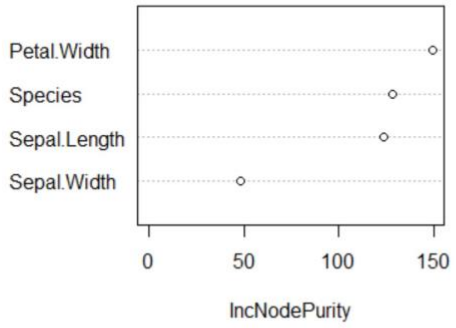
If you want to communicate why you chose to remove or impute missing data, a graph of the missingness can be helpful. Here we have a bar graph of the proportion of each predictor variable that was missing. And on the right, we have a graph of the patterns of missingness shown in the data showing that 67.7% of the rows of the data had no missing values. 14.5% of the data had missing values in the variables NonD and Dream, but no other variables and so on.

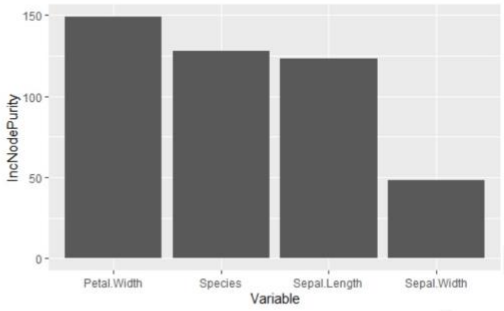
Notes:

```
library(VIM)
aggr(sleep, numbers=TRUE, sortVars=TRUE, cex.axis=.7,
gap=3,
     ylab=c("Proportion of missingness","Missingness
Pattern"))
```

Communicating which predictors are most important in the best model

- Variable importance graph





Many of the modeling types in this course feature a built-in variable importance graph, which allows you to communicate which predictor variables are most important in the best model. The default variable importance graph for many of these modeling types isn't super attractive. So you can also extract this information and create your own bar graph of the variable importance using ggformula or ggplot2.

Notes:

```
fit_rf = randomForest(Petal.Length ~ ., data = iris)
varImpPlot(fit_rf)

imp = data.frame(Variable = rownames(fit_rf$importance),
                 IncNodePurity = fit_rf$importance)
imp %>%
  mutate(Variable = reorder(Variable, -IncNodePurity)) %>%
  gf_col(IncNodePurity ~ Variable)
```

Depending on the modeling type, you may access the variable importance differently. So it's important to think carefully about your model and read the documentation where appropriate. For example, for an artificial neural network. you can use the Garson plot function.

But for linear regression, you'd want to use the p-values from the linear regression results. I prefer to have my more important variables be larger on the variable importance graph rather than smaller, close to zero. So I like to use the negative log base 10 of the p-values instead.

Notes:

```
library(nnet)
set.seed(100)
iris2 = data.frame(scale(iris[ ,1:4]))
fit_nnet = nnet(Sepal.Width ~ Petal.Length + Sepal.Length +
Petal.Width,
            data = iris2, size = 2, maxit = 400,
            linout = TRUE)
library(NeuralNetTools)
garson(fit_nnet)


iris2 = data.frame(scale(iris[ ,1:4]))
fit_lm = lm(Sepal.Width ~ Petal.Length + Sepal.Length +
Petal.Width, data = iris2)
```

```
results = summary(fit_lm)
lm_coef = data.frame(variable = rownames(results$coef)[-1],
p_value = results$coef[-1,4])
lm_coef %>%
  mutate(variable = reorder(variable, p_value)) %>%
  gf_col(-log10(p_value) ~ variable) %>%
  gf_hline(yintercept = -log10(.05), col = "red", lty = 2)
%>%
  gf_labs(title = "Sepal length and Petal length are most
important",
          subtitle = "Bars above the red line are
significant at the 0.05 level")
```

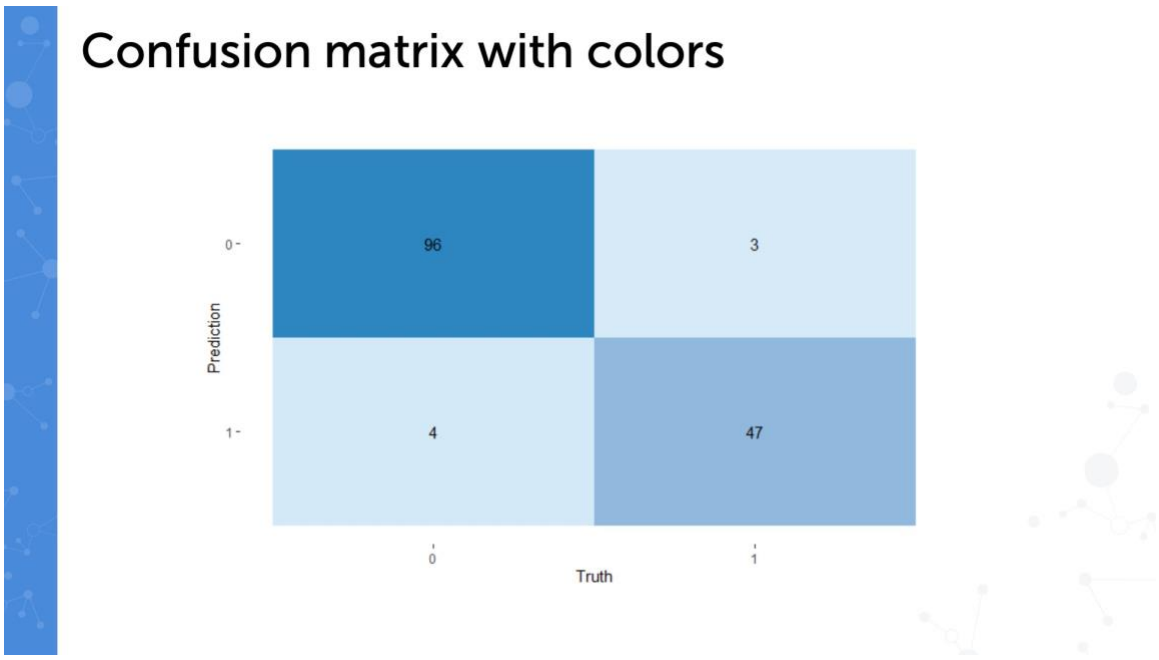
These are tables, not graphs

- Table of variables

Variable	Type	Range
Petal width	Quantitative	0.1-2.5
Sepal length	Quantitative	4.3-7.9
Sepal width	Quantitative	2.0-4.4
Species	Categorical	setosa, versicolor, virginica

For the purposes of the projects in this course, tables do not count as graphs. For example, in this table of information about predictor variables, the colors and the spatial layout helps to make the table more readable. But it doesn't really communicate any additional information beyond what's already in the text. So this counts as a table not as a graph.

If you're thinking about including a table like this, think about whether you have something to say about it in the text of your executive summary. If this table is communicating something useful about why you chose the data cleaning that you did or that will help you interpret your model, then go for it. But if you're just including a table to take up space, don't bother. Your audience can read which variables are in the data set by looking at the data set itself if they need to.



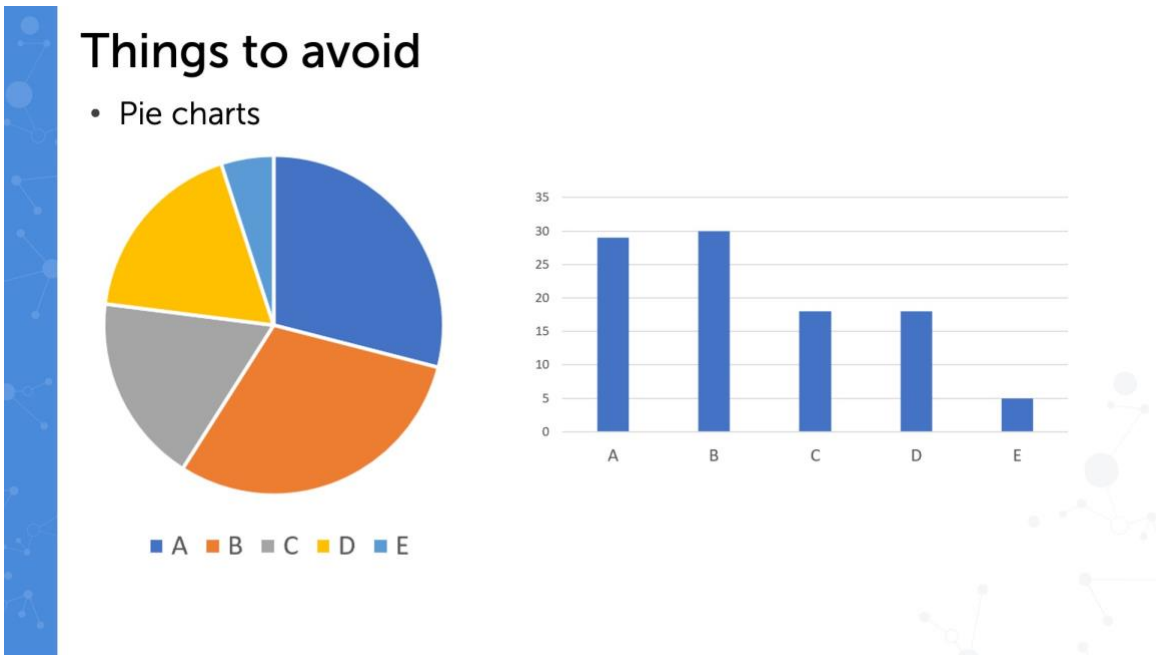
A confusion matrix with colors is on the borderline between a table and a graph. The colors do communicate some information about the magnitude of the number in each of the four cells. But it's not a lot of added information.

And the confusion matrix only tells us about the accuracy of one model. It doesn't help us compare the relative accuracy of multiple models. And it doesn't help us interpret the most important predictor variables. So for purposes of the projects in this class, a confusion matrix with colors counts as a table.

Notes:

```
iris2 <- iris %>%
  mutate(is_virginica = factor(ifelse(Species ==
"virginica", 1, 0)))

ctrl = trainControl(method = "cv", number = 10)
fit_petal_length = train(is_virginica ~ Petal.Length,
             data = iris2,
             method = "glm",
             family = "binomial",
             trControl = ctrl)

preds = predict(fit_petal_length)
results = data.frame(preds, truth = iris2$is_virginica)
```

```
cm <- yardstick::conf_mat(results, truth, preds)

autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low="#D6EAF8",high = "#2E86C1")
```



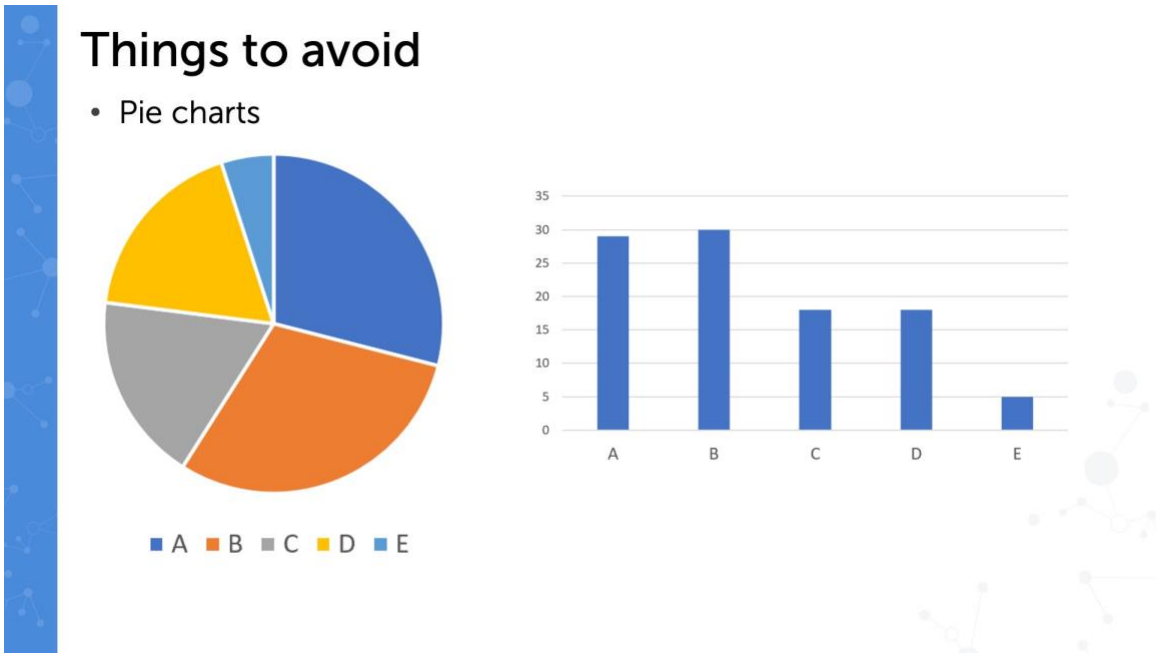
As you're planning your projects for this course, there are a few things to avoid. One of them is pie charts. The human eye is really bad at judging angles.

For example, in the pie chart shown here, can you tell which group is bigger, group A or B? I can't either. And I'm the one who made the chart. In contrast, if we display the same data using a bar graph, we can easily see that group B is slightly larger than group A.

Notes:

Problems with the pie charts
https://www.data-to-viz.com/caveat/pie.html
https://bernardmarr.com/default.asp?contentID=1779.

3d data visualizations
https://www.data-to-viz.com/caveat/3d.html



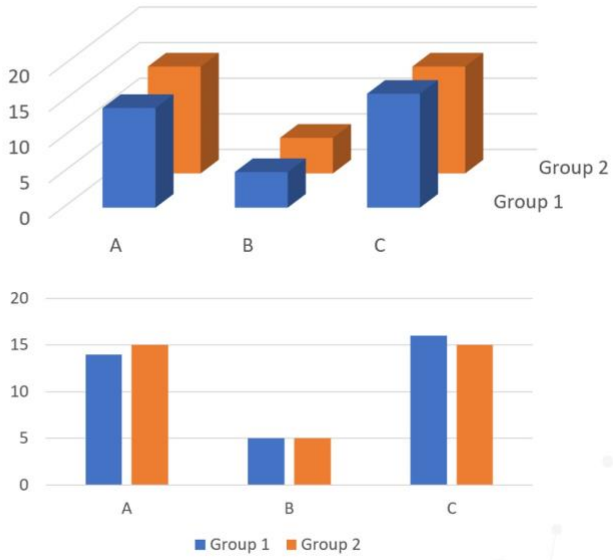
This challenge with interpreting pie charts means that you should use them with caution even in places like dashboards where your audience may really like them. You should only use them when you want to compare just a few categories, two or three, certainly not more than five. And you should always try making a bar chart of the same data. And then decide which one is more informative.

Notes:

Problems with the pie charts
https://www.data-to-viz.com/caveat/pie.html
https://bernardmarr.com/default.asp?contentID=1779.

3d data visualizations
https://www.data-to-viz.com/caveat/3d.html

3d bar graphs



One type of graph that you should avoid not just in this class but in your entire data science life are three-dimensional bar graphs. The three-dimensional perspective on these graphs makes it impossible to fairly interpret and compared data. For example, in the graph shown here, it looks like category A in group one has a height of 10.

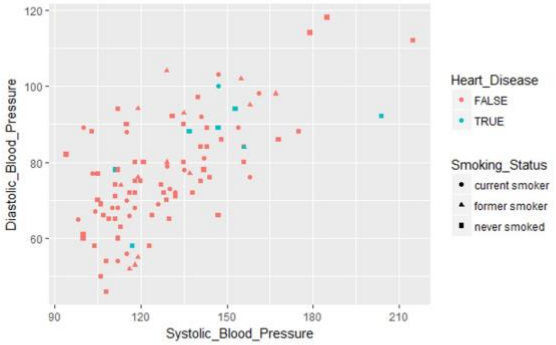
But in a two-dimensional bar graph of the same data, we see that it actually has a height of 14. Three-dimensional bar graphs are at best attractive and useless. And at worst, they can be actively misleading. So they don't belong in any data science application.

Notes:

Problems with the pie charts
https://www.data-to-viz.com/caveat/pie.html
https://bernardmarr.com/default.asp?contentID=1779.

3d data visualizations
https://www.data-to-viz.com/caveat/3d.html

Overly-complicated graphs



- Try displaying the relationship between your response and just 1 or 2 predictors at a time
- Try to summarize the key takeaway in 1-2 sentences

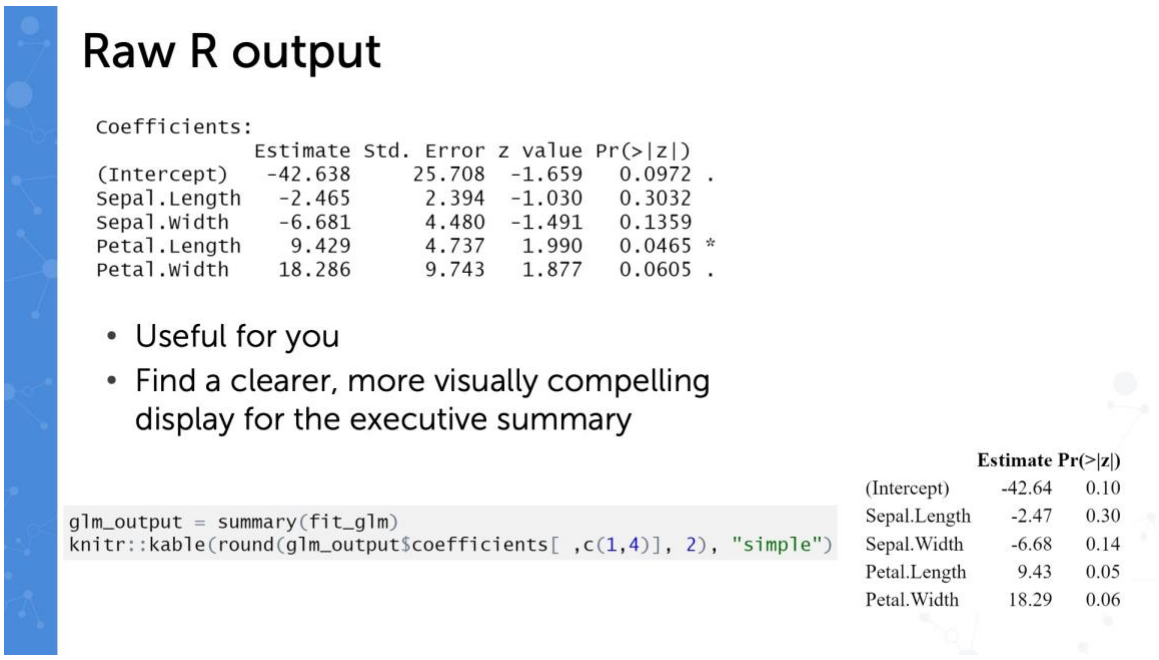
Falling in the realm of things that aren't so bad but still better to avoid are overly complicated graphs, such as this scatterplot showing the relationship between four different variables at the same time. I can see that there's generally a positive trend between diastolic and systolic blood pressure. And I can see that there aren't that many data points that have heart disease equal to true.

But I'm not really sure what the key takeaway of this graph is. If you find yourself looking at a graph like this, try displaying the relationship between your response variable and just one or two predictor variables at a time. And try to summarize the key takeaway of your graph in one to two sentences that can either go in the title of the graph, in the text of your executive summary, or both.

Notes:

```
library(simexaft)
data(BHS)
BHS <- BHS %>%
  mutate(Diastolic_Blood_Pressure = DBP) %>%
  mutate(Systolic_Blood_Pressure = SBP) %>%
  mutate(Heart_Disease = ifelse(CHID == 1, TRUE, FALSE))
%>%
  mutate(Smoking_Status = case_when(
    SMOKE2 == 1 ~ "current smoker",
    SMOKE1 == 1 ~ "former smoker",
    TRUE ~ "never smoked") )
```

```
gf_point(Systolic_Blood_Pressure ~
Diastolic_Blood_Pressure, col =~ Heart_Disease, shape =~
Smoking_Status, data = BHS)
```



Finally, one more thing that you should avoid in your executive summary is raw R output, such as this output from the summary of a linear regression model. This kind of output is really useful for you as you're building and interpreting your model. But it's pretty technical. So you should find a clearer, more visually compelling display for the non-technical executive summary. At a minimum, you could extract just the columns of output that you plan to discuss in the executive summary and then use the kable function to turn it into a table.

Notes:

```
iris2 <- iris %>%
  mutate(is_virginica = ifelse(Species == "virginica", 1,
0))

fit_glm = glm(is_virginica ~ .-Species, data = iris2,
family = "binomial")
summary(fit_glm)

glm_output = summary(fit_glm)
knitr::kable(round(glm_output$coefficients[ ,c(1,4)], 2),
"simple")
```