Statistical Modelling using R

Dr Ebin Deni Raj

29/10/2022

Contents

This document will introduce you to statistical models in R. You will learn to design, train and evaluate models in R.

Introdution

Statistical data is a kind of *summary* of data. The *summary* can be a way to encapsulate patterns in data. Some of these models are part of what's called "machine learning". The summary can be a way to translate from existing observations into conclusions. For example: Classyfying a sales prospect as strong or weak, or quality of biscuit as 'excellent' or 'poor'. Statistical models can be used for classifying events, untangling multiple influences(if you have lot of variables affecting a particular process). "Modeling" is a process of asking questions. "Statistical" refers in part to data – the statistical models you will construct will be rooted in data. But it refers also to a distinctively modern idea: that you can measure what you don't know and that doing so contributes to your understanding.The conclusions you reach from data depend on the specific questions you ask. Like it or not, the researcher plays an active and creative role in constructing and interrogating data. This means that the process involves some subjectivity. But this is not the same as saying anything goes. Statistical methods allow you to make objective statements about how the data answer your questions. In particular, the methods help you to know if the data show anything at all.The word "modeling" highlights that your goals, your beliefs, and your current state of knowledge all influence your analysis of data.

What all models have in common is this: **A model is a representation for a particular purpose.**

Representation -> it stands for something in the real world **Purpose**: -> YOUR particular use for the model.

Mathematical models: Constructed out of mathematical entities such as numbers,model formulas, equations etc..

Statistical Models A special tyoe of mathematical model informed by data and incorporates uncertainity and randomness Statistical models revolve around data. But even so, they are first and foremost models. They are created for a purpose. The intended use of a model should shape the appropriate form of the model and determines the sorts of data that can properly be used to build the model.

There are three main uses for statistical models. They are closely related, but distinct enough to be worth enumerating.

1. **Description**. Sometimes you want to describe the range or typical values of a quantity.For example, what's a "normal" white blood cell count? Sometimes you want to describe the relationship between things. Example: What's the relationship between the price of Oil and consumption by automobiles?
2. **Classification or prediction**. You often have information about some observable traits, qualities, or attributes of a system you observe and want to draw conclusions about other things that you can't directly observe. For instance, you know a patient's white blood-cell count and other laboratory measurements and want to diagnose the patient's illness.
3. **Anticipating the consequences of interventions**. Here, you intend to do something: you are not merely an observer but an active participant in the system. For example, people involved in setting or debating public policy have to deal with questions like these: To what extent will increasing the tax on petroleum products reduce consumption? To what extent will paying the team leader more increase the team's performance?

R and Statistical Modelling

A mathematical model

Recall that a mathematical model is a model framed in terms of mathematical stuff such as formulas. In contrast, a statistical model is a mathematical model based on data. Here let us use a toy mathematical model of outcomes on student tests. The model is implemented as a function, test_scores(), which takes the following inputs:

acad_motivation: a number from -3 to 3 indicating the level of academic motivation *Category_of admission*: a number from -3 to 3 indicating the level of category in which admission is sought *school*: either "public" or "private" to indicate whether the student attends public or private school

You'll start simply by using the test_scores() function to produce outputs for various levels of the inputs.

R objects for statistical Model

As already mentioned above, statistical model consist of dataframes, functions and formulas. Since you already know what is a dataframe, that will not be discussed here.
A function is a mathematical concept: the relationship between an output and one or more inputs. There are lot of packages and functions built for statistical model. Two commonly used functions are:
- rpart - lm

Some vocabulary will help to describe how to represent relationships with functions (formula). Any formula will consist of variables. The **response variable** is the variable whose behavior or variation you are trying to understand. On a graph, the response variable is conventionally plotted on the vertical axis.
The **explanatory variables** are the other variables that you want to use to explain the variation in the response.

Some examples of formula are shown below:
wage ~ exper + sector

The variable to the left of tilde operator is the Response variable, and the otehr variables are the explanatory variables. *The + sign is treated as a separator here(not the actual addition).*

wage ~ sector can be read as any of these:

- wage as a function of sector
- wage accounted for by sector
- wage modeled by sector

- wage explained by sector
- wage given sector
- wage broken down by sector

Run the following code in R to start with statistical modelling.

```
install.packages("mosaic")
install.packages("mosaicData")
install.packages("statisticalModeling_0.3.0.tar.gz", repos = NULL, type = "source")
```

Since the focus of this course is statistical modeling, I assume you already know how to get data into R. Many of the datasets will come from a package used for this topic: *statisticalModeling*

Formulas such as *wage ~ age + exper* are used to describe the form of relationships among variables. In this exercise, you are going to use formulas and functions to summarize data on the cost of life insurance policies. The data are in the AARP data frame, which has been preloaded from the statisticalModeling package.

The mosaic package augments simple statistical functions such as *mean(), sd(), median()*, etc. so that they can be used with formulas. For instance,*mosaic::mean(wage ~ sex, data = CPS85)* will calculate the mean wage for each sex. In contrast, the "built-in" *mean()* function (part of the base package) doesn't accept formulas, making it unnecessarily hard to do things like calculate groupwise means.

Note that we explicitly reference the *mean()* function from the mosaic package using double-colon notation (i.e. *package::function()*) to make it clear that we're not using the base R version of *mean()*.

Find the variable names in the *AARP data frame*, which should be first loaded in your workspace from the *statisticalModeling* package. *AARP* contains life insurance prices for the two sexes at different ages. Construct a formula using the variable names from the *AARP* data frame with an eye toward calculating the insurance *cost* broken down by *sex*. Use that formula as the first argument to *mosaic::mean()* to find the mean cost by sex.

```
library(statisticalModeling)
```

```
## Loading required package: ggplot2
```

```
library(mosaic)
```

```
## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2

##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.

##
## Attaching package: 'mosaic'

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following object is masked from 'package:Matrix':
##
##     mean

## The following object is masked from 'package:ggplot2':
##
##     stat
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, binom.test, cor, cor.test, cov, fivenum, median, prop.test,
##     quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```
library(mosaicData)
library(dplyr)
library(ggplot2)
# Find the variable names in AARP
data(AARP)
names(AARP)
```

```
## [1] "Age"      "Sex"      "Coverage" "Cost"
```

```
# Find the mean cost broken down by sex
mosaic::mean(Cost~Sex, data = AARP)
```

```
##        F        M
## 47.29778 57.53056
```

Formulas can be used to describe graphics in each of the three popular graphics systems: *base* graphics, *lattice* graphics, and in *ggplot2* with the statisticalModeling package. Most people choose to work in one of the graphics systems. I recommend *ggplot2* with the formula interface provided by statisticalModeling.

Make a boxplot of insurance cost broken down by sex in one of the three graphics systems. All can use the same syntax: *function(formula, data = AARP)*. In *base* graphics, the appropriate function is *boxplot()*. In *lattice* graphics, use *bwplot()* to make the boxplot. As always, you will have to load the lattice package. In the formula interface to *ggplot2*, use *gf_boxplot()*. The *statisticalModeling* package (which communicates with *ggplot2*), has been loaded for you. Make a scatterplot of *Cost* versus *Age* in one of the graphics packages. In base graphics: plot(). In lattice graphics: xyplot(). For ggplot graphics: gf_point().

Designing, Training and evaluating Models

The various choices in designing a model are as follows:

- A suitable training data set
- Specify response and explanatory variables
- Select a model architecture
  - Linear model (*lm*)
  - Recursive partitioning: *rpart()*

We have already discussed during the introductory sessions in data science about the process of modelling. Modelling is not a result, instead it is a process/cycle. Figure 1 clearly depicts the whole process.

Training a model is an automatic process cariied out by the program. Training is the way that the model is amde to match the patterns in your data. Sometimes we use the term 'fit the model'. You can think it as the program as a tailor who cuts the cloth to match the "shape" of the data. Metaphors apart, the point is to combine the modelling choices mentioned earlier in this section. A model will represent both the choices you made and the data.

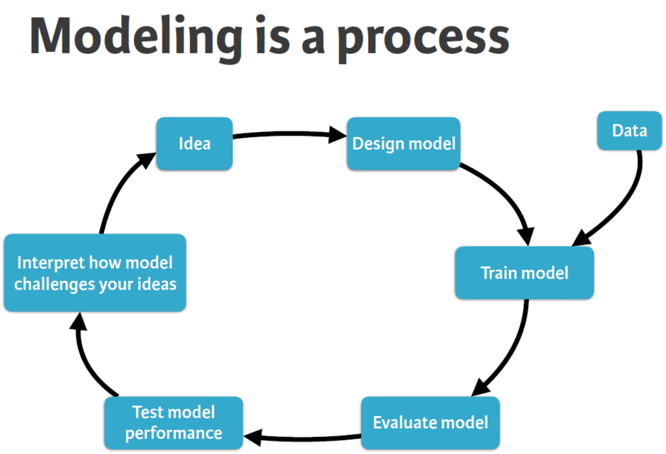


Figure 1: The process of Modelling

We will be using multiple datasets to demonstrate this process. Remember: In this course, we consider training as an automatic process: we provide the formula and the data, the training function does its job and produces a model for us. In later semester you will learn how to create a training function on your own.

Modelling Running times using linear model

In this exercise, you'll build three different models based on the data Runners, which is already in your workspace. The data come from records of the *Cherry Blossom Ten Mile Run*, a foot race held in Washington, DC each spring.

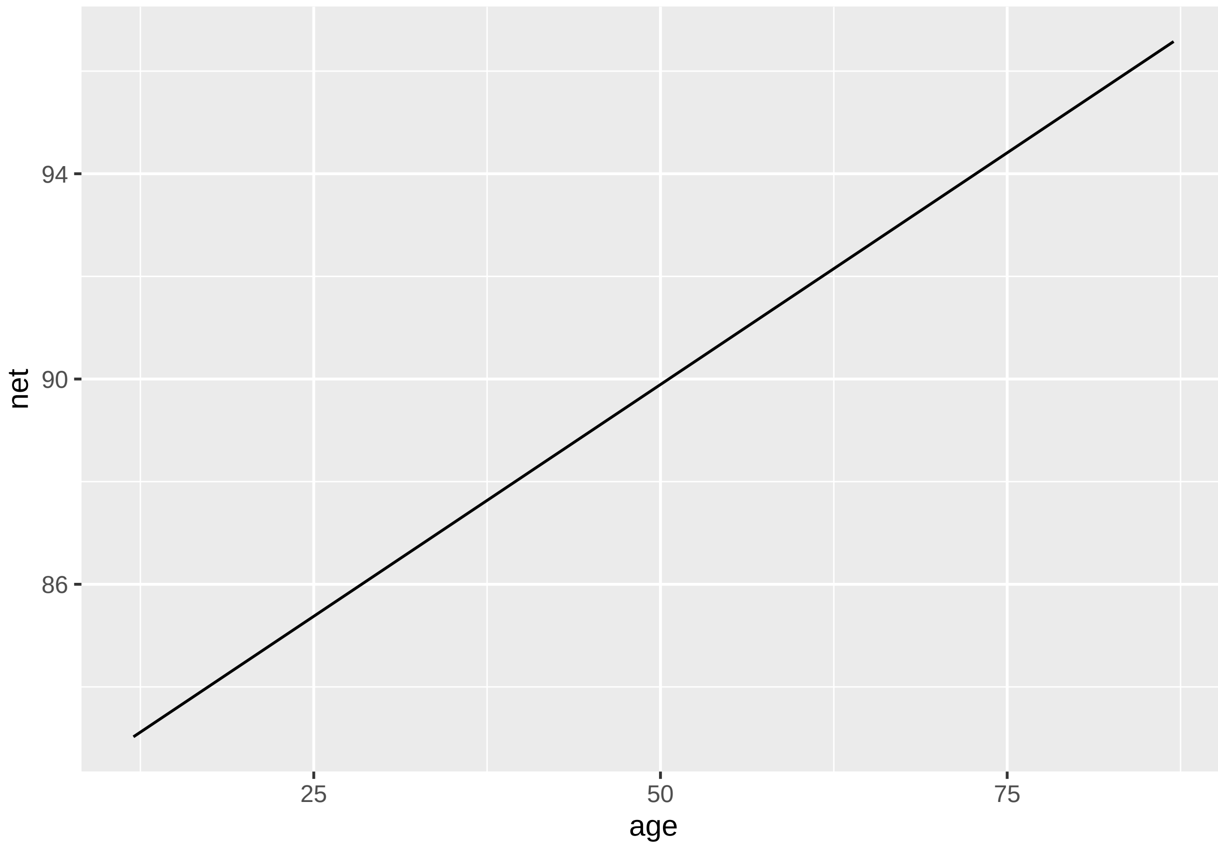
Imagine that you have been assigned the task of constructing a handicap for the runners, so that different runners can compare their times adjusting for their *age* and/or *sex*.

You will construct three different models using the linear model architecture. Each will have the "net" running time as a response and *age* and/or *sex* as explanatory variables. We will do the following task here: Find the names of the variables in Runners
Using the linear model architecture, create a model of net running time as a function of
age: handicap_model_1
sex: handicap_model_2
both age and sex: handicap_model_3
You can look at a graph of each model using the *fmodel()* commands provided for this purpose.

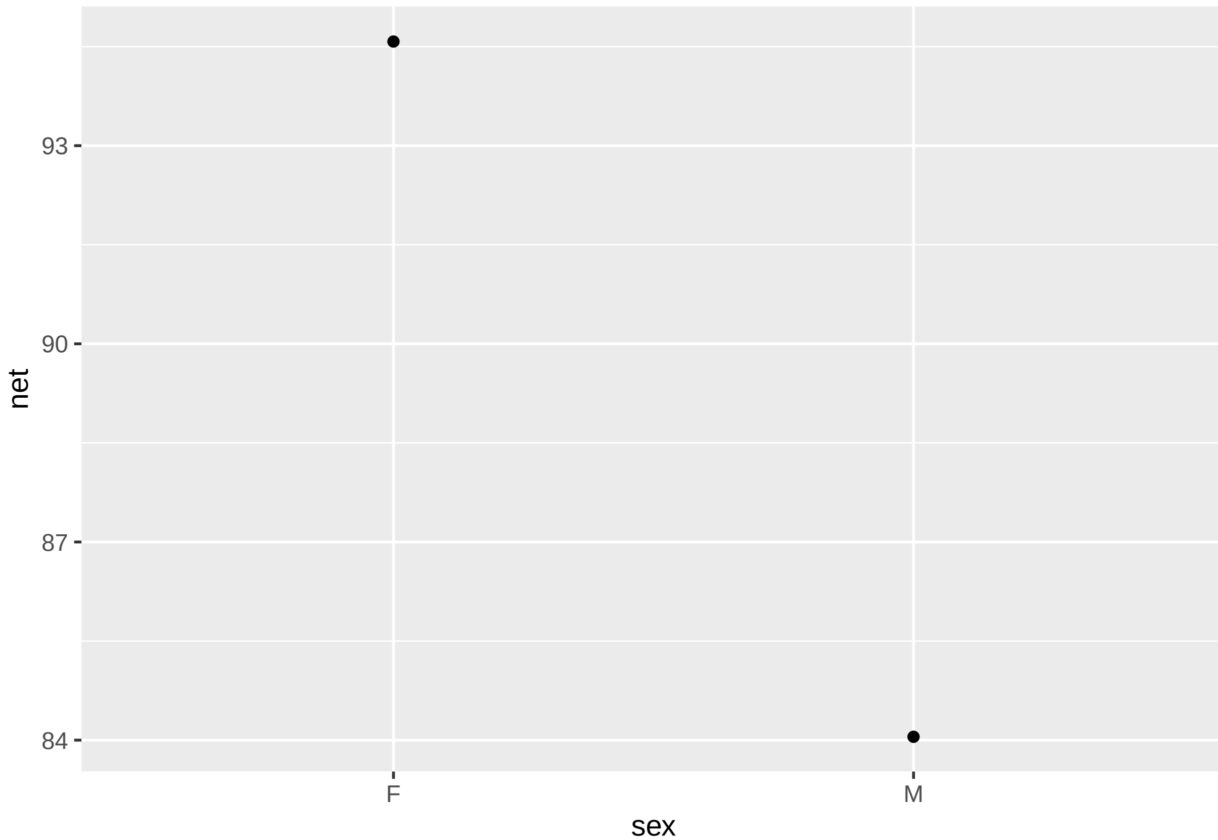
```
# Find the variable names in Runners
data(Runners)
names(Runners)
```

```
## [1] "age"            "net"            "gun"            "sex"
## [5] "year"           "previous"       "nruns"          "start_position"
```

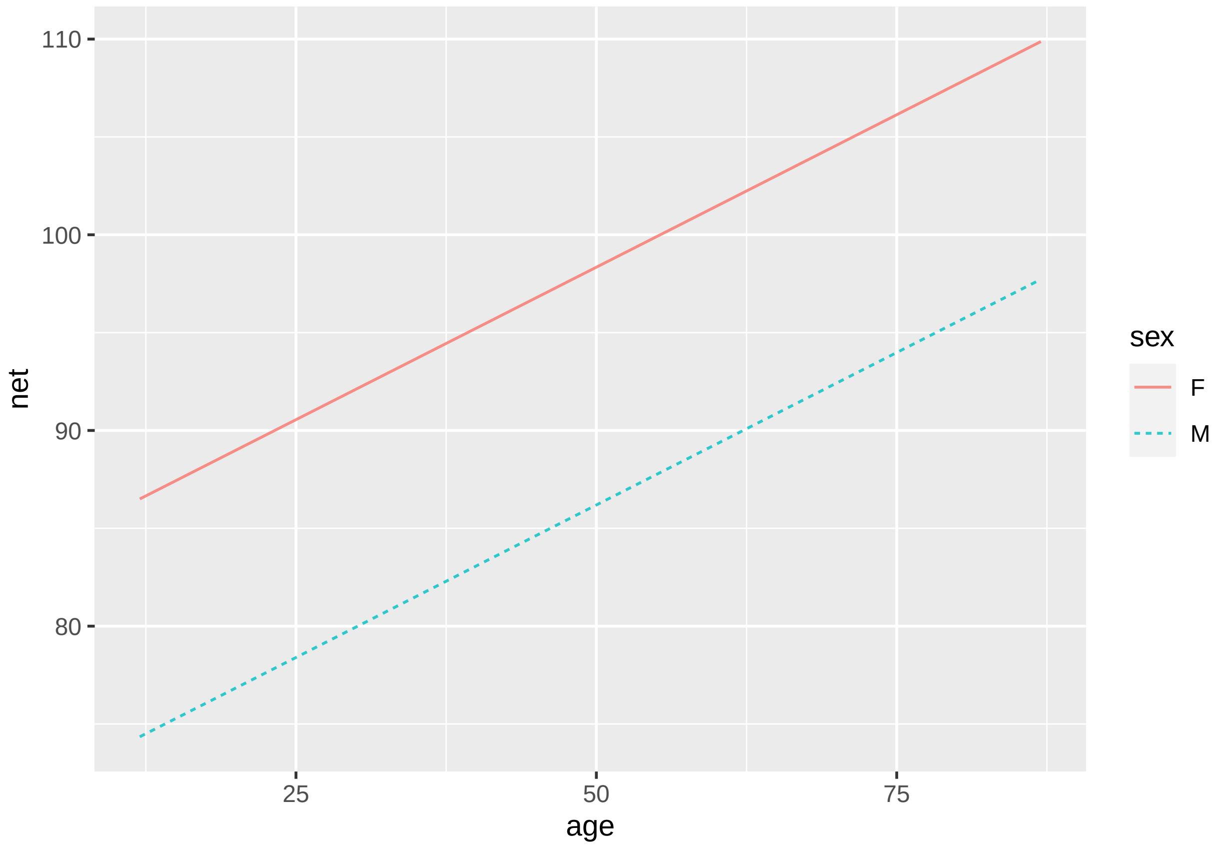
```
# Build models: handicap_model_1, handicap_model_2, handicap_model_3
handicap_model_1 <- lm(net~age, data = Runners)
handicap_model_2 <- lm(net~sex,data=Runners)
handicap_model_3 <- lm(net~age+sex,data=Runners)

# For now, here's a way to visualize the models
fmodel(handicap_model_1)
```




```
fmodel(handicap_model_2)
```




```
fmodel(handicap_model_3)
```



Using the recursive partitioning model architecture

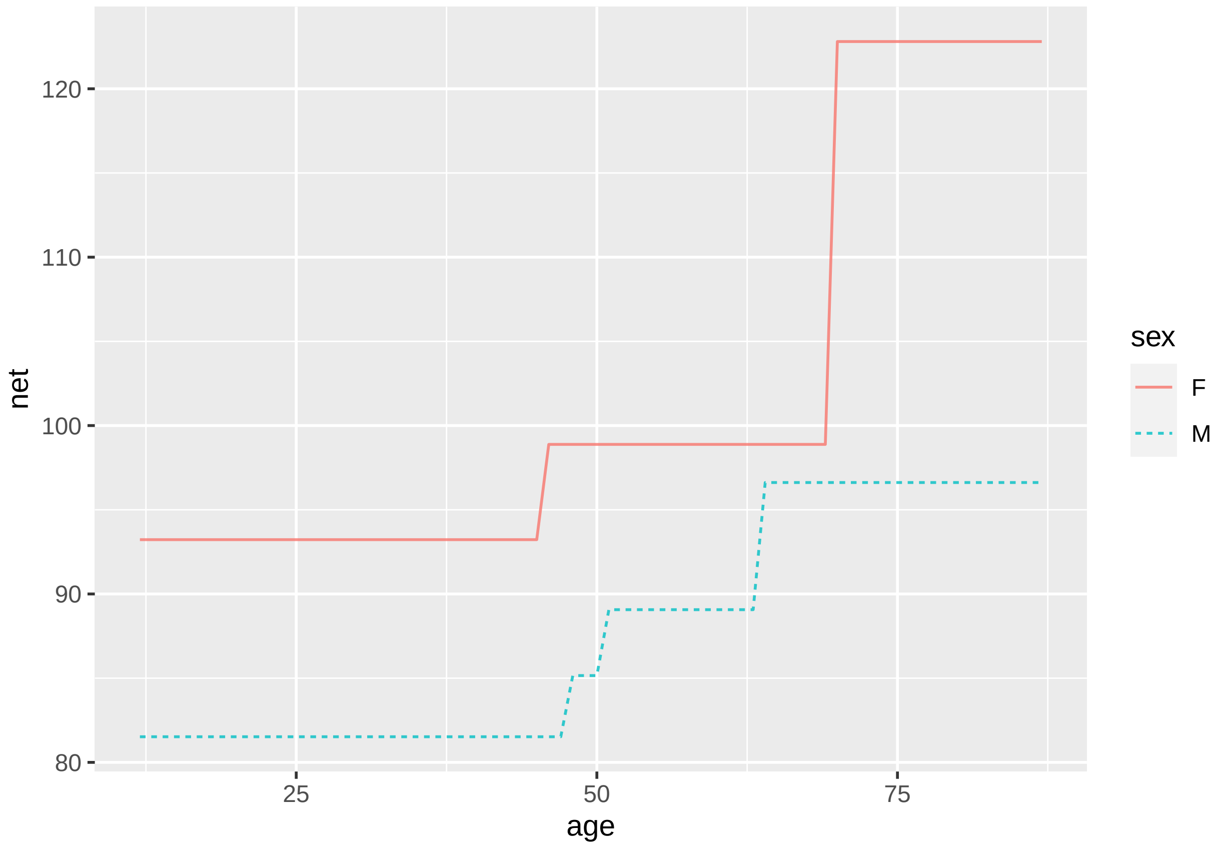
In the previous exercise, you used the linear modeling architecture to construct a model of a runner's time as a function of age and sex. There are many different model architectures available. In this exercise, you'll build models using the recursive partitioning architecture and the same Runners data frame as in the previous question. The model-building function to use is rpart(), which is analogous to lm() for linear models.

The recursive partitioning architecture has a parameter, *cp*, that allows you to dial up or down the complexity of the model being built. Without worrying about the details just yet, you can set this parameter as a named argument to *rpart()*

The task are as follows:

Load the rpart package
Using the *rpart()* function (from the rpart package) construct a model of net running time versus *age* and *sex*. Set the complexity parameter *cp = 0.002*. Store the result as *model_2*.
Examine the model using *fmodel*

```
# Load rpart

library(rpart)
# Build rpart model: model_2
model_2<-rpart(net~age+sex,data=Runners,cp=0.002)

# Examine graph of model_2
fmodel(model_2, ~age + sex)
```



Will they run again?

In the previous two exercises, you built models of the net running time. The purpose for these models was imagined to be the construction of a handicapping system for comparing runners of different sexes and ages. By giving as inputs to the model the *age* and *sex* of a runner, the model produces an "expected running time." This becomes the handicap.

Now, let's imagine another possible purpose for a model: to predict whether or not a person who participated in the race this year will participate next year. For simplicity, a data frame *Ran_twice* has been created in your workspace. *Ran_twice* extracts all the people who had run the race two times and provides a variable, runs_again, that indicates if the person participated the next year (that is, in year three).

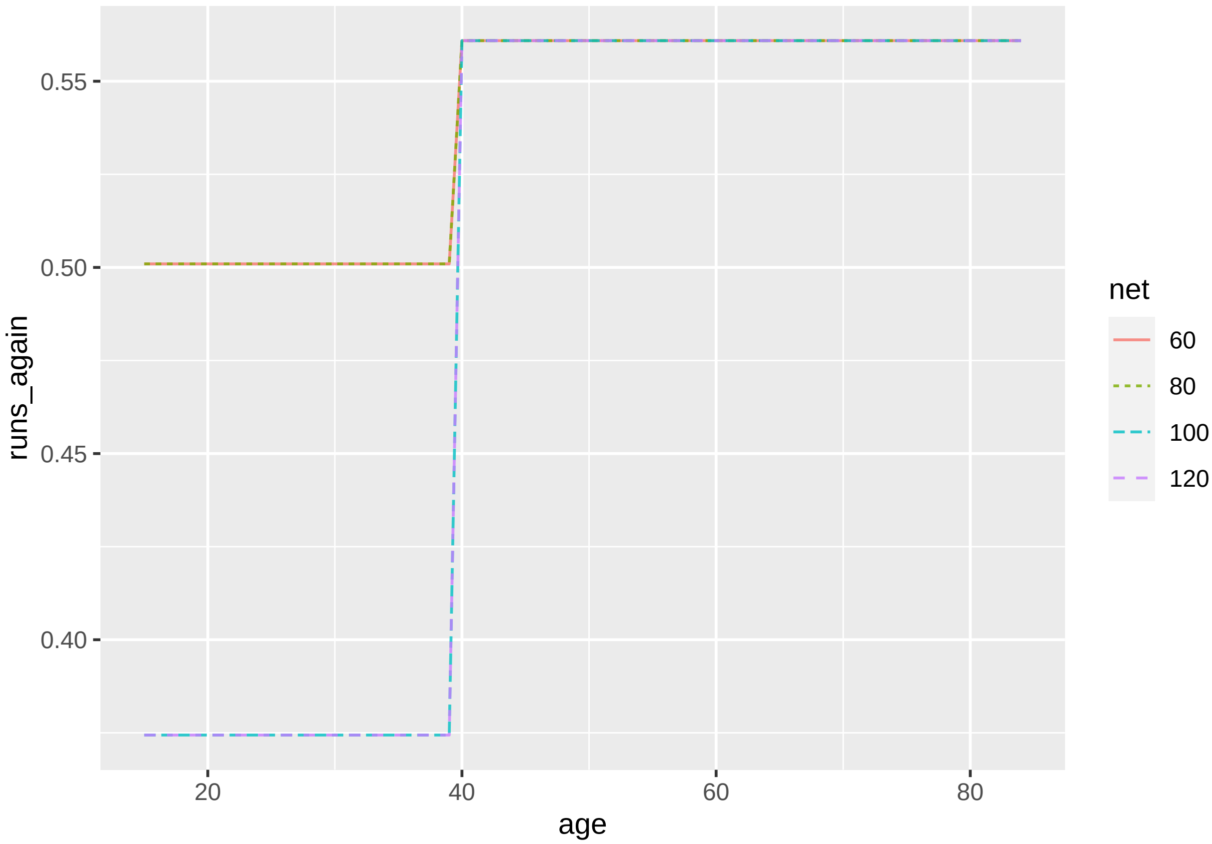
Predicting whether or not a person will run again next year is a very different purpose than finding a typical running time. The model to achieve that new purpose can be very different than in the previous two exercises. In particular:

The output of this model will be *TRUE* or *FALSE*, indicating whether the person will participate next year. That is, the response variable will be *runs_again*. You can use variables like *net* running time as explanatory variables. The response variable *runs_again* is categorical, not numeric. Since *lm()* is intended for quantitative responses, you'll use only the *rpart()* architecture, which works for both numerical and categorical responses. The task to be done, is as follows:

Using rpart() and data = Ran_twice, build a model with response runs_again and explanatory variables age, sex, and net running time. Set the complexity parameter to cp = 0.005. Store the model as run_again_model. Visualize the model as a graph. The y-axis drawn will mark the probability that the outcome is TRUE.

```
# Create run_again_model
Ran_twice<-read.csv("Ran_twice.csv")
```

```r
run_again_model<-rpart(runs_again~age+sex+net,data=Ran_twice,cp=0.005)

# Visualize the model
fmodel(run_again_model, ~age + net, data = Ran_twice)
```



Evaluating Models

The most inmportant thing that decides the future of a model(whether to keep it or remove it!!!) is the evaluation of the model. i.e We will provide new inputs to the modeland calculate the resulting output. Here we will use evaluation in a more explanatory way , or just to make predictions. The trained model does contain all the information needed to construct the training function, and the *predict()* function does the work to put that information together and produce the outputs for the given inputs.

After creating many models, how do you decide which one is a better model. You can compare the predicted value with the actual value. This is known as prediction error. The lesser the prediction error, better is the model. Here , you need to create some inputs to help us understand the relation between *cost* and *age*.

For the following task you will work with the dataset *AARP*

The AARP data frame on the cost of life insurance is loaded in your workspace. Display the names of the variables in AARP. Build a model called insurance_cost_model, using the linear architecture, of insurance cost as a function of age, sex, and coverage.
Construct a data frame, called example_vals which is a data.frame() with three variables: Age, Sex, and Coverage set equal to 60, "F", and 200, respectively. This corresponds to a 60-year old female buying $200,000 of life insurance.
Use predict() to calculate the model cost of insurance. Use the values in example_vals as your input.

Load the statisticalModeling package.
Use evaluate_model() to calculate the model output for the inputs in example_vals.

```
# Display the variable names in the AARP data frame
names(AARP)
```

```
## [1] "Age"      "Sex"      "Coverage" "Cost"
```

```
# Build a model: insurance_cost_model
insurance_cost_model<-lm(Cost~Age+Sex+Coverage,data=AARP)

# Construct a data frame: example_vals
example_vals<-data.frame(Age=60,Sex="F",Coverage=200)

# Predict insurance cost using predict()
predict(insurance_cost_model,example_vals)
```

```
##       1
## 363.637
```

```
# Load statisticalModeling
library(statisticalModeling)

# Calculate model output using evaluate_model() : This is an alternative to predict()
evaluate_model(insurance_cost_model,example_vals)
```

```
##   Age Sex Coverage model_output
## 1  60   F      200      363.637
```

Extrapolation

One purpose for evaluating a model is extrapolation: finding the model output for inputs that are outside the range of the data used to train the model.

Extrapolation makes sense only for quantitative explanatory variables. For example, given a variable x that ranges from 50 to 100, any value greater than 100 or smaller than 50 is an extrapolation.

In this exercise, you'll extrapolate the AARP insurance cost model to examine what the model suggests about insurance costs for 30-year-olds and 90-year-olds. Keep in mind that the model outputs might not make sense. Models trained on data can be a bit wild when evaluated outside the range of the data.

The tasks we are going to do is as:
Build a linear-architecture model of Cost as a function of *Age, Sex, and Coverage*. Call the model *insurance_cost_model*. Create a data frame with variables *Age* set to c(30, 90), *Sex* set to c("F", "M"), and *Coverage* set to c(0, 100). Call this new_inputs_1. Call the *expand.grid()* function with the same arguments you passed to data.frame() to create *new_inputs_2*. Try to figure out what expand.grid() does by experimenting, or *type ?expand.grid* in the console to read the documentation. Use *predict()* to find the output of *insurance_cost_model* given *new_inputs_1* and *new_inputs_2* as inputs. Do the same with *evaluate_model()*.

```
# Build a model: insurance_cost_model
insurance_cost_model <- lm(Cost~Age+Sex+Coverage, data = AARP)

# Create a data frame: new_inputs_1
new_inputs_1 <- data.frame(Age =c(30,90), Sex = c("F","M"),
                           Coverage = c(0,100))

# Use expand.grid(): new_inputs_2
```

```r
new_inputs_2 <- expand.grid(Age =c(30,90), Sex = c("F","M"),Coverage = c(0,100))

# Use predict() for new_inputs_1 and new_inputs_2
predict(insurance_cost_model, newdata = new_inputs_1)
```

```
##         1         2
## -99.98726 292.88435
```

```r
predict(insurance_cost_model, newdata = new_inputs_2)
```

```
##         1         2         3         4         5         6         7         8
## -99.98726 101.11503 -89.75448 111.34781  81.54928 282.65157  91.78206 292.88435
```

```r
# Use evaluate_model() for new_inputs_1 and new_inputs_2
evaluate_model(insurance_cost_model, data = new_inputs_1)
```

```
##   Age Sex Coverage model_output
## 1  30   F        0    -99.98726
## 2  90   M      100    292.88435
```

```r
evaluate_model(insurance_cost_model, data = new_inputs_2)
```

```
##   Age Sex Coverage model_output
## 1  30   F        0    -99.98726
## 2  90   F        0    101.11503
## 3  30   M        0    -89.75448
## 4  90   M        0    111.34781
## 5  30   F      100     81.54928
## 6  90   F      100    282.65157
## 7  30   M      100     91.78206
## 8  90   M      100    292.88435
```

Typical values of Data

Sometimes you want to make a very quick check of what the model output looks like for "typical" inputs. When you use evaluate_model() without the data argument, the function will use the data on which the model was trained to select some typical levels of the inputs. evaluate_model() provides a tabular display of inputs and outputs.

Many people prefer a graphical display. The fmodel() function works in the same way as evaluate_model(), but displays the model graphically. When models have more than one input variable (the usual case) choices need to be made about which variable to display in what role in the graphic. For instance, if there is a quantitative input, it's natural to put that on the x-axis. Additional explanatory variables can be displayed as color or as facets (i.e. small subgraphs). You do not need to display all of the explanatory variables in the graph.

The syntax for *fmodel()* is

fmodel(model_object, ~ x_var + color_var + facet_var) where, of course, you'll use the name of the variable you want on the x-axis instead of *x_var* and similarly for *color_var* and *facet_var* (which are optional). Note that only the right-hand side of the ~ is used in the formula.

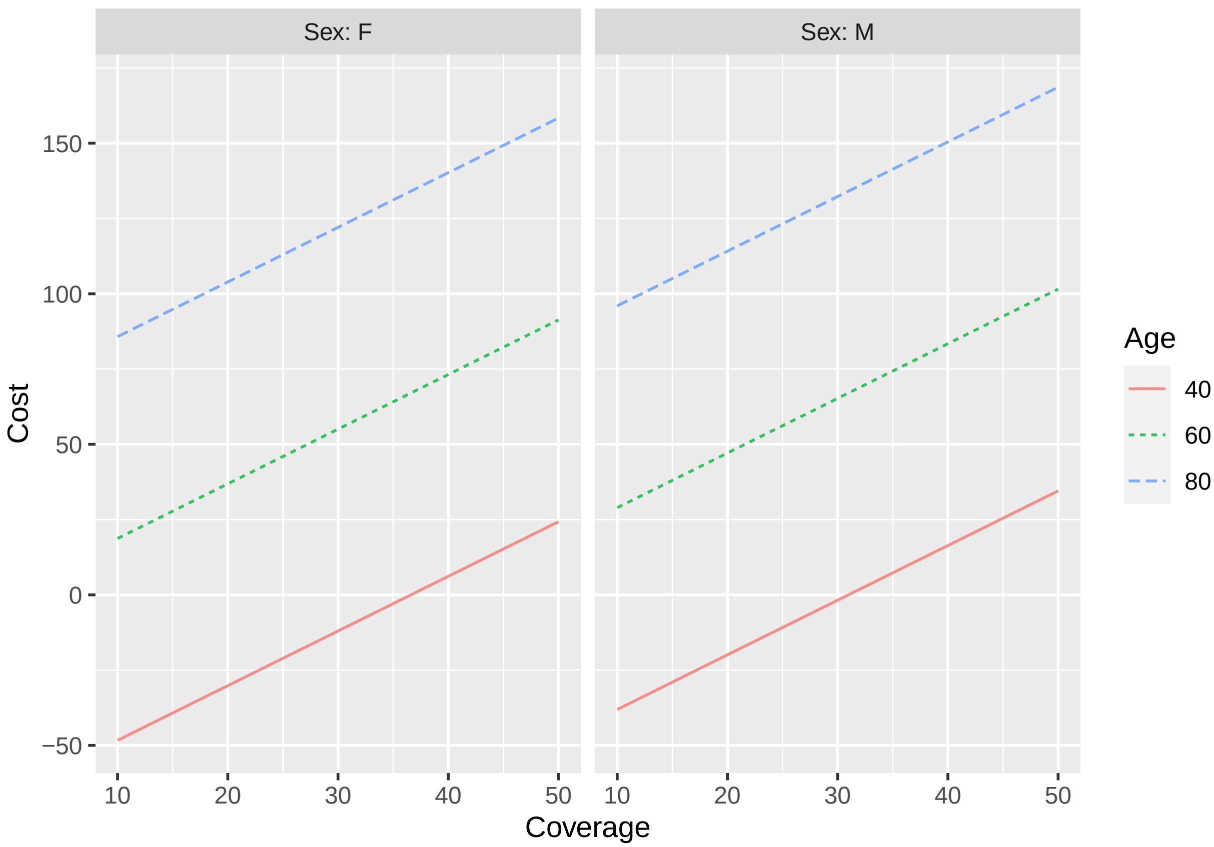
AARP and insurance_cost_model are now available in your workspace(from previous tasks).

Use *evaluate_model()* to calculate the model output for typical values of the explanatory variables (i.e. do not specify the data argument). Construct an appropriate formula to use with *fmodel()* to reproduce the graphic shown in the display.

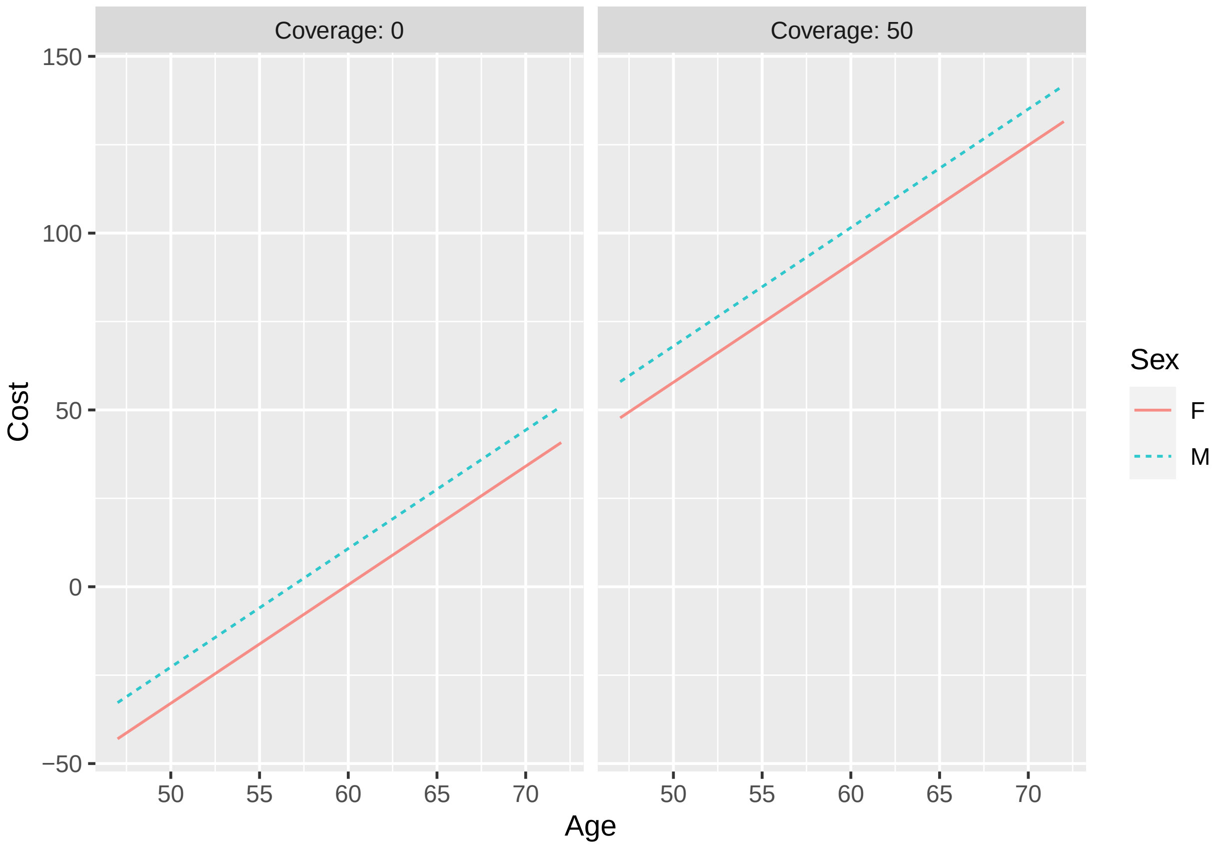
```r
# Evaluate insurance_cost_model
evaluate_model(insurance_cost_model)
```

```
##    Age Sex Coverage model_output
## 1   40   F        0  -66.4702087
## 2   60   F        0    0.5638866
## 3   80   F        0   67.5979818
## 4   40   M        0  -56.2374309
## 5   60   M        0   10.7966643
## 6   80   M        0   77.8307596
## 7   40   F       50   24.2980606
## 8   60   F       50   91.3321558
## 9   80   F       50  158.3662510
## 10  40   M       50   34.5308383
## 11  60   M       50  101.5649336
## 12  80   M       50  168.5990288
```

```r
# Use fmodel() to reproduce the graphic
fmodel(insurance_cost_model, ~ Coverage + Age + Sex)
```




```r
# A new formula to highlight difference in sexes
new_formula <- ~Age + Sex + Coverage

# Make the new plot
fmodel(insurance_cost_model, new_formula)
```



The plot shown is nice enough, but it doesn't serve all purposes. For instance, it's hard to see the differences in insurance costs between the sexes. Change the value of new_formula to try different arrangements of the terms in the formula until you get a plot that more clearly displays the difference between sexes.

Assessing Prediction Performance

We will look more into performance analysis of models. The design choices (namely data, response variable, explanatory variable and model architecture) in buliding a model are completely dependednt on the purpose for which your buliding the model. The way in which we can apply statistical model on a set of data is as follows:

Make predictions about an outcome Run experiments to study relationships between variables *Explore data to identify relationships among variables

You might have a confusion, which model to apply when:

Well,
Basic choices in model architecture:
Categorical response variable (e.g. yes or no, infected or not)
Use rpart() Numerical response variable (e.g. unemployment rate)
Use *lm()* or *rpart()*

Linear model(*lm*) is the best choice for relationships that are gradual and proportional while *rpart* is helpful for dicotonous, discontinouus replationships.

When you train and test a model, you use data with values for the explanatory variables as well as the response variable. Training effectively creates a function that will take as inputs values for the explanatory variables and produce as output values corresponding to the response variable.

If the model is good, when provided with the inputs from the testing data, the outputs from the function will give results “close” to the response variable in the testing data. How to measure “close”? The first step is to subtract the function output from the actual response values in the testing data. The result is called the prediction error and there will be one such error for every case in the testing data. You then summarize that set of prediction errors. The best way to summarize the set of prediction error is the eman square of the prediction errors.

Choosing explanatory variables

The general startegy for deciding whether a potential explanatory variable is useful for prediction is to compare the predictive performance of two models , one with and one without the variable being considered. Let’s return to the data on runners in the Cherry Blossom Ten Mile Race. Suppose that we’ve built a linear model of net running time using the obvious explanatory variables: age and sex. For this exercise, you’ll use a small set of data with only 100 runners: Runners_100. You can find the Runners_100.csv file in your directory(zip file)

Now you want to find out if you can use use a runner’s previous experience to improve the model predictions, so you’ll build a second model that includes previous as an explanatory variable in addition to age and sex. When evaluated on the training data, each of the two models will produce an output for every case in the training data.

In this exercise, you’ll compare predictions from the two models. The tasks will be: Build a base model, using the linear model architecture, of net running time as a function of age and sex. The training data is the Runners_100 data frame. Evaluate the base model on the inputs from the training data. Build an augmented model, adding previous as an explanatory variable. Evaluate the augmented model on the training data. Find the mean square difference between the output for the base model and the output for the augmented model. This tells us how much outputs from the two models differ, typically.

```r
# Build a model of net running time
Runners_100<-read.csv("Runners100.csv")
base_model <- lm(net ~ age + sex, data = Runners_100)

# Evaluate base_model on the training data
base_model_output <- predict(base_model, newdata = Runners_100)

# Build the augmented model
aug_model <- lm(net ~ age + sex + previous, data = Runners_100)

# Evaluate aug_model on the training data
aug_model_output <- predict(aug_model, newdata = Runners_100)

# How much do the model outputs differ?
mean((base_model_output - aug_model_output) ^ 2, na.rm = TRUE)
```

```
## [1] 0.5157921
```

Adding previous as an explanatory variable changes the model outputs. In the next exercise, you’ll start to examine whether the augmented model gives better predictions. By the way, in this and the next exercise you’ll use a sample of just 100 runners because later you’re going to have to deal with the statistical question of whether the data you have is sufficient to make any claim at all.

Prediction performance

In the previous exercise, you built two models of net running time: net ~ age + sex and net ~ age + sex + previous. The models were trained on the Runners_100 data. The two models made somewhat different predictions: the standard deviation of the difference was about 1 minute (as compared to a mean net running time of about 90 minutes).

Knowing that the models make different predictions doesn't tell you which model is better. In this exercise, you'll compare the models' predictions to the actual values of the response variable. The term prediction error or just error is often used, rather than difference. So, rather than speaking of the mean square difference, we'll say mean square error.

The tasks will be : Build and evaluate the base model net ~ age + sex on the Runners_100 training data. Do the same for the augmented model: net ~ age + sex + previous. For each model, take the difference between the response variable in the training data and the model output when given the training data as input. This will give you two sets of case-by-case differences, one for each model. Calculate the mean square error for each model. The smaller the mean square error, the closer the model outputs are to the actual response variable.

```
# Build and evaluate the base model on Runners_100
base_model <- lm(net ~ age + sex, data = Runners_100)
base_model_output <- predict(base_model, newdata = Runners_100)

# Build and evaluate the augmented model on Runners_100
aug_model <- lm(net ~ age+sex+previous, data=Runners_100)
aug_model_output <- predict(aug_model, data=Runners_100)

# Find the case-by-case differences
base_model_differences <- with(Runners_100, net - base_model_output)
aug_model_differences <- with(Runners_100,net-aug_model_output)

# Calculate mean square errors
mean(base_model_differences ^ 2)
```

```
## [1] 131.5594
```

```
mean(aug_model_differences ^ 2)
```

```
## [1] 131.0436
```

The augmented model gives slightly better predictions. But, as you'll see in the next exercise, comparing models using the training data gives an unfair advantage to the augmented model. This problem can be addressed with cross validation(which you will learn in a while)

The statistics part

You've seen only part of the technique for using mean square error (MSE) to decide whether to include an explanatory variable in a linear model architecture. The technique isn't yet complete because of a problem: Whenever you use it you will find that the model with the additional explanatory variable has smaller prediction errors than the base model! The technique always gives the same indication: include the additional explanatory variable. You'll start to fix this problem so that the technique of comparing MSE becomes useful and meaningful in practice.

This exercise gives another example of the problem at work. To start, build a model to serve as the base. You'll use *wage* from the *CPS85* dataset as the response variable, and any of the other variables as the explanatory variables. Then you'll build a second model that adds another explanatory variable to those in the base model. You'll see that the MSE is smaller in the expanded model than in the base model.

Of course, it might be that the added variable genuinely contributes to the quality of predictions. To make sure that the variable added to the second model is not in fact genuinely capable of improving predictions, you'll construct that variable to be complete random junk with no explanatory power whatsoever.

The goal is to:
Create a new variable in CPS85 named bogus that contains random TRUE and FALSE values. Use rnorm(nrow(CPS85)) > 0 to generate this column. Construct a linear model called base_model using wage as the response variable and educ, sector and sex (NOT bogus!) as explanatory variables. Construct a second model called aug_model that has all the same explanatory variables as base_model but also includes bogus. Find the standard deviation of prediction errors from the base model. Find the standard deviation of prediction errors from the augmented model

```
# Add bogus column to CPS85 (don't change)
CPS85$bogus <- rnorm(nrow(CPS85)) > 0

# Make the base model
base_model <- lm(wage ~ educ+sector+sex, data = CPS85)

# Make the bogus augmented model
aug_model <- lm(wage ~ educ+sector+sex+ bogus, data = CPS85)

# Find the MSE of the base model
mean((CPS85$wage - predict(base_model, newdata = CPS85)) ^ 2)
```

```
## [1] 19.73308
```

```
# Find the MSE of the augmented model
mean((CPS85$wage - predict(aug_model, newdata = CPS85)) ^ 2)
```

```
## [1] 19.73297
```

Even though the bogus variable has no genuine explanatory power, the MSE is smaller when bogus is included as an explanatory variable.

Testing and training datasets

In the last excercise to test the model performance we had introduced some new data. Instead of that here we will use a technique called cross validation. The dataset will be divided into two non-overalapping sets. Some of the cases will be put into training set and the remaining will be put into testing set. Once we have the training and testing dataset we can construct the two models. The training data as the name suggests will be used to train the model and to asses the performance we will use the testing data. The testing data is "new" in that the models were trained without seeing it. But since the testing data contains values for the response variable, we can compare the models predictions to the actual gresponse. We use the test data explanatory variables as inputs to the model. The model generates, for each case an output value. Comparing the output to the test data response variable gives us the model error.

In a nutshell the process goes like this:
Use the trained models to calculate the output when given the inputs from the testing data. Then subtract these model outputs from the known, actual response variable in the testing data. Some of the errors might be positive: which means the model under-estimated the response. Some of the errors might be negative: which means the model over estimated the reponse. The usual practice is to square the error, so that all are positive numbers.

To reduce the set of square errors to a single "Figure of merit" for the model, its common to calculate the sum of square errors or alternatively the mean square error.

In this exercise, you'll see one way to split your data into non-overlapping training and testing groups. Of course, the split will be done at random so that the testing and training data are similar in a statistical sense.

The code in the editor uses a style that will give you two prediction error results: one for the training cases and one for the testing cases. Your goal is to see whether there is a systematic difference between prediction accuracy on the training and on the testing cases.

Since the split is being done at random, the results will vary somewhat each time you do the calculation. As you'll see in later exercises, you deal with this randomness by rerunning the calculation many times.

The tasks are: Examine the code that adds a column named training_cases to Runners_100 consisting of random TRUEs and FALSEs. The TRUEs will be the training cases and the FALSEs will be the testing cases. Build the linear model net ~ age + sex on the training data. The code for subsetting Runners_100 is provided for you. Use evaluate_model() to find the model predictions on the testing data. You can use a similar call to subset() on Runners_100, replacing training_cases with !training_cases (i.e. "not training cases"). Calculate the mean square error (MSE) on the testing data. This will be the mean of (net - model_output)^2.

```
# Generate a random TRUE or FALSE for each case in Runners_100

Runners_100$training_cases <- rnorm(nrow(Runners_100)) > 0

# Build base model net ~ age + sex with training cases
base_model <- lm(net~age+sex, data = subset(Runners_100, training_cases))

# Evaluate the model for the testing cases
Preds <- evaluate_model(base_model, data = subset(Runners_100,!training_cases))

# Calculate the MSE on the testing data
with(data = Preds, mean((net - Preds$model_output)^2))
```

```
## [1] 142.204
```

As a general rule, estimates of prediction error based on the training data will be smaller than those based on the testing data. Still, because the division into training and testing sets is done at random, it will happen from time to time that the opposite appears. Now you should be able to explain why we had you work with Runners_100 rather than the full dataset Runners. The worsening in prediction error between the training and testing phases is more evident in small datasets than in large ones.

Repeating random trials

In the previous exercise, you implemented a cross validation trial. We call it a trial because it involves random assignment of cases to the training and testing sets. The result of the calculation was therefore (somewhat) random.

Since the result of cross validation varies from trial to trial, it's helpful to run many trials so that you can see how much variation there is. As you'll see, this will be a common process as you move through the course.

To simplify things, the cv_pred_error() function in the statisticalModeling package will carry out this repetitive process for you. All you need do is provide one or more models as input to cv_pred_error(); the function will do all the work of creating training and testing sets for each trial and calculating the mean square error for each trial. Easy!

The context for this exercise is to see whether the prediction error calculated from the training data is consistently different from the cross-validated prediction error. To that end, you'll calculate the in-sample error using only the training data. Then, you'll do the cross validation and use a t-test to see if the in-sample error is statistically different from the cross-validated error.

```
# The model
model <- lm(net ~ age + sex, data = Runners_100)

# Find the in-sample error (using the training data)
in_sample <- evaluate_model(model, data = Runners_100)
in_sample_error <-
```

```r
  with(in_sample, mean((net - model_output)^2, na.rm = TRUE))

# Calculate MSE for many different trials
trials <- cv_pred_error(model)

# View the cross-validated prediction errors
trials
```

```
##        mse model
## 1 146.1618 model
## 2 141.8224 model
## 3 138.7510 model
## 4 139.7383 model
## 5 146.0613 model
```

```r
# Find confidence interval on trials and compare to training_error
mosaic::t.test(~ mse, mu = in_sample_error, data = trials)
```

```
##
##  One Sample t-test
##
## data:  mse
## t = 7.0496, df = 4, p-value = 0.002135
## alternative hypothesis: true mean is not equal to 131.5594
## 95 percent confidence interval:
##  138.1953 146.8187
## sample estimates:
## mean of x
##   142.507
```

The error based on the training data is below the 95% confidence interval representing the cross-validated prediction error.

To add or not to add (an explanatory variable)?

In this exercise, you're going to use cross validation to find out whether adding a new explanatory variable improves the prediction performance of a model. Remember that models are biased to perform well on the training data. Cross validation gives a fair indication of the prediction error on new data.

The goal is to: Train an augmented model that includes previous along with the other explanatory variables. Run cross validation trials on the two models using cv_pred_error(). Give the trained models as arguments and store the result in an object called trials. Perform a two-sample t-test to compare the cross validation trials for the two models. The general syntax for a two-sample t-test is t.test(quant ~ group, data = _____). You'll have to figure out what are the names corresponding to quant and group in the output of cv_pred_error(). Print trials to the console to figure out which of its variables correspond to quant and group.

```r
# The base model
base_model <- lm(net ~ age + sex, data = Runners_100)

# An augmented model adding previous as an explanatory variable
aug_model <- lm(net ~ age + sex + previous, data = Runners_100)

# Run cross validation trials on the two models
trials <- cv_pred_error(base_model, aug_model)
```

```r
# Compare the two sets of cross-validated errors
t.test(mse ~ model, data = trials)
```

```
##
##  Welch Two Sample t-test
##
## data:  mse by model
## t = 1.2289, df = 4.7194, p-value = 0.2768
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.940448  8.146125
## sample estimates:
##  mean in group aug_model mean in group base_model
##                 143.7979                 141.1951
```

Notice that cross validation reveals that the augmented model makes worse predictions (larger prediction error) than the base model. **Bigger is not necessarily better when it comes to modeling!**

##Exploring Data with models The techniques which we have seen in the last section are appropriate for numeric response variables. It will be helpful only to predict error for numeric/quantitative variable. In such models, since the output is numeric, its possible to calculate the difference between the model's predictiona nd the actual observed value. How to measure the error, if the variables are categorical?? One possible option is to count and compare the True/False values. One better option is to calculate the mean rather that the sume of Trues/False that result from comparison. In some cases,Categorical output can be arranged in such a way to return a numerical value for each possible level of the response. This number is the probability, according to the model,of the outcome being that particular level. For each case , we can extract the probability that the model assigned to the observed outcome. This probavility is called "likelihood" of that case. Then multiply the likelihoods together to get the total likelihood of the observations. You can even take the log of sum of the likelihood(ofcourse mathematically they will be equivalent)

The maximum error rate

The 10,000 runners in Runners can't all start at the same time. They line up behind the start (the line-up goes for about half a mile). There is a handful of elite runners who are given spots right at the start line, but everyone else gets in line.

The start_position variable categorizes the enthusiasm of the runners based on how close they maneuvered to the start line before the gun. The variable is categorical, with levels "calm", "eager", and "mellow". The context for this exercise is whether other variables in Runners can account for start_position. Since the response variable start_position is categorical, rpart() is an appropriate architecture.

In this exercise, you'll investigate the prediction performance of what is sometimes called the null model. This is a model with no explanatory variables, the equivalent to "I don't know what might explain that." The output of the null model will be the same for every input.

You might think that random guessing of the output would be just about the same as the output of the null model. So you'll also look at the prediction performance of random guessing.

The tasks are as shown below:

Construct the null model with start_position as the response variable.
Evaluate that model to get the outputs for each case. Note the type = "class" argument, which sets the format of the model output to be the levels from the response variable.
Calculate the error rate by comparing start_position to model_output.
Construct a set of random guesses. The command to do this, based on shuffle(), is provided in the editor.
Calculate the error rate by comparing start_position to the random guess.

```r
Runners$all_the_same <- 1 # null "explanatory" variable
null_model <- rpart(start_position ~ all_the_same, data = Runners)

# Evaluate the null model on training data
null_model_output <- evaluate_model(null_model, data = Runners, type = "class")

# Calculate the error rate
with(data = null_model_output, mean(start_position != model_output, na.rm = TRUE))
```

```
## [1] 0.5853618
```

```r
# Generate a random guess...
null_model_output$random_guess <- shuffle(Runners$start_position)

# ...and find the error rate
with(data = null_model_output, mean(start_position != random_guess, na.rm = TRUE))
```

```
## [1] 0.6555211
```

Note that random guessing does not perform as well as the null model.

A non-null model

In the previous exercise, you saw that the null model performs better at classification than random guessing. The error rate you found for the null model was 58.5%, whereas random guessing gave an error of about 66%.

In this exercise, you'll build a model of start_position as a function of age and sex. Train an rpart() model on Runners with model formula start_position ~ age + sex. Use a complexity parameter of cp = 0.001. To calculate the in-sample error rate, first get the model output when using the training data as input. Be sure to ask for the output to be in the form of classes, rather than probabilities of each class. That's what type = "class" is for. Compare the model output to the actual values of the response variable to get the error rate.

```r
# Train the model
model <- rpart(start_position ~ age + sex, data = Runners, cp = 0.001)

# Get model output with the training data as input
model_output <- evaluate_model(model, data = Runners, type = "class")

# Find the error rate
with(data = model_output, mean(model_output!=start_position, na.rm = TRUE))
```

```
## [1] 0.5567794
```

The model using age and sex to predict start_position has an in-sample error rate that's only slightly better than that of the null model. Is this because age and sex are predictive, or because you used the training data to calculate the error rate? The cross validation estimate of error rate will tell you.

A better model?

In the previous two exercises, you compared a null model of start_position to a model using age and sex as explanatory variables. You didn't use cross validation, so the calculated error rates are biased to be low. In this exercise, you'll apply a simple cross validation test: splitting the data into training and testing sets.

Your job is to evaluate the models on the testing sets and calculate the error rate.

A hint about interpreting the results: it's often the case that explanatory variables that you think should contribute to prediction in fact do not. Being able to reliably discern when potential explanatory variables do not help is a key skill in modeling.

Evaluate each of the three models on the testing cases. Calculate the prediction error rate for each model. Print the error rates to the console and think about the following: Does adding age as an explanatory variable improve predictions over the null model? Does adding sex improve predictions over just using age? A proper calculation would repeat the random division into training and testing several times in order to be able to decide if the models have prediction errors that are statistically different. This is what cv_pred_error() does.

```
Training_data<-read.csv("Training_data.csv")
Testing_data<-read.csv("Testing_data.csv")
null_model <- rpart(start_position ~ all_the_same,
                    data = Training_data, cp = 0.001)
model_1 <- rpart(start_position ~ age,
                 data = Training_data, cp = 0.001)
model_2 <- rpart(start_position ~ age + sex,
                 data = Training_data, cp = 0.001)

# Find the out-of-sample error rate
null_output <- evaluate_model(null_model, data = Testing_data, type = "class")
model_1_output <- evaluate_model(model_1, data = Testing_data, type = "class")
model_2_output <- evaluate_model(model_2, data = Testing_data, type = "class")

# Calculate the error rates
null_rate <- with(data = null_output,
                  mean(start_position != model_output, na.rm = TRUE))
model_1_rate <- with(data = model_1_output,
                     mean(start_position != model_output, na.rm = TRUE))
model_2_rate <- with(data = model_2_output,
                     mean(start_position != model_output, na.rm = TRUE))

# Display the error rates
null_rate
```

```
## [1] 0.6448598
```

```
model_1_rate
```

```
## [1] 0.7009346
```

```
model_2_rate
```

```
## [1] 0.6728972
```

A more proper calculation would repeat the random division into training and testing several times in order to be able to decide if the models have prediction errors that are statistically different.

Exploring data for relationships

Till now we have been looking at decisions about model and explanatory variables based on prediction error.Now we will see, how models can be used to examine relationships in a mass of data. It is very difficult to analyse long pages of data.

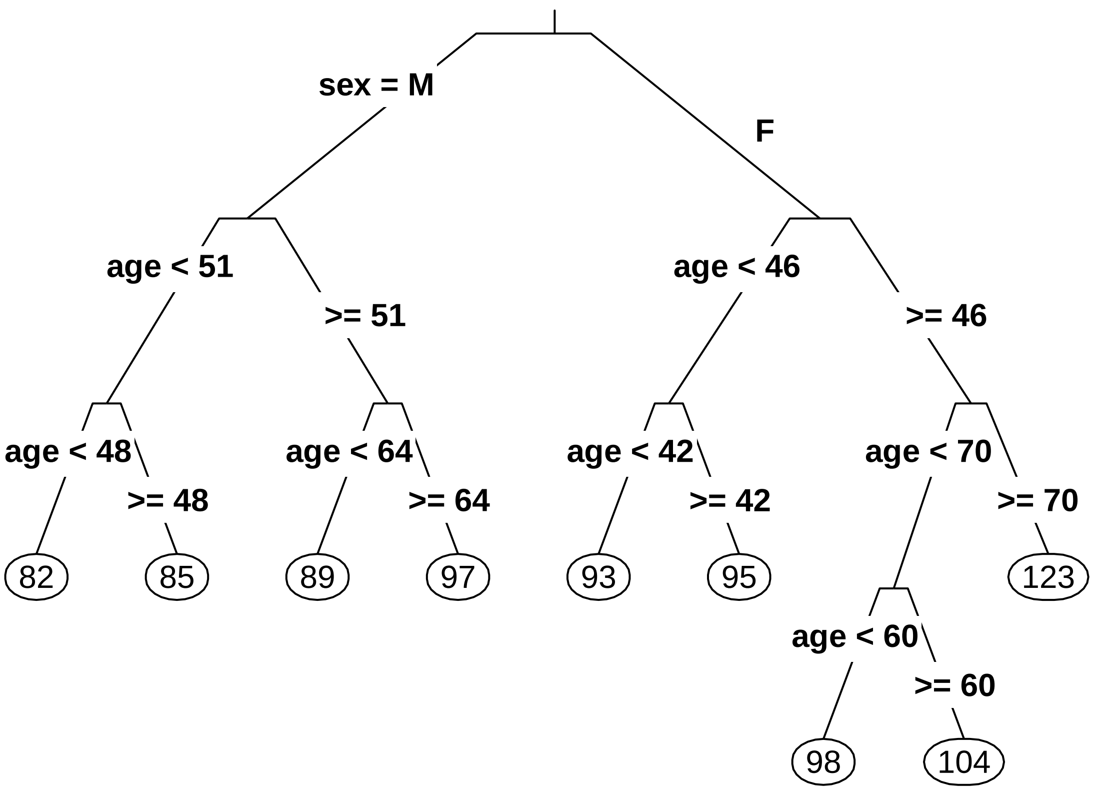
Evaluating a recursive partitioning model Consider this model formula about runners' net times: net ~ age + sex. The graphic shows the recursive partitioning for this formula. At the very bottom of the tree, in circles, are values for the response variable. At the very top is the root. (Modeling convention is to draw such trees upside down compared to the familiar botantical form, where the roots are at the bottom.)

Training an rpart() model amounts to finding a set of divisions of the cases. Starting with all the cases at the root, the model divides them up into two groups: males on the left and females on the right. For males, a further split is made based on age: those younger than 50 and those 50 and over. Similarly, females are also split on age, with a cut-point of 46 years. So, for a 40 year-old female, the model output is 93 (with the same units as the response variable: minutes).

The rpart() function uses a sensible default for when to stop dividing subgroups. You can exercise some control over this with the cp argument.

The prp() function plots the model as a tree. [Explore and try this out]

```
library(rpart.plot)
model_2 <- rpart(net ~ age + sex, data = Runners, cp = 0.001)
prp(model_2, type = 3)
```



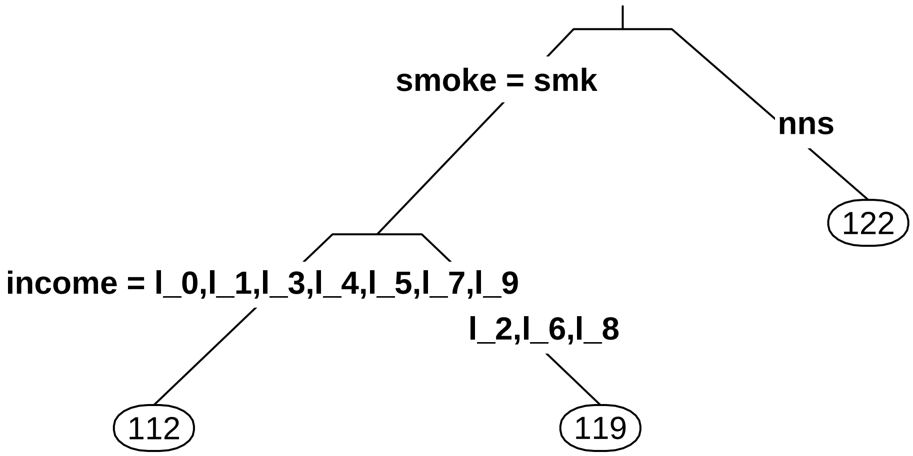
The model output for a 58 year-old female will be 98 minutes

We will try this for one more dataset: The Birth_weight data frame comes from a study of the risks factors for being underweight at birth. Let's explore the factors that might be related to birth weight.

One way to explore data is by building models with explanatory variables that you think are important, but in my view this is really confirmation rather than exploration. For instance, consider these models. The first involves explanatory variables that relate to social or lifestyle choices and the second involves biological variables. (Note: income is given as one of 8 levels, from poorest to richest. baby_wt is in ounces: 105 ounces is one kilogram.)

```
model_1 <- rpart(baby_wt ~ smoke + income,
                 data = Birth_weight)
model_2 <- rpart(baby_wt ~ mother_age + mother_wt,
                 data = Birth_weight)
```

```
prp(model_1, type = 3)
```



The results might suggest to you that some of these explanatory variables are important and others aren't. Now build a "bigger" model on your own, combining all of those variables. Based on this "bigger" model, interpret the relationship among the explanatory variables as they relate to baby_wt.

Covariates and Effect Size

Real-world systems are complicated. To faithfully reflect that complexity, models can incorporate multiple explanatory variables. This section introduces the notion of covariates and how they allow you to model the effect of an explanatory variable while taking into account the effects of other variables.
Covariates are explanatory variables that are not themselves of interest to the modeler, but which may shape the response variable. The name is given to show the modeller's attitude towards the variable, it might be important but its not the primary interest.

House prices Lets assume, a real estate firm, wants you to model the value of various amenities in houses. This task is to calculate the value of a fireplace. For this purpose, they have provided you with a dataset, Houses_for_sale.

A newcomer to modeling might assume that the way to address this question is with a simple model: price ~ fireplaces. Let's start there.

As you'll see, a covariate can make a big difference! Train a linear model using Houses_for_sale with the formula price ~ fireplaces. Call the model simple_model. Use evaluate_model() to see what is the difference in price (according to the model) for houses with and without a fireplace. Copy and paste values from the evaluate_model() output to do a simple subtraction to find the difference in price. Store it under the name simple_worth. The name simple_worth might clue you in that there's more going on. Make a second model in which you add a covariate living_area. Call the model sophisticated_model. Use evaluate_model() to get the model output for a few values of the inputs. Copy and paste values from the evaluate_model() output to compare the change in price when you hold living_area constant at 2000 sq. feet for houses with and without a fireplace. Do the subtraction and store the result as sophisticated_worth. Are the values of the worth of a fireplace similar for the two models?

```
# Train the model price ~ fireplaces
simple_model <- lm(price ~ fireplaces, data = Houses_for_sale)

# Evaluate simple_model
evaluate_model(simple_model)
```

```
##   fireplaces model_output
```

```
## 1          0     171823.9
## 2          1     238522.7
```

```
# Calculate the difference in model price
simple_worth <- 238522.7 - 171823.9

# Train another model including living_area
sophisticated_model <- lm(price ~ fireplaces + living_area, data = Houses_for_sale)

# Evaluate that model
evaluate_model(sophisticated_model)
```

```
##   fireplaces living_area model_output
## 1          0        1000     124043.6
## 2          1        1000     133006.1
## 3          0        2000     233357.1
## 4          1        2000     242319.5
## 5          0        3000     342670.6
## 6          1        3000     351633.0
```

```
# Find price difference for fixed living_area
sophisticated_worth <- 242319.5 - 233357.1
```

Since the number of fireplaces is related to the amount of living area, a model of price built with just one of the variables will fail to distinguish between the separate effects of each.

Crime and poverty

The data frame Crime gives some FBI statistics on crime in the various US states in 1960.

The variable R gives the crime rate in each state. The variable X gives the number of families with low income (i.e. less than half the median). The variable W gives the average assets of families in the state. You're going to build separate models R ~ X and R ~ W to estimate what the effect on the crime rate is of each of those variables. Then you'll construct R ~ X + W, using each of the explanatory variables as a covariate for the other.

```
# Train model_1 and model_2
model_1 <- lm(R ~ X, data = Crime)
model_2 <- lm(R ~ W, data = Crime)

# Evaluate each model...
evaluate_model(model_1)
```

```
##     X model_output
## 1 100    106.82223
## 2 200     89.46721
## 3 300     72.11219
```

```
evaluate_model(model_2)
```

```
##     W model_output
## 1 400     68.32909
## 2 600    103.70777
## 3 800    139.08644
```

```
# ...and calculate the difference in output for each
change_with_X <- 89.46721 - 106.82223
```

```r
change_with_W <- 103.70777 - 68.32909

# Train model_3 using both X and W as explanatory variables
model_3 <- lm(R ~ X + W, data = Crime)

# Evaluate model_3
evaluate_model(model_3)
```

```
##     X   W model_output
## 1 100 400    -62.60510
## 2 200 400     31.03422
## 3 300 400    124.67354
## 4 100 600     41.22502
## 5 200 600    134.86434
## 6 300 600    228.50366
## 7 100 800    145.05515
## 8 200 800    238.69447
## 9 300 800    332.33379
```

```r
# Find the difference in output for each of X and W
change_with_X_holding_W_constant <- 228.50366 - 134.86434
change_with_W_holding_X_constant <- 134.86434 - 31.03422
```

Notice that the change with X is negative in the model that doesn't include W as a covariate. That is, the model is indicating that having more families with low income reduces the crime rate. But changing X while holding W constant gives a strongly positive relationship. This is because areas with high X tend to have low W: low income is associated with low assets. And low assets tend to be associated with less crime.

Equal pay? Gender pay equity is a matter of considerable concern. That's the setting for this task. Keep in mind that the issue is complicated and the data for this exercise are very limited, so don't draw broad conclusions. Instead, focus on the methods: how does the introduction of covariates change the story told by the models?

You'll be working with data (Trucking_jobs) from a trucking company, giving information about the earnings of 129 employees. The primary interest is whether earnings differ by sex.

Potential covariates are:

Simple personal information: age and hiredyears. Type of work done, as represented by the person's job title. You will build five models of earnings using a linear model architecture. The first has no covariates. Others include each of the covariates singly and the final one includes all of the covariates.

earnings ~ sex earnings ~ sex + age earnings ~ sex + hiredyears earnings ~ sex + title earnings ~ sex + age + hiredyears + title

Build the five models, naming them model_1 through model_5. Use evaluate_model() to see the model output from each so that you can compare the earnings of the two sexes. When a covariate is included, choose one level for the covariate and compare the two sexes at a fixed level for that covariate. You may need to look at the data to find valid levels for the title variable. Store the differences you find for each model as diff_1, diff_2, and so on. Note whether the difference in earnings between the sexes changes from one model to another.

```r
# Train the five models
model_1 <- lm(earnings ~ sex, data = Trucking_jobs)
model_2 <- lm(earnings ~ sex + age, data = Trucking_jobs)
model_3 <- lm(earnings ~ sex + hiredyears, data = Trucking_jobs)
model_4 <- lm(earnings ~ sex + title, data = Trucking_jobs)
model_5 <- lm(earnings ~ sex + age + hiredyears + title, data = Trucking_jobs)
```

```r
# Evaluate each model ...
evaluate_model(model_1)
```

```
##   sex model_output
## 1   M     40236.35
## 2   F     35501.25
```

```r
evaluate_model(model_2, age = 30)
```

```
##   sex age model_output
## 1   M  30     35138.86
## 2   F  30     32784.54
```

```r
evaluate_model(model_3, hiredyears = 5)
```

```
##   sex hiredyears model_output
## 1   M          5     39996.93
## 2   F          5     36366.89
```

```r
evaluate_model(model_4, title = "REGL CARRIER REP")
```

```
##   sex            title model_output
## 1   M REGL CARRIER REP     27838.38
## 2   F REGL CARRIER REP     28170.71
```

```r
evaluate_model(model_5, age = 20, hiredyears = 0,
               title = "OUTSIDE REGIONAL ACCOUNT MGR")
```

```
##   sex age hiredyears                        title model_output
## 1   M  20          0 OUTSIDE REGIONAL ACCOUNT MGR     61497.72
## 2   F  20          0 OUTSIDE REGIONAL ACCOUNT MGR     61513.00
```

```r
# ... and calculate the gender difference in earnings
diff_1 <- 40236.35 - 35501.25
diff_2 <- 35138.86 - 32784.54
diff_3 <- 39996.93 - 36366.89
diff_4 <- 27838.38 - 28170.71
diff_5 <- 61497.72 - 61513.00
```

The results differ from model to model. By including a covariate, you are effectively determining which two groups to compare (e.g. Men vs. Women both of age 30, or men vs. Women with a given job title.)

Effect size

When covariates are selected to the way we think the system works,in the real world can do meaningful experiments on the model to quantify how variables are related. We can measure how much does the model output change for a given change in input. This is known as effect size. To use a word like "effect" implies some sort of causation. In any models, the inputs cause the outputs. Effect sizes are often constructed specifically to represent cause and effect. Modeler's interest is often in cause and effect. The effect size can be quantified as a rate or a difference. For quantitative input, effect size is represented as a rate.
For categorical input , the units of effect size are those of the response variable(As catagorical variables do not have units) and will be represented as a difference.

How do GPAs compare?

The performance of university and high-school students in the US are often summarized by a "gradepoint average" (GPA). The grade that a student earns in each course is translated to a numerical scale called a gradepoint: 4.0 is at the high end (corresponding to an "A") and 0 is at the low end (a fail).

The GPA calculation is done, of course, by taking a student's gradepoints and averaging. But this is not the only way to do it. gpa_mod_1 in the editor shows a gradepoint average calculation using a linear model. The data, College_grades, give the grades in each course taken by each of 400+ students at an actual college in the midwest US. sid is the student's ID number. The formula gradepoint ~ sid can be read, "gradepoint is explained by who the student is."

Evaluating the model for students "S32115" and "S32262" shows that they have very similar gradepoint averages: 3.66 and 3.33, respectively.

The effect_size() calculation compares two levels of the inputs. You could get this result through simple subtraction of the evaluated model values. By default, effect_size() picks the levels to compare, but you can override this by providing specific evaluation level(s) of explanatory variables (e.g. sid = "S32115") and the to argument (e.g. to = "S32262").

```
# Calculating the GPA
gpa_mod_1 <- lm(gradepoint ~ sid, data = College_grades)

# The GPA for two students
evaluate_model(gpa_mod_1, sid = c("S32115", "S32262"))
```

```
##      sid model_output
## 1 S32115     3.448571
## 2 S32262     3.442500
```

```
# Use effect_size()
effect_size(gpa_mod_1, ~ sid)
```

```
##      change    sid to:sid
## 1 0.4886364 S32259 S32364
```

```
# Specify from and to levels to compare
effect_size(gpa_mod_1, ~ sid, sid = "S32115", to = "S32262")
```

```
##         change    sid to:sid
## 1 -0.006071429 S32115 S32262
```

```
# A better model?
gpa_mod_2 <- lm(gradepoint ~ sid + dept + level, data = College_grades)

# Find difference between the same two students as before
effect_size(gpa_mod_2, ~ sid, sid = "S32115", to = "S32262")
```

```
##      change    sid to:sid dept level
## 1 0.4216295 S32115 S32262    d   200
```