Lecture 26

Peter Shaffery

4/8/2021

Introduction to Missing Data

Lectures follow chapters of "Statistical Analysis with Missing Data" (Little and Rubin, 2002)

One of the most common problems "data problems" you will face in statistical analysis is *missing data*. Missing data are what they sound like, datapoints which have incomplete information about one or more units in our analysis.

Consider, for example, the titanic dataset

```
library(tidyverse)
library(magrittr)
library(rstanarm)

dat=read.csv('../../data/titanic.csv')
```

Computational Side

Missing data is commonly represented in computer records by the letters `NA`, meaning "Not Available". R treats `NA` as a special quantity, which has it's own rules and operations:

```
NA + 1
```

```
## [1] NA
```

```
NA == 1
```

```
## [1] NA
```

```
NA == NA
```

```
## [1] NA
```

```
c(NA,1)
```

```
## [1] NA  1
```

```
paste(NA,'foo',sep='')
```

```
## [1] "NAfoo"
```

Notice in that last example, the command `paste` (which concatenates strings, `paste('a','b',sep=',')` outputs `"a,b"`) doesnt treat `NA` as "missing data", but actually converts it to a string. This is one of the most annoying things that can happen with `NA`s: they will sometimes be converted (or recorded) as "string NAs", `"NA"` (or even worse, `"na"`). This *does not* have the special properties of `NA` and will be the bane of your existence if you're not careful about checking for them:

```
is.na('NA')
```

```
## [1] FALSE
```

```
is.na(NA)
```

```
## [1] TRUE
```

One way to keep track of all these is to use a judicious application `dplyr::na_if`:

```
bad.dat = data.frame(
  'idx'=c(1,2,3),
  'missing1'=c(NA,'a','b'),
  'missing2'=c('what','the','na')
  )

bad.dat %>%
  na_if('na')
```

```
##   idx missing1 missing2
## 1   1     <NA>     what
## 2   2        a      the
## 3   3        b     <NA>
```

To further complicate matters, sometimes `NA`s will be represents using `NaN` (or `Nan` or `nan`), meaning "Not a Number".

```
is.nan(NaN)
```

```
## [1] TRUE
```

```
is.nan(NA)
```

```
## [1] FALSE
```

This one is a pain to handle:

```
nightmare.dat = data.frame(
  'idx'=c(1,2,3),
  'missing1'=c(NA,'a','b'),
  'missing2'=c(2.43,NaN,31.4),
  'missing3'=c('what','the','na')
  )

nightmare.dat %>%
  na_if(NaN) %>%
  na_if('na')
```

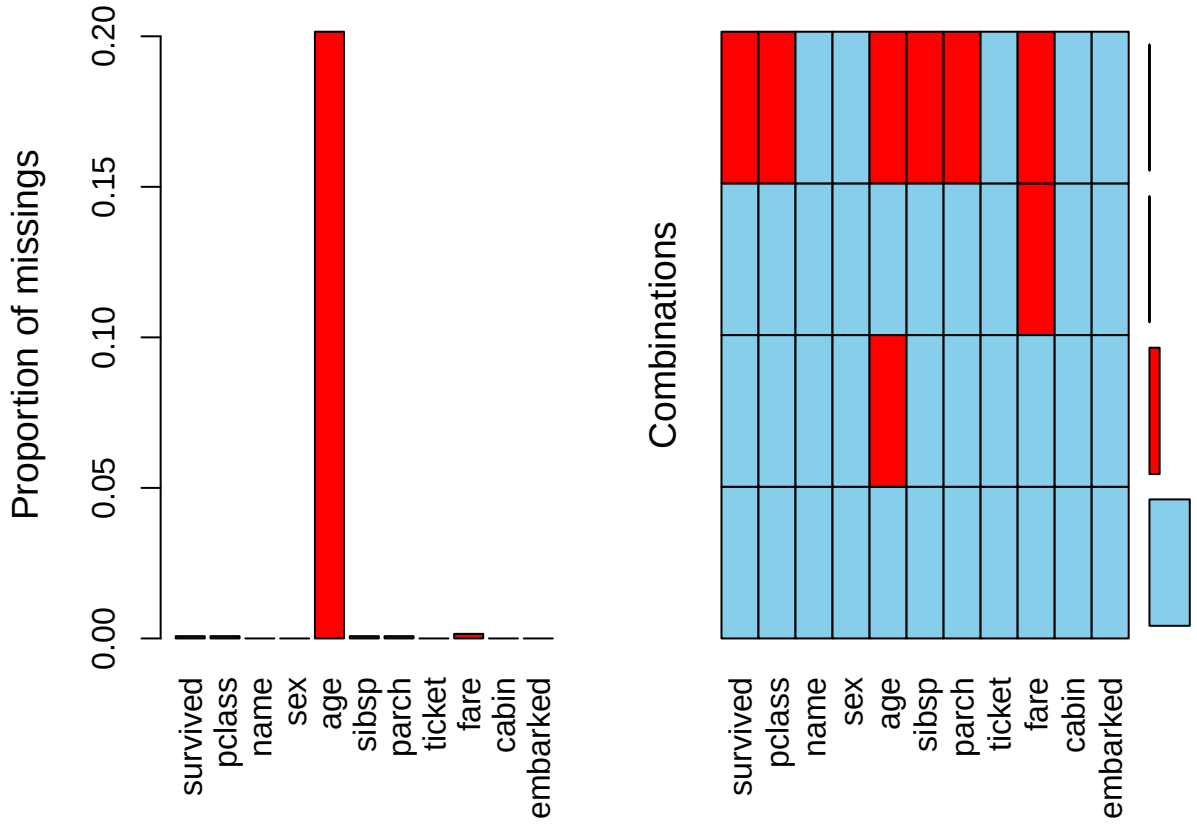
```
##   idx missing1 missing2 missing3
## 1   1     <NA>     2.43     what
## 2   2        a      NaN      the
## 3   3        b    31.40     <NA>
```

In general, it's fine to just keep true `NaN`s as they occur, because in most situations they will behave sufficiently like `NA`s to not cause problems, but be aware that these are different concepts.

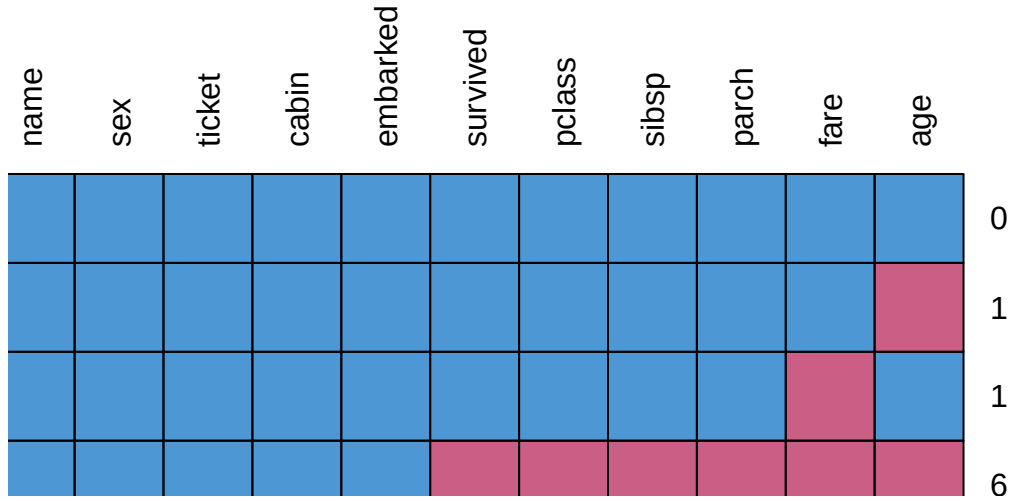
Once you've cleaned up your `NA`s, the next step is to determine how prevalent they are. One very useful tool for this are "missing data patterns":

```
library(mice) # "Multiple Imputation by Chained Equations
library(VIM) # Visualization and Imputation of Missing Data
```

```
aggr(dat) # VIM version
```




```
md.pattern(dat, rotate.names=TRUE) # mice version
```




```
##      name sex ticket cabin embarked survived pclass sibsp parch fare age
## 1045    1   1      1     1        1        1      1     1     1    1   0
## 263     1   1      1     1        1        1      1     1     1    0   1
## 1       1   1      1     1        1        1      1     1     1    0   1
## 1       1   1      1     1        1        0      0     0     0    0   6
##         0   0      0     0        0        1      1     1     1    2 264 270
```

```
dat %<>% drop_na(-age)
```

In the `titanic` dataset we see that AGE variable has the highest amount of missingness by far.

Statistical Models with Missing Data

In the past we've dealt with missingness by simply dropping the rows that have missing values:

```
# clearly age matters
mod = glm(survived ~ sex + as.factor(pclass) + sibsp + age + fare, data=dat%>%drop_na, family=binomial)
summary(mod)
```

```
##
## Call:
## glm(formula = survived ~ sex + as.factor(pclass) + sibsp + age +
##     fare, family = binomial, data = dat %>% drop_na)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6972  -0.6637  -0.4246   0.6675   2.5198
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)          3.807305   0.397517   9.578  < 2e-16 ***
## sexmale             -2.567957   0.171414 -14.981  < 2e-16 ***
## as.factor(pclass)2 -1.273143   0.259137  -4.913 8.97e-07 ***
## as.factor(pclass)3 -2.235679   0.269341  -8.301  < 2e-16 ***
## sibsp               -0.342717   0.101887  -3.364 0.000769 ***
## age                 -0.039320   0.006641  -5.920 3.21e-09 ***
## fare                 0.001484   0.001902   0.780 0.435239
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1413.57  on 1044  degrees of freedom
## Residual deviance:  969.97  on 1038  degrees of freedom
## AIC: 983.97
##
## Number of Fisher Scoring iterations: 4
```

This strategy for dealing with missing data analysis is referred to as **complete case analysis**. While it is certainly the *easiest* way of dealing with missing data, it does have some drawbacks.

For one, we're dumping a lot of data, 263 rows is like 20% of the dataset, and that's kind of as waste. Moreover, we may be losing important information:

```
dat %>% group_by(sex,pclass) %>% summarize(sum(is.na(age)))
```

```
## # A tibble: 6 x 3
## # Groups:   sex [2]
##   sex    pclass `sum(is.na(age))`
##   <chr>   <int>             <int>
## 1 female      1                11
## 2 female      2                 3
## 3 female      3                64
```

```
## 4 male        1                28
## 5 male        2                13
## 6 male        3               144
```

The missingness here is imbalanced across categories! Therefore the decision to ignore the missing data could bias our inference regarding these (or other) independent variables.

What we need is a technique that will allow us to handle the missingness of some data. There are many possible ways that we could do so. The choice of method, and it's ultimate efficacy, hinges on the **missing data mechanism**; how the data came to be missing in the first place.

Missingness Mechanisms

Missingness mechanisms fall into three important categories:

1. **Not Missing at Random (NMAR)**
2. **Missing At Random (MAR)**
3. **Missing Completely at Random (MCAR)**

In order to define these categories rigorously, we have to first introduce some notation.

Say that we have a dataset D, which includes both our vector of dependent variables (outcomes) $\vec{y} = [y_1, ..., y_n]^T$, as well as our matrix of independent variables $X = [\vec{x}_1, ..., \vec{x}_n]^T$. Furthermore, say that for some of the observations $D_i = \texttt{NA}$, which means that y_i or an element(s) of $\vec{x}_i$ are missing.

Now, we'll define a second "missingness vector" $\vec{m} = [m_1, ..., m_n]^T$, where $m_i = \begin{cases} 1, D_i = \texttt{NA} \\ 0, D_i \neq \texttt{NA} \end{cases}$. We will interpret $\vec{m}$ as a "mask" for the data: when $m_i = 1$ we are prevented from knowing the true value of D_i. Furthermore, we will sometimes use $\vec{D}_{\text{obs}}$ and $\vec{D}_{\text{mis}}$ to denote the observed and missing components of the data.

The reason we define the missingness vector $\vec{m}$ separately, is that it allows us to endow it with a probability distribution which depends both on a parameter(s) ϕ, as well as on the data itself $\vec{y}$:

$$\vec{m} \sim P[\vec{m}|D, \phi]$$

Thinking about missing data in this way, as a random variable with a distribution that can be estimated or learned about, is the core of statistical imputation. Moreover, we can use the specific form of $P[\vec{m}|\vec{y}, \phi]$ to differentiate between different types of missingness mechanisms.

Not Missing at Randoms

NMAR data is the worst case, and depending on context may just not be addressable from a statistical point of view. NMAR data is defined by a missingness mechanism of the form:

$$\vec{m} \sim P[\vec{m}|\vec{D}_{\text{mis}}, \phi]$$

A simple example of this would be $y_i \sim N(\mu, \sigma^2)$, where we want to estimate μ and σ^2. Now, let's say that due to a flaw in our measuring tool, there is a threshold τ, and if $y_i > \tau$ then y_i is set equal to `NA`. Clearly:

$$m_i \sim P[y_i > \tau] = \int_{\tau}^{\infty} N(y_i; \mu, \sigma^2) dy_i$$

For a more realistic example, say that we are performing some experiment on mice in three age groups, and that we expect age to be an important determinant of individuals' response to the treatment. If our new lab intern screws up some component of the data collection process, and all of the samples in the oldest age category are contaminated, there's not much you can do to recover that data.

NMAR data can be further broken down into two cases: data which is *censored* (where the specifics of the missing data mechanism, eq. τ) are known, and data which are not censored.

Censored data can sometimes be dealt with by incorporating the knowledge of the censoring mechanism into the model likelihood. One example of this is longitudinal medical studies, where patients who survive long enough often just drop out of a study. You can still use their last known survival time as a component in the analysis.

On the other hand, if the censoring mechanism is unknown to you, then you're pretty much out of luck. In this case the only recourse (besides collecting new data) is complete case analysis: dropping the portions of the data which were excluded, and being explicit about that constraint in your analysis.

Missing Completely at Random

At the other end of the spectrum is MCAR data. This is defined by a missingness mechanism that has no dependence on the data:

$$\vec{m} \sim P[\vec{m}|\phi]$$

MCAR data is the best case, because it impacts your sample size and nothing else. Consider again the example of our clumsy intern, except this time it's just a random 10% of the individual outcomes which are contaminated, rather than any particular age group. Since the samples in each age group are iid, this is effectively just a reduction in the number of measurements within each group. So long as our initial sample size was sufficiently large, we should be fine to just ignore the missing data and perform a complete case analysis.

Missing at Random

Most of the time missing data will not fall into either extreme, but will instead be just MAR. Data which is MAR has the missingness mechanism:

$$\vec{m} \sim P[\vec{m}|D_{\text{obs}}, \phi]$$

That is, the missingess of either y_i or m_i, depends on some variable which we have measured and which is never missing.

One example of this is our `titanic` dataset:

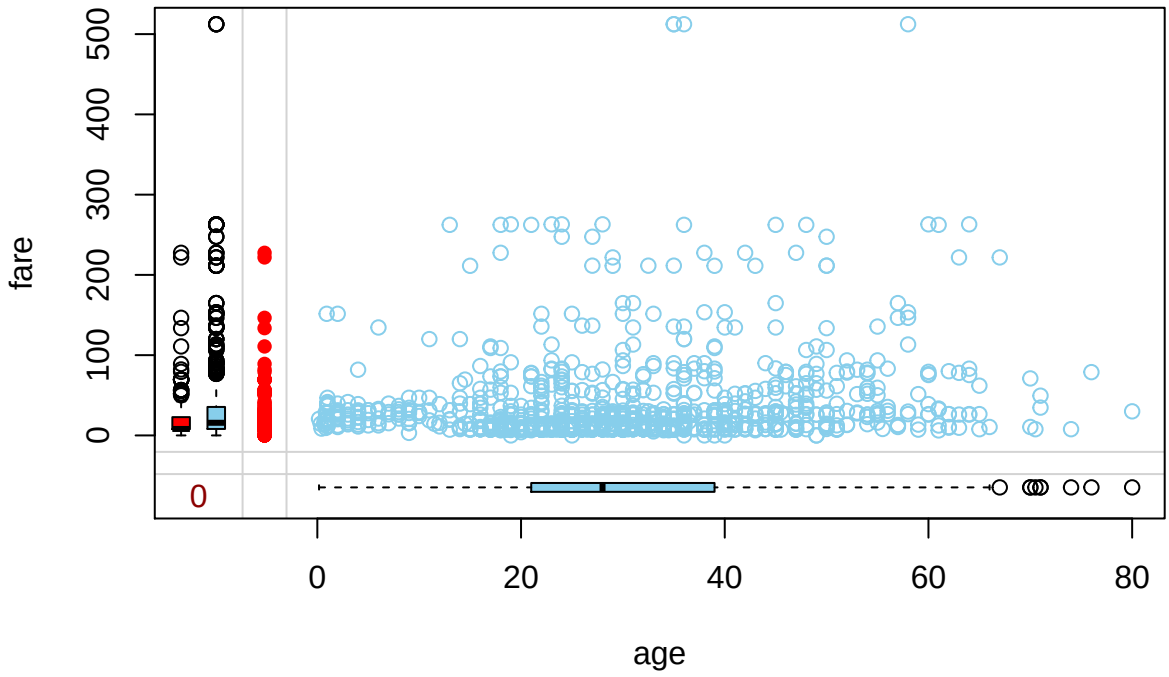
```
dat %>% group_by(sex,pclass) %>% summarize(sum(is.na(age)))
```

```
## # A tibble: 6 x 3
## # Groups:   sex [2]
##   sex    pclass `sum(is.na(age))`
##   <chr>   <int>             <int>
## 1 female      1                11
## 2 female      2                 3
## 3 female      3                64
## 4 male        1                28
## 5 male        2                13
## 6 male        3               144
```

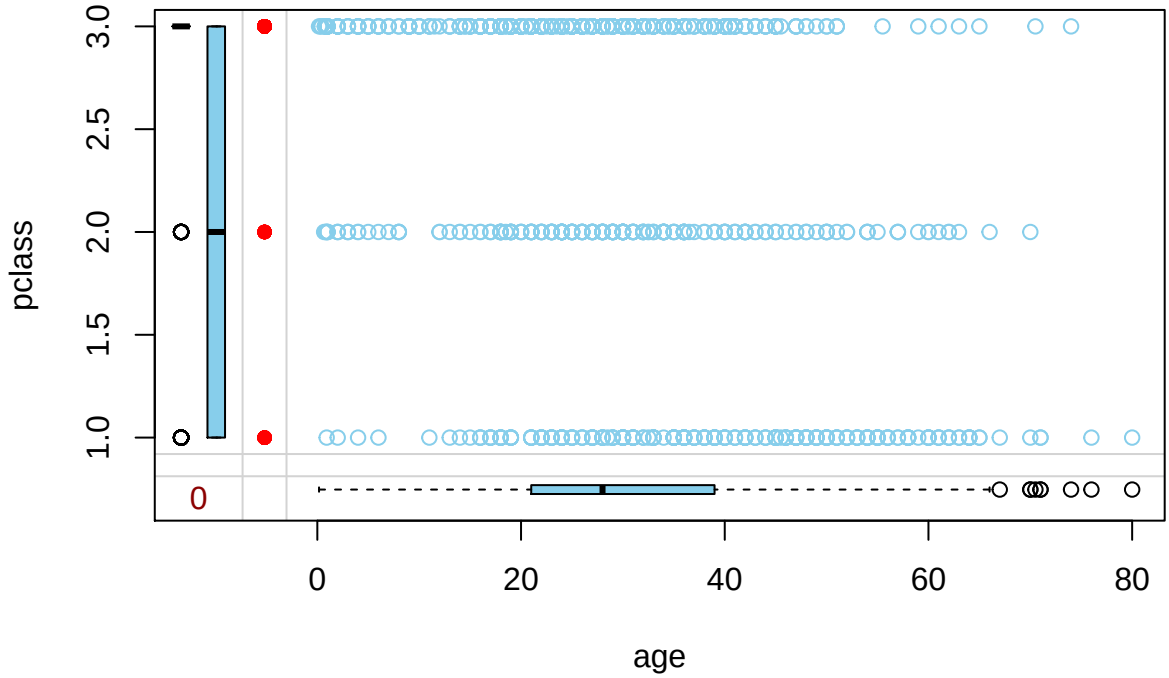
We see that the missingess of AGE has clear dependence on the variables SEX and PCLASS, both of which are completely observed (except for like one case which we'll just ignore).

Another way that we could assess this is through a margin plot:

```
marginplot(dat[,c('age','fare')])
```




```
marginplot(dat[,c('age','pclass')])
```



While with NMAR and MAR data our main tool was complete case analysis (or special methods, in the case of censored NMAR data), with MCAR data we have a few other options which are more appropriate:

1. *Weighting Methods:* Estimate $\pi_i = P[m_i|D_{\text{obs}}, \phi]$, and then use it to weight the data in some smart way. For example, say that we have iid normal data y_i, with dependent variable x_i which is equal to 0 or 1. Say that the missingness mechansim is $P[m_i = 1|x_i = 0] = .25$ and $P[m_i = 1|x_i = 1] = .75$. The "correct" estimate of the normal mean μ, is the weighted sample mean $\hat{mu} = \frac{1}{n}\sum_i \bar{\pi}_i^{-1} y_i$, where $\bar{\pi}_i$ are just the normalized weights $\pi_i = \frac{\pi_i}{\sum_i \pi_i}$. You will sometimes hear π_i referred to as the **design weights** (this is a common technique in survey analysis to deal with nonresponse)

2. *Imputation Methods:* Use your knowledge $P[m_i|D_{\text{obs}}, \phi]$ to generate samples of *synthetic data*, $\hat{D}_{\text{mis}}$. You then treat these samples as if they were "real" measurements, and fit your model with the sampled data $\hat{D} = [D_{\text{obs}}, \hat{D}_{\text{mis}}]$. Typically this is done as part of a **multiple imputation** strategy, where you create a few different samples $\hat{D}^{(i)}$, and then you average the resulting model coefficients obtained from each $\hat{D}^{(i)}$ This is the strategy that we will focus on for the rest of lecture.

3. *Model-Based Methods:* Incorporate your knowledge of $P[m_i|D_{\text{obs}}, \phi]$ directly into your model likelihood (like with the censored data mentioned above). This is a very rich approach to missing data problems, but because its so contextual we won't be able to give it much space here. For a detailed discussion check out Parts II and III of the textbook mentioned at the top of the lecture.

Imputation

The idea with imputation is that we will "fill-in" the missing data with some reasonable guesses, and then fit our model to the filled-in dataset. What differentiates the varieties of imputation is *how* you fill in the data. Very broadly, these method fall into two classes: **explicit** method, where you actually fit a statistical model for the missing data, and **implicit** methods, where you use observed data as a "surrogate" for the model.

Explicit Methods

Mean Imputation

The simplest way of imputing missing data is to just construct some suitable subgroups in your data, and replace missing data with subgroup-level means. For the titanic dataset, a crude implementation of this strategy would look like:

```
age.mn = dat %>%
  group_by(sex,pclass) %>%
  transmute(age.mn = mean(age,na.rm=TRUE)) %>%
  ungroup %>%
  select(age.mn)

dat$age.imputed = coalesce(dat$age,unlist(age.mn))

mod.mean.imputed = glm(survived ~ sex + as.factor(pclass) + sibsp + age.imputed + fare, data=dat, family
coefficients(mod)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##        3.807304714       -2.567956904       -1.273142868       -2.235679273
##              sibsp                age               fare
##       -0.342717356       -0.039319948        0.001483718
```

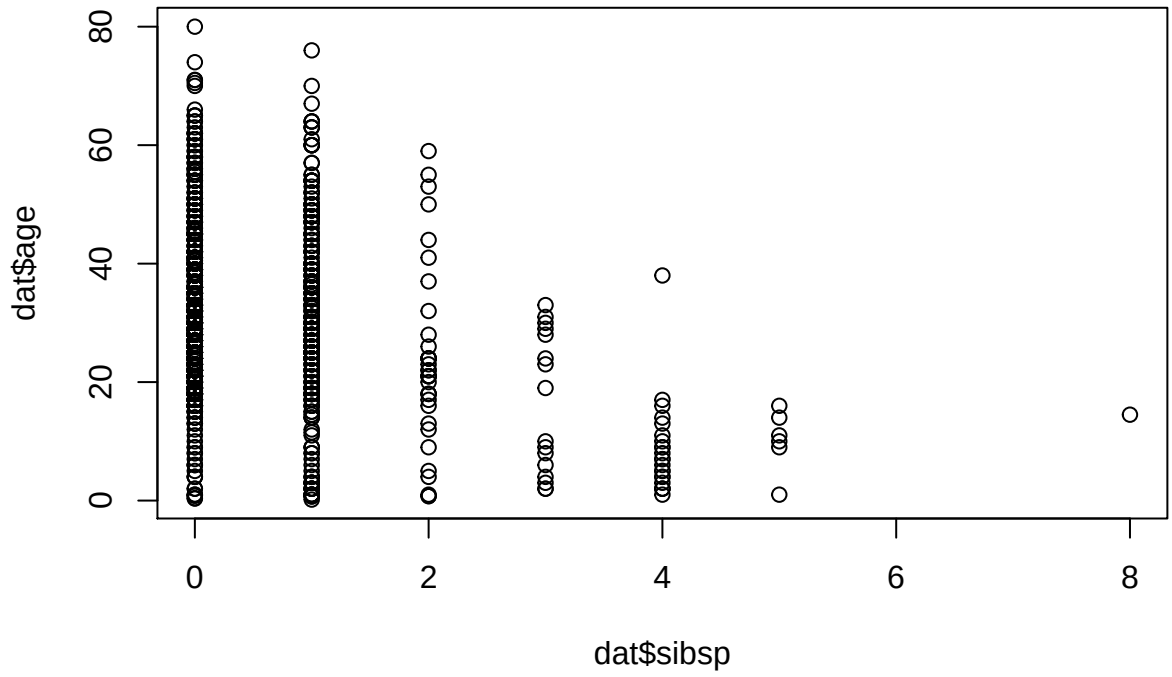
```
coefficients(mod.mean.imputed)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##        3.719318180       -2.553445232       -1.230501115       -2.205888284
##              sibsp        age.imputed               fare
##       -0.345699627       -0.039283595        0.001791985
```

Regression Imputation

Fit a regression model for the missing variable(s) to some other variables of interest, and then impute missing data with the predicted regression mean. This method is conceptually identically to the above, except it also incorporates continuous predictors, as well as categorical (ie. group labels):

```
plot(dat$sibsp,dat$age) # introduce a non-categorical variable to differentiate from subgroup mean appr
```




```
age.mod = lm(age~sex+as.factor(pclass)+sibsp, data=dat%>%drop_na)
age.reg = predict(age.mod,newdata=dat)

dat$age.imputed = coalesce(dat$age,age.reg)

mod.reg.imputed = glm(survived ~ sex + as.factor(pclass) + sibsp + age.imputed + fare, data=dat, family=
coefficients(mod)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##        3.807304714       -2.567956904       -1.273142868       -2.235679273
##              sibsp                age               fare
##       -0.342717356       -0.039319948        0.001483718
```

```
coefficients(mod.reg.imputed)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##        3.740806855       -2.559331121       -1.232374919       -2.200134562
##              sibsp        age.imputed               fare
##       -0.387495275       -0.039185373        0.001818712
```

Stochastic Regression Imputation

As above, fit a regression model for the missing variable(s). However rather than using predicted regression mean, draw a sample from the full forecasting distribution for each missing datapoint. The idea here is to incorporate some additional uncertainty into the imputation, through the residuals.

```
set.seed(12345)
age.mod = lm(age~sex+as.factor(pclass)+sibsp, data=dat%>%drop_na)
sigma = sd(resid(age.mod))
age.stoch.reg = predict(age.mod,newdata=dat) + rnorm(nrow(dat), 0, sigma)

dat$age.imputed = coalesce(dat$age,age.stoch.reg)

mod.stoch.reg.imputed = glm(survived ~ sex + as.factor(pclass) + sibsp + age.imputed + fare, data=dat,
```

```
coefficients(mod)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##        3.807304714       -2.567956904       -1.273142868       -2.235679273
##              sibsp                age               fare
##       -0.342717356       -0.039319948        0.001483718
```

```
coefficients(mod.stoch.reg.imputed)
```

```
##        (Intercept)             sexmale as.factor(pclass)2 as.factor(pclass)3
##         3.54391873        -2.57567488        -1.18023564        -2.13315998
##              sibsp        age.imputed               fare
##       -0.38464480        -0.03378121         0.00181747
```

```
dat %<>% select(-age.imputed)
```

Implicit Methods

Hot Deck Imputation

The only implicit method that we will look at today is **hot deck** imputation. Unlike the explicit methods, which constructed a concrete model for the AGE variable, *implicit* method will sidestep the modeling stage. The idea here is that we will instead use the data itself as our "model".

Say that we have a datapoint D^*_{mis}, which contains some missing data that we would like to impute. The hot deck approach does so in two steps:

1. Using the non-missing components of D^*_{mis}, select (usually at random) a **donor** datapoint D_{donor} which has no missing data
2. Fill in the missing components of D^*_{mis} using the corresponding components of D_{donor}

For a single datapoint, we might implement this as:

```
set.seed(12345)

D.star = dat[is.na(dat$age),] %>% sample_n(1)

donor.sex = D.star$sex
donor.pclass = D.star$pclass
candidate.donor = (dat$sex==donor.sex)&(dat$pclass==donor.pclass)

D.donor = dat[candidate.donor,] %>% sample_n(1)
D.star.imputed = coalesce(D.star, D.donor)
```

Now, both steps 1 and 2 can be done in a lot of different ways. Donor candidates are usually select from a pool of "similar" candidates, but similarity can be measured in a lot of different ways. Similarly, how you pick a donor once you've got your candidate pool can vary a lot. You might choose donors weighted by similarity. You might choose different donors to impute different columns. It's pretty much all up for grabs.

In general, rather than making all these decisions yourself, it's simplest to just use some pre-designed packages:

```
reg.imputed = VIM::regressionImp(age~sex+pclass+fare, data=dat)
reg.imputed %>% head
```

```
##   survived pclass                                            name    sex
## 1        1      1                   Allen, Miss. Elisabeth Walton female
## 2        1      1                  Allison, Master. Hudson Trevor   male
## 3        0      1                    Allison, Miss. Helen Loraine female
## 4        0      1             Allison, Mr. Hudson Joshua Creighton   male
```

```
## 5        0      1 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6        1      1                              Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked age_imp
## 1 29.0000     0     0  24160 211.3375      B5        S   FALSE
## 2  0.9167     1     2 113781 151.5500 C22 C26        S   FALSE
## 3  2.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 4 30.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 5 25.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 6 48.0000     0     0  19952  26.5500     E12        S   FALSE
```

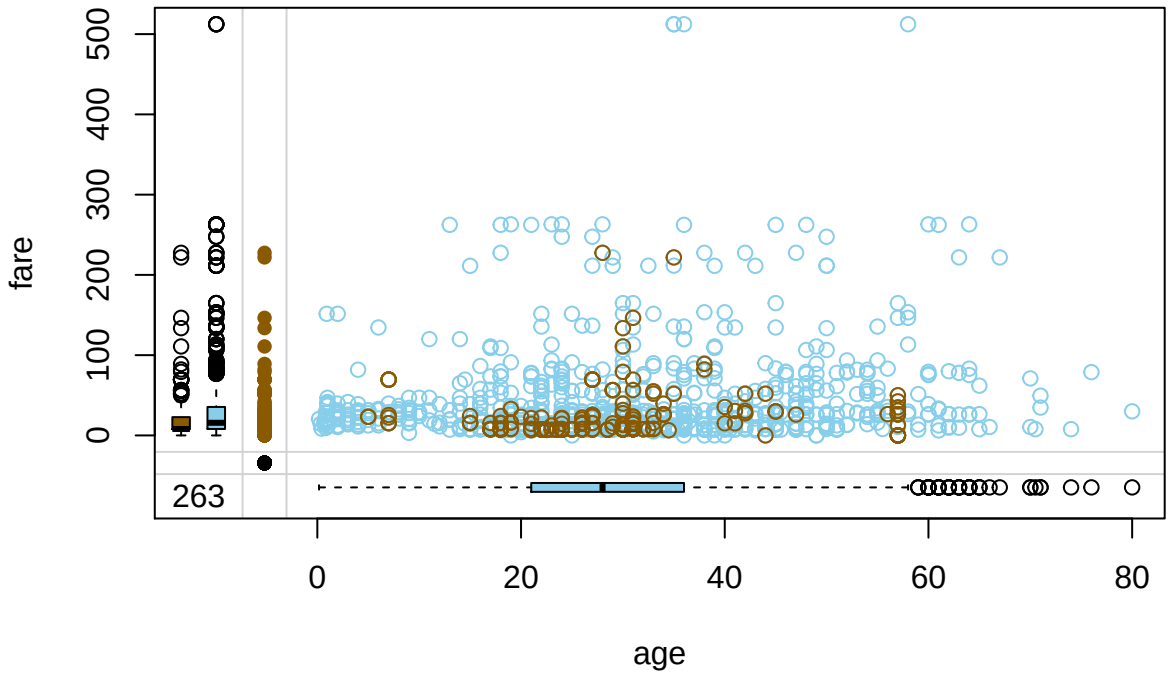
```
hd.imputed = VIM::hotdeck(dat,variable='age',domain_var=c('sex','pclass'))
hd.imputed %>% head
```

```
##   survived pclass                                             name    sex
## 1        1      1                    Allen, Miss. Elisabeth Walton female
## 2        1      1                   Allison, Master. Hudson Trevor   male
## 3        0      1                     Allison, Miss. Helen Loraine female
## 4        0      1             Allison, Mr. Hudson Joshua Creighton   male
## 5        0      1 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6        1      1                              Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked age_imp
## 1 29.0000     0     0  24160 211.3375      B5        S   FALSE
## 2  0.9167     1     2 113781 151.5500 C22 C26        S   FALSE
## 3  2.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 4 30.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 5 25.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 6 48.0000     0     0  19952  26.5500     E12        S   FALSE
```

```
knn.imputed = VIM::kNN(dat,variable='age')
knn.imputed %>% head
```

```
##   survived pclass                                             name    sex
## 1        1      1                    Allen, Miss. Elisabeth Walton female
## 2        1      1                   Allison, Master. Hudson Trevor   male
## 3        0      1                     Allison, Miss. Helen Loraine female
## 4        0      1             Allison, Mr. Hudson Joshua Creighton   male
## 5        0      1 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6        1      1                              Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked age_imp
## 1 29.0000     0     0  24160 211.3375      B5        S   FALSE
## 2  0.9167     1     2 113781 151.5500 C22 C26        S   FALSE
## 3  2.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 4 30.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 5 25.0000     1     2 113781 151.5500 C22 C26        S   FALSE
## 6 48.0000     0     0  19952  26.5500     E12        S   FALSE
```

```
marginplot(knn.imputed[,c('age','fare','age_imp')],delimiter='_imp')
```



Multiple Imputation

Bootstrapping

Aside from implicit vs explicit, there is one other main distinction between imputation techniques: *random* techniques (stochastic regression or hot deck), and *deterministic* techniques (regression imputation or `VIM::kNN`).

In general, while deterministic method are simpler, theoretically they have some drawbacks. Primarily, they don't give you any sense of how much your particular choice of imputed values effects your final model estimates. In some particular cases you can derive standard errors, but it adds a bit of computational baggage.

Random imputation techniques, on the other hand, do not have this drawback. By sampling multiple versions of the same imputed data, you can fit the model to each sample and get an approximate distribution of your model coefficients under the sampling distribution of the **imputed data**.

When this distribution varies a lot, it indicates that your imputation is injecting a lot of uncertainty into the model, whereas when this distribution is fairly consistent then imputation doesn't matter that much:

```
impute.and.fit = function(dat){
  dat.imputed = VIM::hotdeck(dat, variable='age', domain_var = c('sex','pclass'))

  mod.imputed = glm(survived ~ sex + as.factor(pclass) + sibsp + age + fare,
                    data=dat.imputed,
                    family=binomial)

  return(as.numeric(coefficients(mod.imputed)))

}

samps = matrix(0,nrow=10,ncol=7)
for (i in 1:10){
  samps[i,] = impute.and.fit(dat)
```

```
}

apply(samps,2,mean)
```

```
## [1]  3.373130962 -2.559746301 -1.131787970 -2.067492318 -0.325992783
## [6] -0.031148581  0.001928165
```

```
apply(samps,2,sd)
```

```
## [1] 0.156811178 0.010982925 0.050708409 0.067491632 0.009446437 0.003535147
## [7] 0.000132874
```

Now, we do need to be a little careful about defining what we're estimating here. Recall that, for a single imputed dataset $D_{\text{imp}}^{(i)}$, our resulting coefficient estimate $\hat{\beta}^{(i)}$ has some uncertainty due to just the estimation process, which we assess using it's standard error $s.e.(\hat{\beta}^{(i)})$. This is **not** the same thing as the changes in $\hat{\beta}$ due to the imputation, which we measured above.

To distinguish between these concepts we will define the **within-imputation** variance of $\hat{\beta}^{(i)}$, $W_i = s.e.(\hat{\beta}^{(i)})^2$, and the **between-imputation** variance of, $B = \frac{1}{m}\sum_{i=1}^{m}(\hat{\beta}^{(i)} - \bar{\beta})^2$ (where $\beta\beta$ is just the mean of the $\hat{\beta}^{(i)}$.)

Because the imputation process has introduced this extra uncertainty B into our estimate $\hat{\beta}$, we cannot simply use $s.e.(\hat{\beta}_i)$ to eg. perform a hypothesis test. We need to combine W_i and B in an appropriate way to properly represent our uncertainty of $\hat{\beta}$.

The formula for this is actually quite simple:

$$T = (\frac{1}{m}\sum_{i=1}^{m} W_i) + \frac{m+1}{m}B = \bar{W} + \frac{m+1}{m}B$$

This is called the **total variance** of $\bar{\beta}$, and allows us to compute a final standard error for our mean coefficient vector $s.e.(\bar{\beta}) = \sqrt{T}$. For sufficiently large datasets, it can be shown that $\bar{\beta}$ is approximately normal, and so we can, for example, compute confidence intervals using our usual t-statistic.

While it's possible to do this all by hand, in practice it's kind of a pain, so as usual we will instead use a package's workflow, in this case `mice`:

```
vars = c('survived','age','pclass','sex','sibsp')

# we don't want mice to use the survived variable to impute age, so we'll construct a predictor matrix
pred.mat = matrix(1,nrow=length(vars),ncol=length(vars))
pred.mat = pred.mat - diag(length(vars))
pred.mat[1,] = 0
pred.mat[,1] = 0

# By default, mice uses an imputation method called Predictive Mean Matching (PMM)
# The specifics of this imputation strategy are a little complicated, but it's like hot deck in that it
dat.imputed = mice(dat[vars], predictorMatrix=pred.mat)
```

```
##
##  iter imp variable
##   1   1  age
##   1   2  age
##   1   3  age
##   1   4  age
##   1   5  age
##   2   1  age
```

```
##   2   2  age
##   2   3  age
##   2   4  age
##   2   5  age
##   3   1  age
##   3   2  age
##   3   3  age
##   3   4  age
##   3   5  age
##   4   1  age
##   4   2  age
##   4   3  age
##   4   4  age
##   4   5  age
##   5   1  age
##   5   2  age
##   5   3  age
##   5   4  age
##   5   5  age
```

```
## Warning: Number of logged events: 1
```

```
mods.imputed = with(data=dat.imputed, expr = glm(survived ~ age + as.factor(pclass) + sex + sibsp, famil
mod.mi = pool(mods.imputed)
summary(mod.mi)
```

```
##                 term    estimate   std.error  statistic        df      p.value
## 1        (Intercept)  3.58749771 0.337859940  10.618299  170.6363 0.000000e+00
## 2                age -0.03272886 0.006581972  -4.972501  110.8242 2.430427e-06
## 3 as.factor(pclass)2 -1.22163950 0.218727620  -5.585209  619.2461 3.497545e-08
## 4 as.factor(pclass)3 -2.18394545 0.215254926 -10.145856  159.9598 0.000000e+00
## 5            sexmale -2.60036293 0.153425746 -16.948674 1294.6429 0.000000e+00
## 6              sibsp -0.32527939 0.084239066  -3.861384  996.2959 1.200182e-04
```