A quick intro to handling missing data with R

Dr. Simon Caton

22/9/2018

Introduction

Missing data is common, but what are the options for dealing with it? The easiest is to delete parts of the dataset, but this can be problematic especially if the dataset is small. Let's start with a simple example, 1000 random numbers drawn from a standard normal distribution:

```r
x <- rnorm(1000, mean = 0, sd = 1)
```

We know (as this is a standard normal distribution) that the mean should be 0, and the standard deviation 1:

```r
mean(x)
```

```
## [1] 0.05369685
```

```r
sd(x)
```

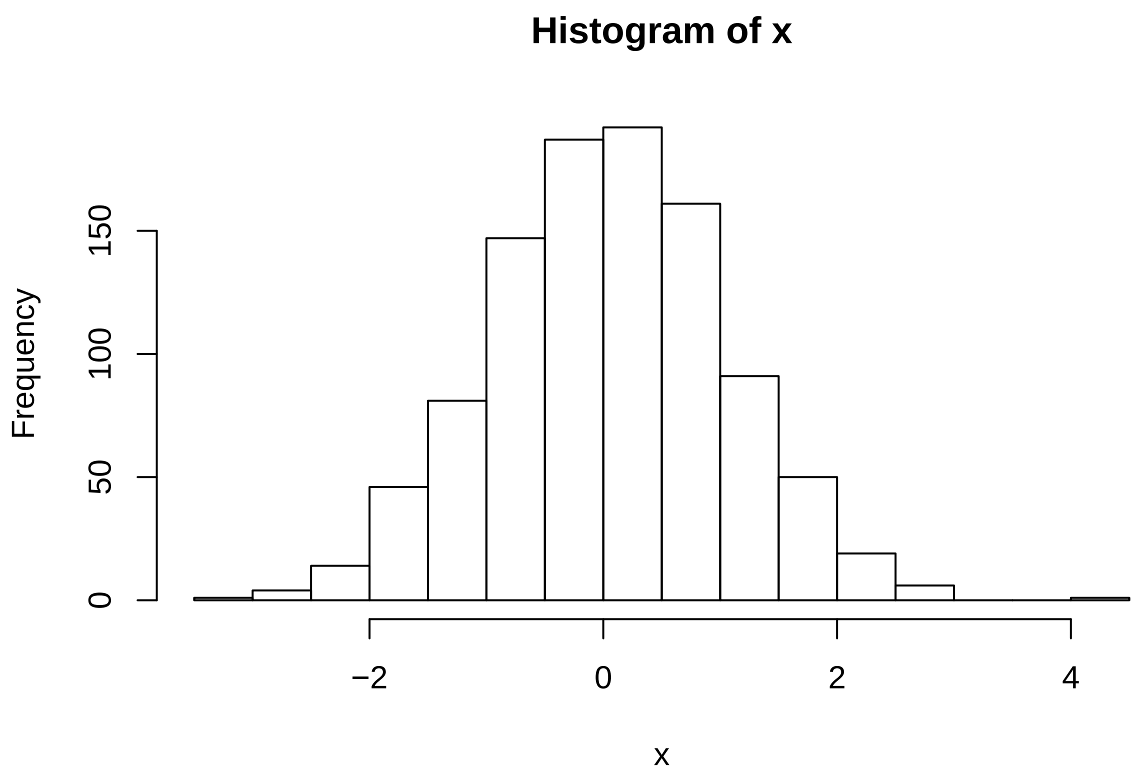
```
## [1] 1.00048
```

It's pretty close. Let's also get a few more descriptive statistics, so that we can compare different methods of data imputation. As we will need to do this a few times, we'll define a function.

```r
descriptives <- function(x) {
  u <- length(unique(x))
  missing <- sum(is.na(x)) / length(x) * 100
  quantiles <- quantile(x, na.rm = T)
  av <- mean(x, na.rm = T)
  stdev <- sd(x, na.rm = T)

  df <- data.frame(u, missing, quantiles[1], quantiles[2],
                   quantiles[3], quantiles[4], quantiles[5], av, stdev)

  names(df) <- c("no. unique", "% missing", "min", "first quartile",
                 "median", "third quartile", "max", "mean", "sd")
  rownames(df) <- NULL
  return(df)
}

df <- descriptives(x)
```

Now, just get the shape of x, and we have a good starting point to see the effects of different strategies.

```r
hist(x)
```



Injecting missing data

So let's inject some missing values. We'll make 10% of the data disappear (you should feel free to come back to this point later and increase this amount!!):

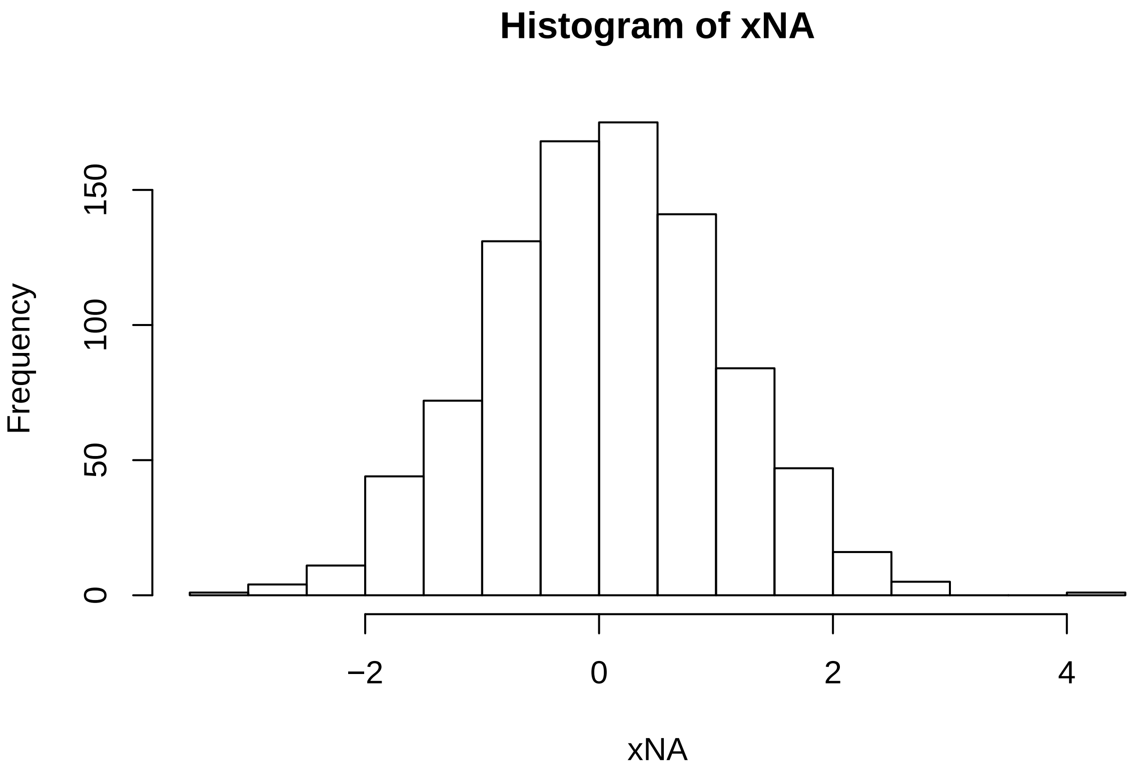
```
index <- sample(length(x), length(x)*.1, replace = F)
xNA <- x
xNA[index] <- NA
df <- rbind(df, descriptives(xNA))
```

So now we have 100 missing values. Let's compare the before and after:

```
rownames(df) <- c("before", "after")
df
```

```
##        no. unique % missing       min first quartile     median
## before       1000         0 -3.004033     -0.6231411 0.05386626
## after         901        10 -3.004033     -0.6095315 0.05633444
##        third quartile      max       mean       sd
## before      0.7560655 4.401797 0.05369685 1.000480
## after       0.7578704 4.401797 0.05532680 1.001399
```

```
hist(xNA)
```



Imputing missing values

There are a few basic strategies when we have only one feature:

1. Take the mean
2. Take the median
3. Take the mode (not applicable in this case, all values are unique)
4. 0 (or some other number)

Let's see what difference this makes:

```
library(kableExtra)

xMean <- xNA
xMedian <- xNA
x0 <- xNA
xMean[is.na(xMean)] <- mean(xMean, na.rm = T)
xMedian[is.na(xMedian)] <- median(xMedian, na.rm = T)
x0[is.na(x0)] <- 0

df <- rbind(df, descriptives(xMean))
df <- rbind(df, descriptives(xMedian))
df <- rbind(df, descriptives(x0))

rownames(df) <- c("x", "10% missing", "take mean", "take median", "take 0")
kable(df[, c(1:5)], caption = "Changes in x via imputation I")
```

```
kable(df[, c(6:dim(df)[2])], caption = "Changes in x via imputation II")
```

That's quite a difference (see changes in mean, median and the quartiles). If we plot these, we can see the

Table 1: Changes in x via imputation I

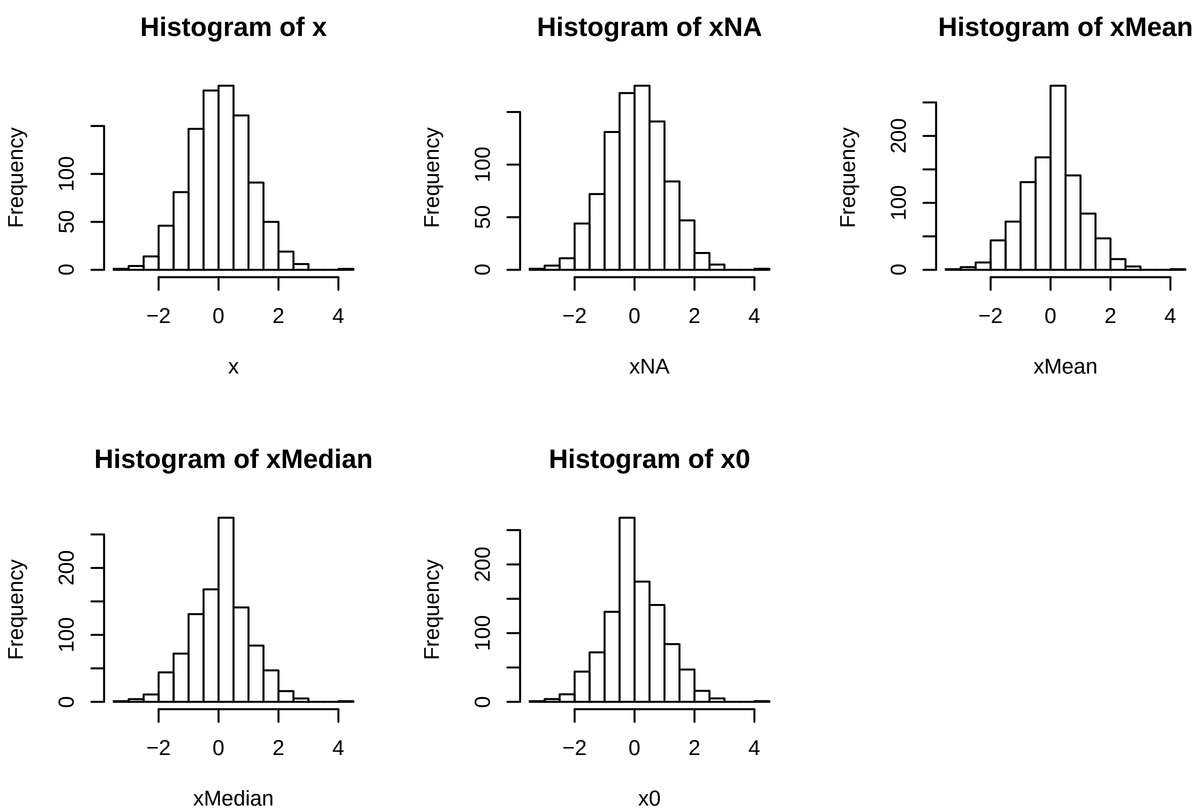
	no. unique	% missing	min	first quartile	median
x	1000	0	-3.004033	-0.6231411	0.0538663
10% missing	901	10	-3.004033	-0.6095315	0.0563344
take mean	901	0	-3.004033	-0.5395501	0.0553268
take median	901	0	-3.004033	-0.5395501	0.0563344
take 0	901	0	-3.004033	-0.5395501	0.0000000

Table 2: Changes in x via imputation II

	third quartile	max	mean	sd
x	0.7560655	4.401797	0.0536969	1.0004804
10% missing	0.7578704	4.401797	0.0553268	1.0013995
take mean	0.6709313	4.401797	0.0553268	0.9499581
take median	0.6709313	4.401797	0.0554276	0.9499582
take 0	0.6709313	4.401797	0.0497941	0.9501032

change in distribution shape more easily.

```
par(mfrow=c(2,3))
hist(x)
hist(xNA)
hist(xMean)
hist(xMedian)
hist(x0)
```



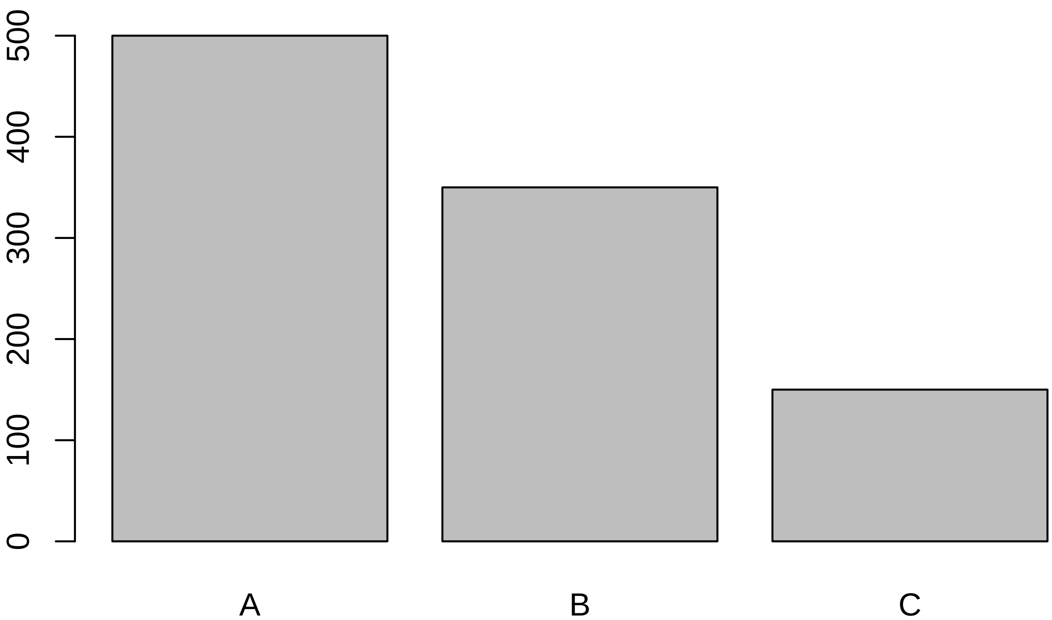
In all cases, you can see that the imputation of missing values, or to be more precise the assumptions underriding the imputation strategy, have on the data: the measure of central tendency is now more frequent. Whilst, it's clear that we probably don't want to remove the missing values, we do need to be aware that imputation does have a specific impact on the data!

Categorical Data

With categorical data, not much changes really, except that methods based on numeric notions of central tendency are not appropriate: the mean / median of categorical data wouldn't make any sense.

Consider the following:

```
y <- factor(c(rep("A", 500), rep("B", 350), rep("C", 150)))
barplot(table(y))
```



At the moment, our data is ordered A, B, C, we could to shuffle it, but it won't really make any difference to our random "NAing" of the data. If we modify our function now for categorical data, we get a similar base line for our imputation strategies.

```
descriptivesCat <- function(x) {
  u <- length(unique(x))
  missing <- sum(is.na(x)) / length(x) * 100
  t <- table(x)
  modeFreq <- max(t)
  mode <- names(t)[t==modeFreq]

  x <- x[x != mode]
  t <- table(x)
  mode2Freq <- max(t)
  mode2 <- names(t)[t==mode2Freq]

  df <- data.frame(u, missing, mode, modeFreq, mode2, mode2Freq)

  names(df) <- c("no. unique", "% missing", "1st mode", "1st mode freq",
                 "2nd mode", "2nd mode freq")
```

```r
  rownames(df) <- NULL
  return(df)
}

df <- descriptivesCat(y)
```

So if we now remove 10% and 30% of the data this time:

```r
index <- sample(length(y), length(y)*.1, replace = F)
yNA10 <- y
yNA10[index] <- NA

df <- rbind(df, descriptivesCat(yNA10))

index <- sample(length(y), length(y)*.3, replace = F)
yNA30 <- y
yNA30[index] <- NA

df <- rbind(df, descriptivesCat(yNA30))
df
```

```
##   no. unique % missing 1st mode 1st mode freq 2nd mode 2nd mode freq
## 1          3         0        A           500        B           350
## 2          4        10        A           445        B           319
## 3          4        30        A           357        B           242
```

So, this time, there aren't really many options for imputation:

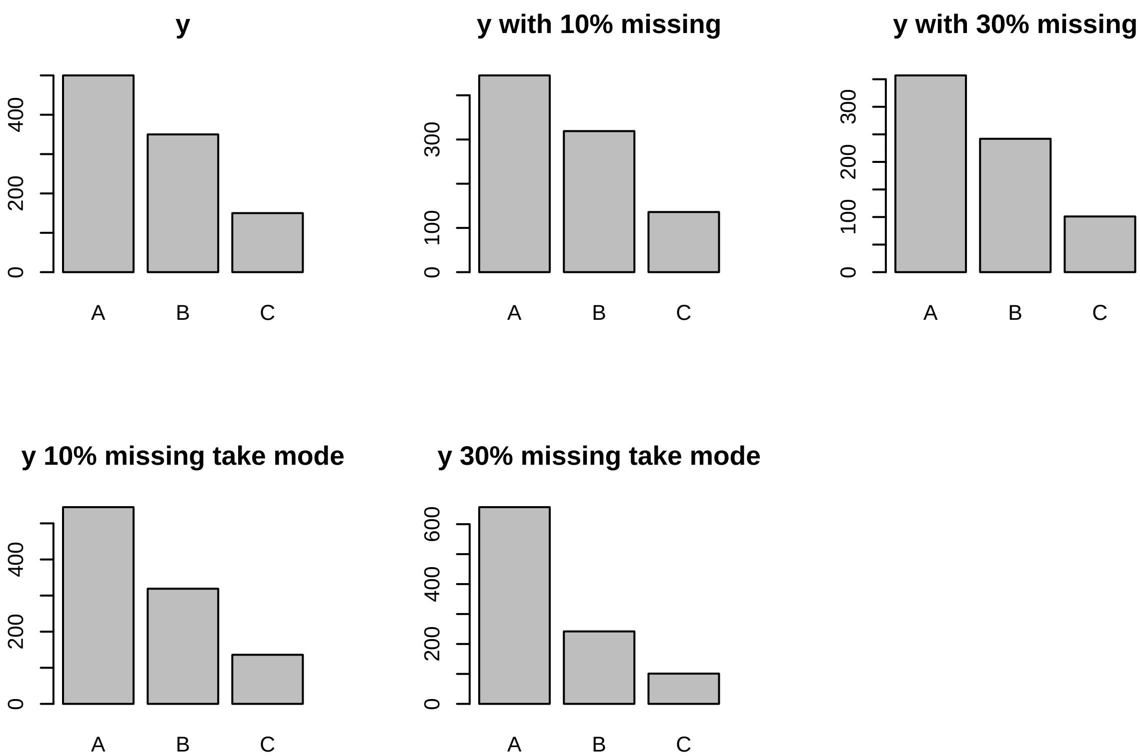
1. take the mode
2. . . . ?

So if we take the mode, what happens?

```r
yNA10mode <- yNA10
yNA30mode <- yNA30

yNA10mode[is.na(yNA10mode)] <- df$`1st mode`[2]
yNA30mode[is.na(yNA30mode)] <- df$`1st mode`[2]

df <- rbind(df, descriptivesCat(yNA10mode))
df <- rbind(df, descriptivesCat(yNA30mode))

rownames(df) <- c("y", "10% missing", "30% missing", "10% take mode", "30% take mode")

par(mfrow=c(2,3))
barplot(table(y), main="y")
barplot(table(yNA10), main="y with 10% missing")
barplot(table(yNA30), main="y with 30% missing")
barplot(table(yNA10mode), main="y 10% missing take mode")
barplot(table(yNA30mode), main="y 30% missing take mode")

kable(df, caption="Changes in y via imputation")
```

Table 3: Changes in y via imputation

	no. unique	% missing	1st mode	1st mode freq	2nd mode	2nd mode freq
y	3	0	A	500	B	350
10% missing	4	10	A	445	B	319
30% missing	4	30	A	357	B	242
10% take mode	3	0	A	545	B	319
30% take mode	3	0	A	657	B	242



As you can see, because the data does not follow a uniform distribution, imputation (via the mode) exacerbates the class imbalance; A becomes more frequent.

Adult dataset

If we use a real dataset what are our options?

```
adult <- read.csv("adult.data", header = F,
                  na.strings = c("", "?"),
                  sep = ",", strip.white = TRUE)

names(adult) <- c(
  "age",
  "workclass",
  "fnlwgt",
  "education",
  "education-num",
  "marital-status",
  "occupation",
```

```r
  "relationship",
  "race",
  "sex",
  "capital-gain",
  "capital-loss",
  "hours-per-week",
  "native-country",
  "earning"
)
```

First we need to get an idea of the volume of missing data, and how it's distributed.

```r
sapply(adult, FUN = function(x) {sum(is.na(x))})
```

```
##            age      workclass         fnlwgt      education  education-num
##              0           1836              0              0              0
## marital-status     occupation   relationship           race            sex
##              0           1843              0              0              0
##   capital-gain   capital-loss hours-per-week native-country        earning
##              0              0              0            583              0
```

To get an idea of the quality of our dataset, we could actually slightly modify our two descriptive functions, as follows:

```r
dataQualityNum <- function(df) {
  n <- sapply(df, function(x) {is.numeric(x)})
  df_numerics <- df[, n]
  instances <- sapply(df_numerics, FUN=function(x) {length(x)})
  missing <- sapply(df_numerics, FUN=function(x) {sum(is.na(x))})
  missing <- missing / instances * 100
  unique <- sapply(df_numerics, FUN=function(x) {length(unique(x))})
  quantiles <- t(sapply(df_numerics, FUN=function(x) {quantile(x)}))
  means <- sapply(df_numerics, FUN=function(x) {mean(x)})
  sds <- sapply(df_numerics, FUN=function(x) {sd(x)})

  df_numeric <- data.frame(Feature=names(df_numerics),
                        Instances=instances,
                        Missing=missing,
                        Cardinality=unique,
                        Min=quantiles[,1],
                        FirstQuartile=quantiles[,2],
                        Median=quantiles[,3],
                        ThirdQuartile=quantiles[,4],
                        Max=quantiles[,5],
                        Mean=means,
                        Stdev=sds)

  rownames(df_numeric) <- NULL

  return(df_numeric)
}
```

And similarly, for our categorical function

```r
dataQualityCat <- function(df) {
  n <- sapply(df, function(x) {is.numeric(x)})
```

```r
df_categoricals <- df[, !n]

instances <- sapply(df_categoricals, FUN=function(x) {length(x)})
missing <- sapply(df_categoricals, FUN=function(x) {sum(is.na(x))})
missing <- missing / instances * 100
unique <- sapply(df_categoricals, FUN=function(x) {length(unique(x))})

modeFreqs <- sapply(df_categoricals, FUN=function(x) {
  t <- table(x)
  modeFreq <- max(t)
  return(modeFreq)
})

modes <- sapply(df_categoricals, FUN=function(x) {
  t <- table(x)
  modeFreq <- max(t)
  mode <- names(t)[t==modeFreq]
  return(mode)
})

modeFreqs2 <- sapply(df_categoricals, FUN=function(x) {
  t <- table(x)
  modeFreq <- max(t)
  mode <- names(t)[t==modeFreq]
  x <- x[x != mode]
  t <- table(x)
  mode2Freq <- max(t)

  return(mode2Freq)
})

modes2 <- sapply(df_categoricals, FUN=function(x) {
  t <- table(x)
  modeFreq <- max(t)
  mode <- names(t)[t==modeFreq]
  x <- x[x != mode]
  t <- table(x)
  mode2Freq <- max(t)
  mode2 <- names(t)[t==mode2Freq]

  return(mode2)
})

df_categorical <- data.frame(Feature=names(df_categoricals),
                       Instances=instances,
                       Missing=missing,
                       Cardinality=unique,
                       FirstMode=modes,
                       FirstModeFreq=modeFreqs,
                       SecondMode=modes2,
                       SecondModeFreq=modeFreqs2)

rownames(df_categorical) <- NULL
```

Table 4: Data Quality Adult: numeric features I

Feature	Instances	Missing	Cardinality	Min	FirstQuartile
age	32561	0	73	17	28
fnlwgt	32561	0	21648	12285	117827
education-num	32561	0	16	1	9
capital-gain	32561	0	119	0	0
capital-loss	32561	0	92	0	0
hours-per-week	32561	0	94	1	40

Table 5: Data Quality Adult: numeric features II

Feature	Median	ThirdQuartile	Max	Mean	Stdev
age	37	48	90	38.58165	13.64043
fnlwgt	178356	237051	1484705	189778.36651	105549.97770
education-num	10	12	16	10.08068	2.57272
capital-gain	0	0	99999	1077.64884	7385.29208
capital-loss	0	0	4356	87.30383	402.96022
hours-per-week	40	45	99	40.43746	12.34743

```
  return(df_categorical)
}
```

To be fair, it would be possible to write the sapply invocations for mode and mode frequency more efficiently, but the code would be harder to understand.

If we were to now call these functions on our Adult dataset, we would get:

```
dq_numeric <- dataQualityNum(adult)
kable(dq_numeric[, 1:6], caption="Data Quality Adult: numeric features I")
```

```
kable(dq_numeric[, c(1, 7:dim(dq_numeric)[2])],
      caption="Data Quality Adult: numeric features II")
```

```
dq_categorical <- dataQualityCat(adult)
kable(dq_categorical[, 1:5], caption="Data Quality Adult: categorical features I")
```

```
kable(dq_categorical[, c(1, 6:dim(dq_categorical)[2])],
      caption="Data Quality Adult: categorical features II")
```

Table 6: Data Quality Adult: categorical features I

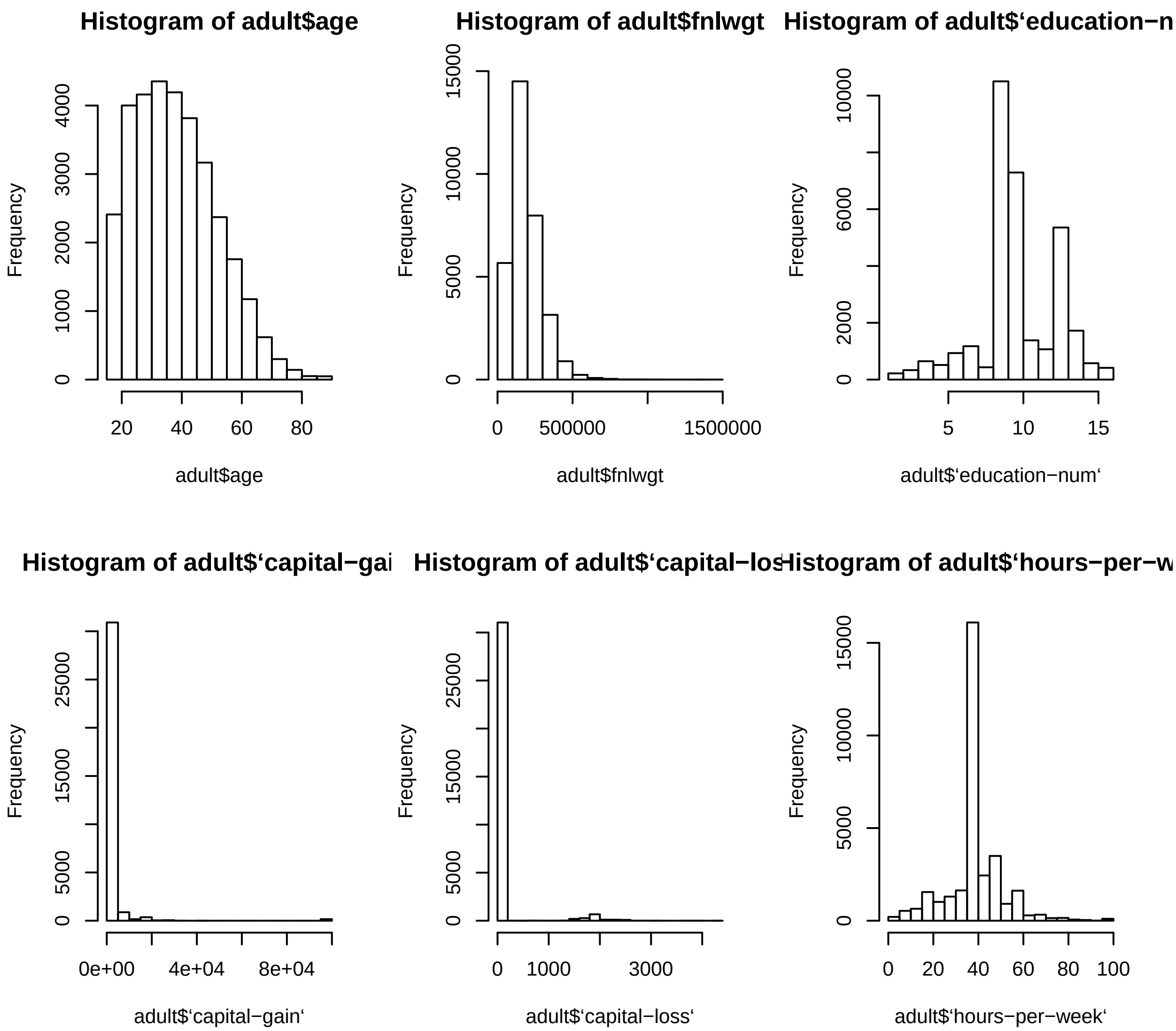
Feature	Instances	Missing	Cardinality	FirstMode
workclass	32561	5.638648	9	Private
education	32561	0.000000	16	HS-grad
marital-status	32561	0.000000	7	Married-civ-spouse
occupation	32561	5.660146	15	Prof-specialty
relationship	32561	0.000000	6	Husband
race	32561	0.000000	5	White
sex	32561	0.000000	2	Male
native-country	32561	1.790486	42	United-States
earning	32561	0.000000	2	<=50K

Table 7: Data Quality Adult: categorical features II

Feature	FirstModeFreq	SecondMode	SecondModeFreq
workclass	22696	Self-emp-not-inc	2541
education	10501	Some-college	7291
marital-status	14976	Never-married	10683
occupation	4140	Craft-repair	4099
relationship	13193	Not-in-family	8305
race	27816	Black	3124
sex	21790	Female	10771
native-country	29170	Mexico	643
earning	24720	>50K	7841

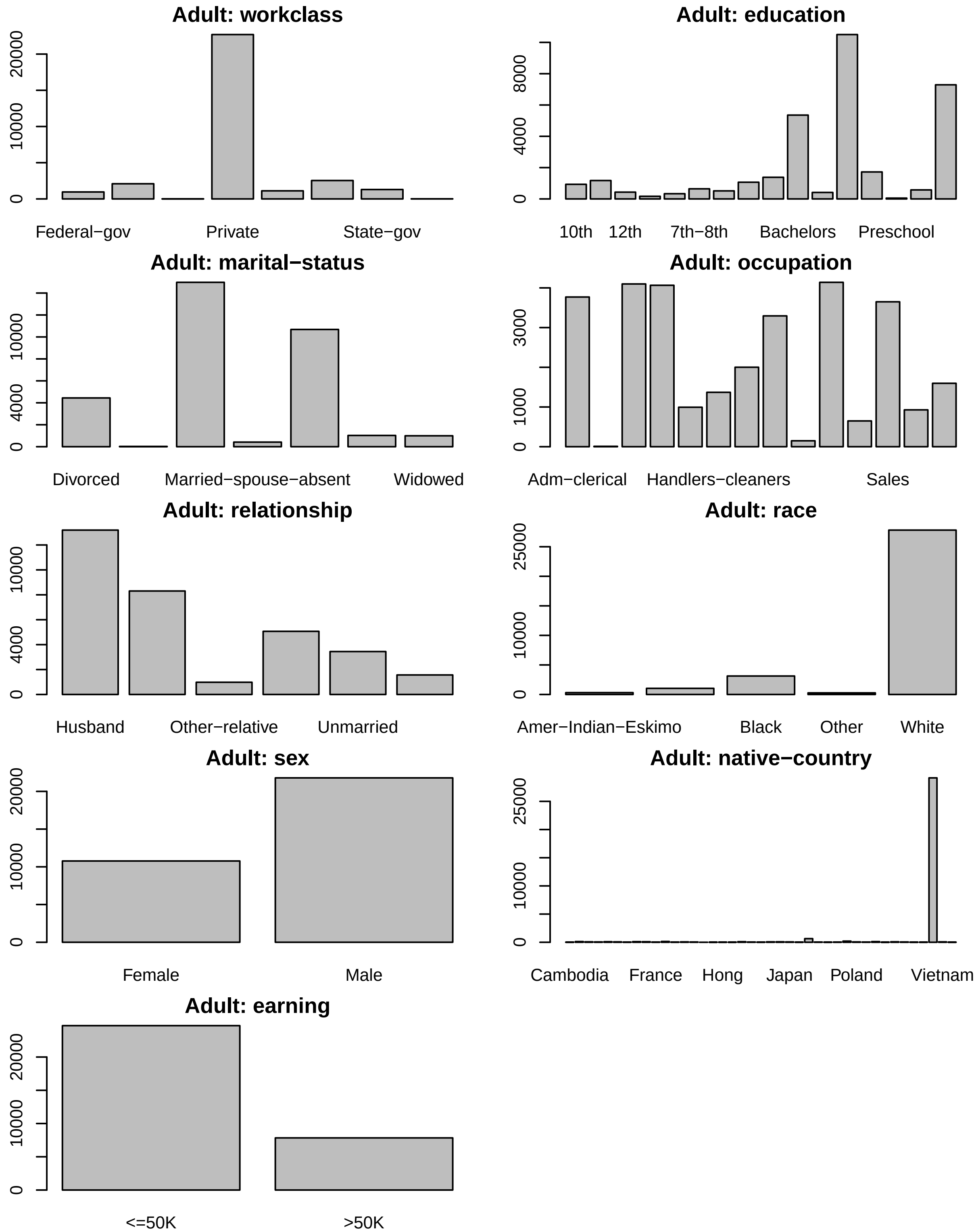
For completeness, we should also plot their distributions to get an idea of their shape:

```
par(mfrow=c(2, 3))
hist(adult$age)
hist(adult$fnlwgt)
hist(adult$`education-num`)
hist(adult$`capital-gain`)
hist(adult$`capital-loss`)
hist(adult$`hours-per-week`)
```



I'll be honest, I didn't feel like typing all the names of the columns... so here's a way to loop over the features to generate graphs:

```
n <- sapply(adult, function(x) {is.numeric(x)})
df_categoricals <- adult[, !n]
par(mar=c(2,2,2,2), mfrow=c((dim(df_categoricals)[2] %/% 2)+1,2))
for (i in 1:length(df_categoricals)) {
  barplot(table(adult[, names(df_categoricals)[i]]),
          main=paste("Adult:", names(df_categoricals[i])))
}
```



None of the numeric features have any missing values, only the following categorical features:

- workclass; cardinality 9
- occupation; cardinality 15 and
- native-country; cardinality 42

Given that workclass and native-country are so extremely dominated by one class, there may be some merit in choosing the mode for the imputation strategy:

```
wc <- is.na(adult$workclass)
nc <- is.na(adult$`native-country`)

adult$workclass[is.na(adult$workclass)] <-
  dq_categorical[dq_categorical$Feature=="workclass", "FirstMode"]

adult$`native-country`[is.na(adult$`native-country`)] <-
  dq_categorical[dq_categorical$Feature=="native-country", "FirstMode"]

sapply(adult, FUN=function(x) {sum(is.na(x))})
```

```
##            age      workclass         fnlwgt      education  education-num
##              0              0              0              0              0
## marital-status     occupation   relationship           race            sex
##              0           1843              0              0              0
##   capital-gain   capital-loss hours-per-week native-country        earning
##              0              0              0              0              0
```

```
wasMissing <- wc | nc
```

wasMissing is a vector containing rows that had a missing value in either workclass OR (denoted by |) native-country. As we have imputed these values, we wouldn't normally want them biasing the imputation of occupation. However, as you can see, most rows where one of these values is missing, occupation is also missing:

```
adultNoMissing <- adult[!wasMissing, ]
sapply(adultNoMissing, FUN=function(x){sum(is.na(x))})
```

```
##            age      workclass         fnlwgt      education  education-num
##              0              0              0              0              0
## marital-status     occupation   relationship           race            sex
##              0              7              0              0              0
##   capital-gain   capital-loss hours-per-week native-country        earning
##              0              0              0              0              0
```

For occupation, we going to use a machine learning method for the imputation process. This process will build a predictive model to predict missing values in occupation, using values in other features. For now, this is sufficient information, more details on exactly what is happening here will arise in your data mining module (we're using a random forest – an ensemble-based decision tree method). Do keep in mind, that an ML-approach will be slow and computationally expensive if you are trying to impute values across multiple columns, when the dataset is large, or the cardinality of the feature(s) is high. In an attempt to mitigate any biases from the two other columns we imputed earlier, we're going to tell *mice* to ignore those features.

```
library(mice)
start <- Sys.time()
mice_mod <- mice(adult[, !names(adult) %in%
     c('workclass', 'native-country')], method='rf')
# Complete the missing values
mice_output <- complete(mice_mod)
stop <- Sys.time()
```

To give an idea of runtime (this was one column, and 1843 missing instances) the imputation required:

```
stop - start
```

```
## Time difference of 10.74753 mins
```

Table 8: Data Quality Adult: categorical features imputed I

	Feature	Instances	Missing	Cardinality	FirstMode
1	workclass	32561	5.638648	9	Private
4	occupation	32561	5.660146	15	Prof-specialty
8	native-country	32561	1.790486	42	United-States
3	occupation	32561	0.000000	14	Craft-repair
81	workclass	32561	0.000000	8	Private
9	native-country	32561	0.000000	41	United-States

Table 9: Data Quality Adult: categorical features imputed II

	Feature	FirstModeFreq	SecondMode	SecondModeFreq
1	workclass	22696	Self-emp-not-inc	2541
4	occupation	4140	Craft-repair	4099
8	native-country	29170	Mexico	643
3	occupation	4341	Prof-specialty	4334
81	workclass	24532	Self-emp-not-inc	2541
9	native-country	29753	Mexico	643

If we now repeat the data quality table for categorical data, we can see what impact(s) the imputation has had, but look only at the affected features (to save space), we also need to add the two columns we excluded back in.

```
mice_output$workclass <- adult$workclass
mice_output$`native-country` <- adult$`native-country`

dq_categorical2 <- dataQualityCat(mice_output)

dq_compare <- rbind(dq_categorical[dq_categorical$Feature %in%
  c("workclass", "occupation", "native-country"), ],
  dq_categorical2[dq_categorical2$Feature %in%
  c("workclass", "occupation", "native-country"), ])

kable(dq_compare[, 1:5], caption="Data Quality Adult: categorical features imputed I")
```

```
kable(dq_compare[, c(1, 6:dim(dq_compare)[2])],
      caption="Data Quality Adult: categorical features imputed II")
```

As we can see, where we took the mode (workclass and native-country), that value has just increased. However, for the ML approach (occupation) the first and second mode have switched. Now, we have no ground truth to determine whether this is accurate or not. We could (of course) check the accuracy (as well as other performance measures) of the ML model, but that's it… but this is the challenge of imputation, we never truely know if we've got it right.

Summary

There are several options available for handling missing values:

1. take the mean (if numeric)
2. take the median (if numeric)
3. take the mode (if categorical)
4. use machine learning (works for both)

5. remove the affected rows (via for example adult <- na.omit(adult))
6. remove the affected columns

Each of these options has its advantages and disadvantages. Taking a measure of central tendancy (mean / median / mode) will exaggerate that part of the distibution: the slope of the histogram will get steeper around that central tendency measure. Or in the case of categorical data, the class imbalance will get slightly more pronouced. For an ML approach, we fit a model to the data, and draw estimate values from that model for each instance of a missing feature. We need to be aware of potential under- as well as over-fitting of the data. When removing data, this is the cleanist of all the approaches: there are no assumptions, but it is also destructive; throwing data away is never a good thing.

Ultimately, whenever we are doing any form of analysis, we need to check the robustness of our approach(es) to missing values adjusting our conclusions accordingly. Or in plain English: run with and without your approach(es) in place and adjust your conclusion(s) to take into account the observed deviations (best vs. worst) performance as a sort of confidence interval.