IDA Assignment 3

Callum MacAskill

21/12/2020

Document knitted from an Rmarkdown file. All input code included inline, but if needed the original can be found here.

Question 1

1a

To get the percentage of rows with NA's, first we check which rows have NAs, then divide by the number of total rows in the data frame, before multiplying by 100 to get to percentages.

```
data = nhanes
#Row checking is done by applying the anyNA function to each row
#This will return true if a row has an NA in any column or false otherwise
#We then sum these trues and divide by the length of the dataframe
(sum(apply(data, 1, function(x) anyNA(x)))/dim(data)[1])*100
```

```
## [1] 48
```

It's 48% of rows that have NAs.

1b

```
#First impute the data using mices built in defaults, with seed 1
impdata = mice(data, seed = 1, printFlag = FALSE)
#Then create a linear model using mice's with command
lmfit = with(impdata, lm(bmi ~ hyp+ age + chl))
#Finally pool and display resaults, again using built in mice command
poollmfit = pool(lmfit)
poollmfit
```

```
## Class: mipo    m = 5
##          term m    estimate         ubar            b            t dfcom
## 1 (Intercept) 5 19.61789252 10.588884721 0.8662133972 11.628340797    21
## 2         hyp 5  2.19701748  2.886391704 1.2976511612  4.443573098    21
## 3         age 5 -3.55287155  0.744810536 1.3585594461  2.375081872    21
## 4         chl 5  0.05378081  0.000287288 0.0001046083  0.000412818    21
##          df        riv     lambda       fmi
## 1 16.936189 0.09816483 0.08938989 0.1807424
## 2  9.035494 0.53949067 0.35043452 0.4583762
## 3  3.528053 2.18884032 0.68640637 0.7824821
## 4 10.228828 0.43694808 0.30408063 0.4092932
```

Our output `poollmfit` is of class mipo, where the the level of affect from missingness is given by the size of lambda (ref). Here we can see the highest value, and so the variable most effected by missingness in our data,

is age.

1c

First rerun five times with the new seeds

```
impdata2 = mice(data, seed = 2, printFlag = FALSE)
lmfit2 = with(impdata2, lm(bmi ~ hyp+ age + chl))
poollmfit2 = pool(lmfit2)

impdata3 = mice(data, seed = 3, printFlag = FALSE)
lmfit3 = with(impdata3, lm(bmi ~ hyp+ age + chl))
poollmfit3 = pool(lmfit3)

impdata4 = mice(data, seed = 4, printFlag = FALSE)
lmfit4 = with(impdata4, lm(bmi ~ hyp+ age + chl))
poollmfit4 = pool(lmfit4)

impdata5 = mice(data, seed = 5, printFlag = FALSE)
lmfit5 = with(impdata5, lm(bmi ~ hyp+ age + chl))
poollmfit5 = pool(lmfit5)

impdata6 = mice(data, seed = 6, printFlag = FALSE)
lmfit6 = with(impdata6, lm(bmi ~ hyp+ age + chl))
poollmfit6 = pool(lmfit6)
```

We've repeated everything as in Q1b, just with new seeds. Now lets compare results.

```
#The subsetting below pulls out only the relevant columns, term and lambda
poollmfit2[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.4144454
## 2         hyp 0.1430995
## 3         age 0.4033924
## 4         chl 0.2959966
```

```
poollmfit3[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.2772900
## 2         hyp 0.4101152
## 3         age 0.5895051
## 4         chl 0.5621346
```

```
poollmfit4[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.1315114
## 2         hyp 0.1961083
## 3         age 0.2189333
## 4         chl 0.3305334
```

```
poollmfit5[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.4855733
## 2         hyp 0.5942866
```

```
## 3         age 0.4511896
## 4         chl 0.2346065
```

```
poollmfit6[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.4168136
## 2         hyp 0.2960364
## 3         age 0.6549523
## 4         chl 0.5196295
```

We can see that although age has the highest lambda most often (in seeds 2,3 and 6, not counting the intercept), that in seed 3 cholesterol is most affected by missingness, as is hypertension in seed 5.

1d

Now we repeat the analysis again with a higher M (m = 100)

```
#The previous 4 lines of code for each seed can be condensed down to 1
pool(with(mice(data, seed = 2, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.1882474
## 2         hyp 0.2825108
## 3         age 0.4031077
## 4         chl 0.2939693
```

```
pool(with(mice(data, seed = 3, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.2199607
## 2         hyp 0.2425105
## 3         age 0.3093072
## 4         chl 0.3281911
```

```
pool(with(mice(data, seed = 4, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.2144722
## 2         hyp 0.2565132
## 3         age 0.3943223
## 4         chl 0.2835232
```

```
pool(with(mice(data, seed = 5, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.2294356
## 2         hyp 0.2893046
## 3         age 0.3322570
## 4         chl 0.2461956
```

```
pool(with(mice(data, seed = 6, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))[[3]][c(1,10)]
```

```
##          term    lambda
## 1 (Intercept) 0.2472607
## 2         hyp 0.2860700
## 3         age 0.4430300
## 4         chl 0.3113085
```

This would seem to be a good idea. A higher m, meaning more database copies created and imputed before being pooled back together, would reduce the effect of random chance through a larger number of receptions. Given that in q1c we can see the lambda for age has a range of (0.219, 0.654), a difference of over 300%, this would definitely seem to be an area we can improve on. Now, when repeating for m = 100, we can see our lambda for age now has a range of (0.309, 0.443), much more consistent results.

The pattern does not just apply to lambda for age. See below the summary of resulst for default m and m = 100, with seed 2.

```
#Recalculate and name the m=100 dataframe for seed 2
m100 = pool(with(mice(data, seed = 2, printFlag = FALSE, m = 100), lm(bmi ~ hyp+ age + chl)))

#First display with default m
summary(poollmfit2)
```

```
##          term   estimate  std.error statistic        df     p.value
## 1 (Intercept) 19.9464142 4.36349993  4.571196  7.595481 0.002083932
## 2         hyp  1.5304762 2.01529855  0.759429 15.210844 0.459204412
## 3         age -4.0615093 1.27092690 -3.195706  7.827560 0.013061972
## 4         chl  0.0628349 0.02215369  2.836317 10.450088 0.016956575
```

```
#Then after display with the increased m
summary(m100)
```

```
##          term   estimate  std.error  statistic       df      p.value
## 1 (Intercept) 20.4677977 3.63913732  5.6243543 15.53932 4.241712e-05
## 2         hyp  1.7199145 2.14123878  0.8032334 13.65957 4.355947e-01
## 3         age -3.6337887 1.26258958 -2.8780443 11.27749 1.468267e-02
## 4         chl  0.0535212 0.02123996  2.5198358 13.43174 2.511358e-02
```

We can see that the standard error is down for all values except cholesterol (where it has seen a slight increase). This would suggest we get a more a accurate estimation from m = 100, however in a real life setting more tests would need to be carried out (change of seeds, m, ect).

Question 2

If we're investigating the effects of acknowledging or not parameter uncertainty on confidence intervals then the first step is to impute the missing values, using our given methods (stochastic regression imputation and bootstrap variant), calculate and pool our results. Here we are only interested in the results for β_1.

The second step would then be to check if the correct value of our parameter is included in the confidence interval, the correct value being $\beta_1 = 3$. The number of times the correct value (out of 100 possible) was then counted, and is displayed below.

```
#First load the data
load('dataex2.Rdata')
#And create a 100x2 matrix to store our boolean interval values in
conf_mat = matrix(0L, nrow = 100, ncol = 2)
#Ttart a for loop to calculate the intervals for all datasets
for (i in 1:100) {
  #Multiple imputation steps 1-4 condensed into one line, along with taking the summary of the informat
  sto = as.matrix(summary(pool(with(mice(dataex2[,,i], m = 20, seed = 1, printFlag = FALSE, method = "n
  boot = as.matrix(summary(pool(with(mice(dataex2[,,i], m = 20, seed = 1, printFlag = FALSE, method = "
  #We then check if our value of beta is between the intervals, and record as true or false
  #Note as it's being stored in a matrix, true or false becomes 1 or 0
  conf_mat[i,1] = (sto[1]<=3 & sto[2]>=3)
  conf_mat[i,2] = (boot[1]<=3 & boot[2]>=3)
}
#Sum each column to give the required totals
list(sum(conf_mat[,1]), sum(conf_mat[,2]))
```

```
## [[1]]
## [1] 88
##
## [[2]]
## [1] 95
```

In the list above, entry one is the number of times our confidence interval included the correct value using the stochastics regression imputation, 88 times. The second for bootstrapping, 95 times.

Given we were calculating the 95% confidence interval we would expect the true value of β_1 to be included in the interval 95% of the time (at least over a large enough sample), as we can see with the bootstrapping method.

We also see the expected results of using improper multiple imputation, namely less accurate confidence intervals. This is expected as a consequence of using stochastic regression imputation, in which after imputing each missing observation we treat it the same as fully observed variable. Although both methods may converge to the same expected value, by treating imputed variables as observed we fail to fully account for the variation in the missing variables, something the bootstrapping method does. SRI on the other hand displays narrower confidence, and ultimately less accurate, confidence intervals, hence improper multiple imputation should be avoided.

Question 3

First, what does it look like *compute predicted values then pool* using Rubens rule? Assume we are using a linear model of n terms, with m imputations, and that $\beta_n^{(m)}$ is the n th beta term in the m th imputed database. Then our linear model looks something like this

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M} \hat{y}^{(m)}$$

where

$$\hat{y}^{(m)} = \hat{\beta}_0^{(m)} + \hat{\beta}_1^{\ (m)} x_1 + ... + \hat{\beta}_n^{(m)} x_n + \epsilon$$

That is, we calculate out values for beta in each m database, then pool.

On the other hand if we pool regression coefficients *first*, then Rubens rule, we get our individual βs as

$$\hat{\beta}_0 = \frac{1}{M}\sum_{m=1}^{M} \hat{\beta}_0^{(m)} .....\hat{\beta}_n = \frac{1}{M}\sum_{m=1}^{M} \hat{\beta}_n^{(m)}$$

That is, we pool our calculated value for each beta, then take the average using Ruben's rule. Here the betas give the coefficients for the equivalent x.

Using the above equations, our linear regression model is then given by

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + ...\hat{\beta}_n x_n + \epsilon$$

But if we expand our first set of equations we can clearly see these are equivalent

$$\hat{y} = \frac{1}{M}(\sum_{m=1}^{M} \hat{\beta}_0^{(m)} + \hat{\beta}_1^{\ (m)} x_1 + ... + \hat{\beta}_n^{(m)} x_n + \epsilon) \hat{y} = \frac{1}{M}\sum_{m=1}^{M} \hat{\beta}_0^{(m)} x_0 + \frac{1}{M}\sum_{m=1}^{M} \hat{\beta}_1^{(m)} x_1 + ... + \frac{1}{M}\sum_{m=1}^{M} \hat{\beta}_n^{(m)} x_n + \frac{1}{M}\sum_{m=1}^{M} \epsilon \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 -$$

That is, by splitting up our first equation we can clearly see the value of each optimised beta that makes up $\hat{y}$, and both case 1 and 2 are equivalent.

Question 4

4a

First we impute the missing values of y, x_1, then calculate our coefficients of

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon$$

Code below.

```
load('dataex4.Rdata')
#Again steps 1-4 + subsetting condensed to one line, fast to code but possibly harder to read
summary(pool(with(mice(dataex4, seed = 1, m =50, printFlag = F), lm (y ~ x1 + x2 + x1*x2))), conf.int =
```

```
##          term  estimate    2.5 %    97.5 %
## 1 (Intercept) 1.5929831 1.404501 1.7814655
## 2          x1 1.4112333 1.219397 1.6030697
## 3          x2 1.9658191 1.860657 2.0709812
## 4       x1:x2 0.7550367 0.642302 0.8677715
```

We can see our estimates from the *impute, then transform* method above. Note the correct values for $\beta_{1,2,3,4}$ are 1.5, 1, 2 and 1 respectively. This method does not seem to have worked well however, as the correct values for $\beta_{2,4}$ are not included in the confidence intervals for these values.

4b

Let's try and improve on this result. Next we will calculate $x_1 x_2$ and append this value as a new column to out dataframe, and see if this helps our model. Note as x_1 had missing values, $x_1 x_2$ will also missing values. By predicting the value of β_4 without imputing values for $x_1 x_2$ ourselves, we are doing *passive imputation*.

```
load('dataex4.Rdata')
#Create a new column, x1*x2
dataex4 = mutate(dataex4, x1x2 = x1*x2)
#Calculate the new predictor matrix with our new column, set maxit to 1 as we only need the predictor m
pred = mice(dataex4, seed = 1, m =50, printFlag = F, maxit = 1)$pred
#Set the values of the predictor matrix to 0 for this new column, this means the new value will not be
pred[,4]=0
pred[4,]=0
#Then complete the usual analysis
summary(pool(with(mice(mutate(dataex4, x1x1 = x1*x2), seed = 1, m =50, printFlag = F, predictorMatrix =
```

```
##          term  estimate     2.5 %    97.5 %
## 1 (Intercept) 1.5222144 1.3478381 1.6965906
## 2          x1 1.2438490 1.0615724 1.4261256
## 3          x2 2.0024539 1.9059183 2.0989894
## 4        x1x2 0.8656623 0.7687889 0.9625358
```

Unfortunately the correct values for $\beta_{2,4}$ are still not included in the confidence intervals for their values, although we have seen an improvement in the accuracy of both over the *impute, then transform* method.

4c

Finally, lets try and impute the values for $x_1 x_2$ also, and see if our results improve any further.

```
load('dataex4.Rdata')
dataex4 = mutate(dataex4, x1x2 = x1*x2)
summary(pool(with(mice(mutate(dataex4, x1x1 = x1*x2), seed = 1, m =50, printFlag = F), lm (y ~ x1 + x2 +
```

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

```
##          term estimate     2.5 %   97.5 %
## 1 (Intercept) 1.499714 1.3452011 1.654227
## 2          x1 1.003930 0.8414967 1.166363
## 3          x2 2.026180 1.9398113 2.112548
## 4        x1x2 1.017793 0.9303479 1.105238
```

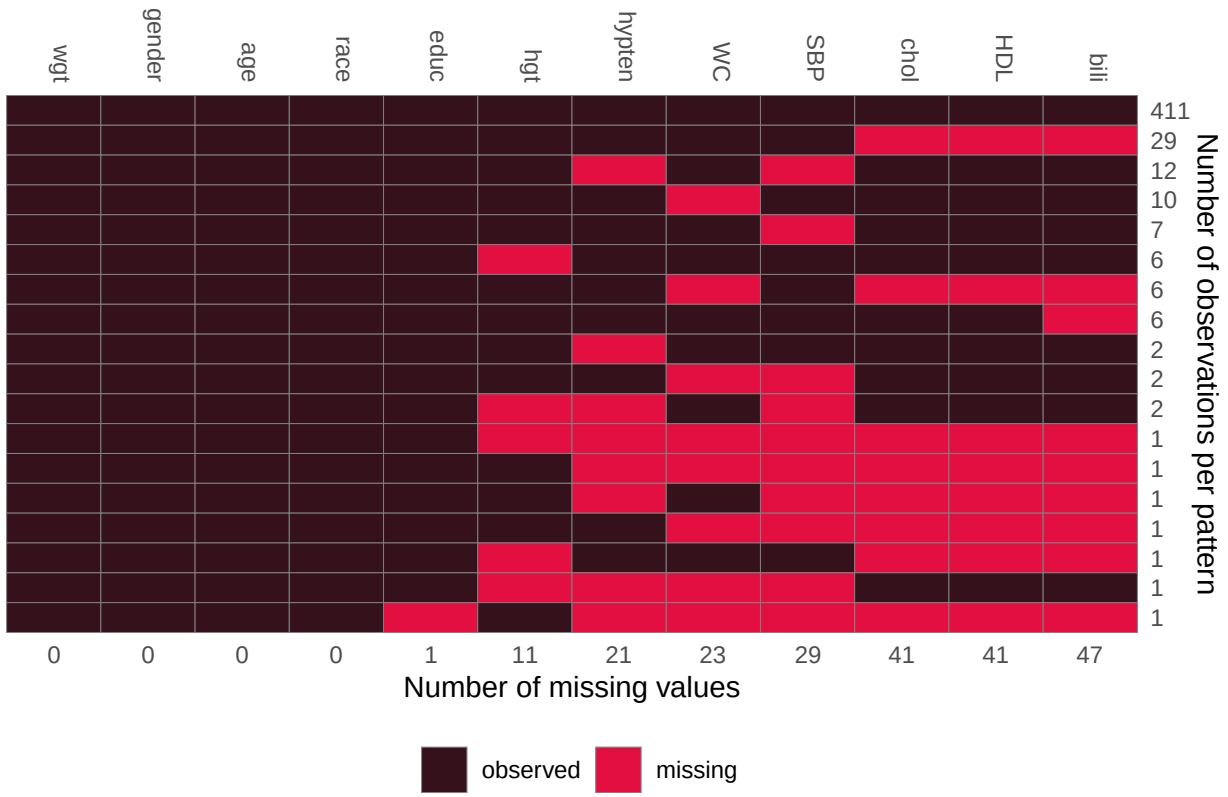
Now we have our strongest model. All values for the β's are in their confidence intervals, and the point estimates are within 3% of the true value for each. Why does treating x_1x_2 as *just another variable* improve our results so much?

In each of our three attempts at this model, we have supplied more information to the lm function at each step, the function which attempts to fit a given model to the data. By giving lm this additional information (not just more values to create a model from, but values with a correct variance and means) we inevitably allow it to create a better model.

Question 5

Before imputing our data and creating our model we should make some common-sense checks to our imputation. First lets take a look at what's missing from the data

```
load('NHANES2.Rdata')
data2 = NHANES2
#Use the JointAI package missing data visualisation function
md_pattern(data2, pattern = FALSE, color =c('#34111b','#e30f41'))
```



Note that out of 500 observations, 411 are fully observed, and in particular only 32 are missing information required for the model. It's fair to say then that we are missing a relatively small amount of data, and therefor do not have to set our value. For the purposes of this question m = 25 should be reasonable.

We should also take a look at how mice will impute this data, to ensure it is using sensible values

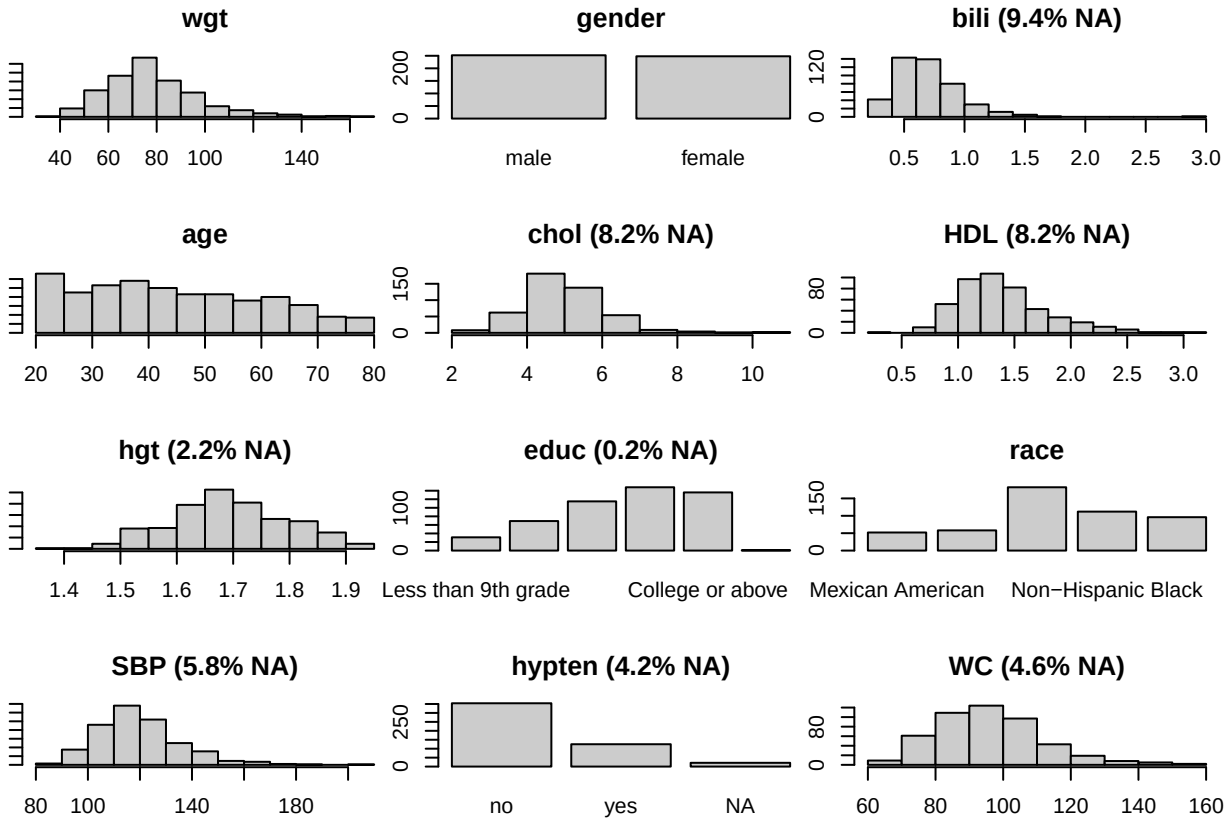
```
#Dry run mice with 0 it to see which methods it will use
M = mice(data2, maxit = 0)
M$method
```

```
##       wgt    gender      bili       age      chol       HDL       hgt      educ
##        ""        ""     "pmm"        ""     "pmm"     "pmm"     "pmm"    "polr"
##      race       SBP    hypten        WC
##        ""     "pmm" "logreg"     "pmm"
```

We can see mice will mainly use predictive mean matching to impute values any numerical values which need imputing, ensuring values in the same range as the observations, meaning no ranges need to be set.

We could change the predictive models used if we wished, for instance height is normally distributed in the population, and indeed a visualisation of the data backs this up.

```
par(mar = c(3,1,3,1))
plot_all(data2)
```



In fact we could argue that weight, cholesterol, HDL, height, SBP and waist circumference are all normally distributed, in which case we could switch our model from pmm to norm. However, whilst these values may be normally distributed in the population as whole, we should note that we do not know that these observations come from a random sample of the population. As discussed above, using norm instead of pmm could also lead to unlikely or impossible imputations, and over a large enough sample pmm should have enough observations to be able to give a full coverage of possible values. It could also be pointed out that several of the continous variables do seem to have a slight skew, again this would favour pmm for imputation.

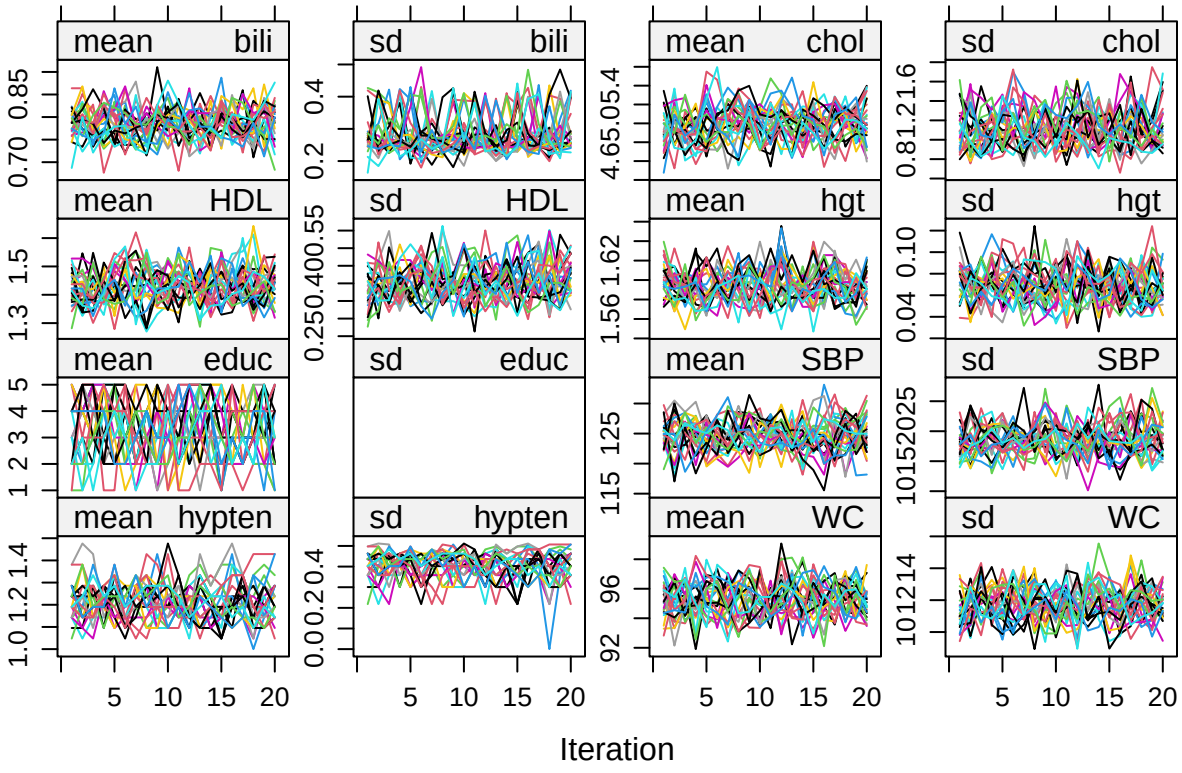
Whilst it is not something we can check for here, we should also note that one of the main requirements of mice is that our missing data is MAR. Here we do not know the missingness mechanism, we would need to discuss it with the data collector, or some other relevant expert. If this data were to be MNAR it would potentially undermine our model and any other conclusions based on our imputations.

Let's assume however that our data is MAR, now we can start imputation and checking for issues.

```
#Impute the data
impdata = mice(data2, m = 25, maxit = 20, seed = 1, printFlag = F)
#Check for issues, should be null
impdata$loggedEvents
```

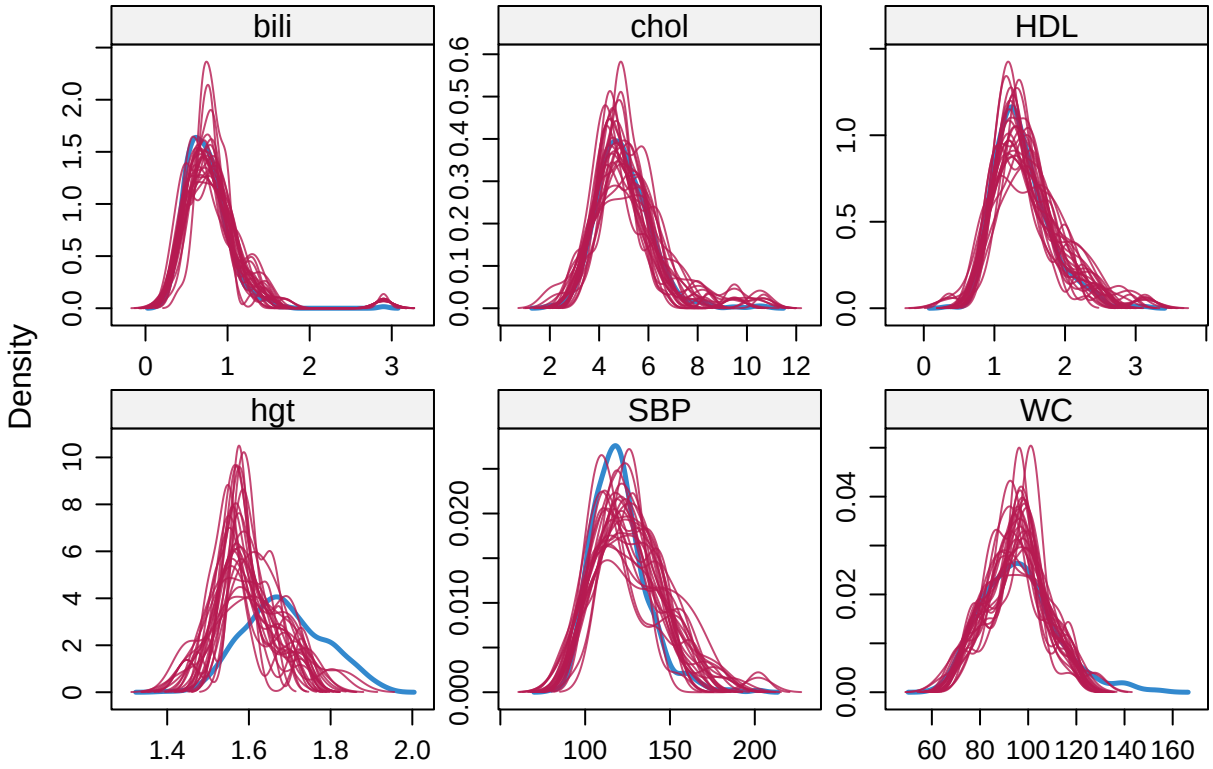
```
## NULL
```

```
#Plot convergence
plot(impdata, layout= c(4,4))
```



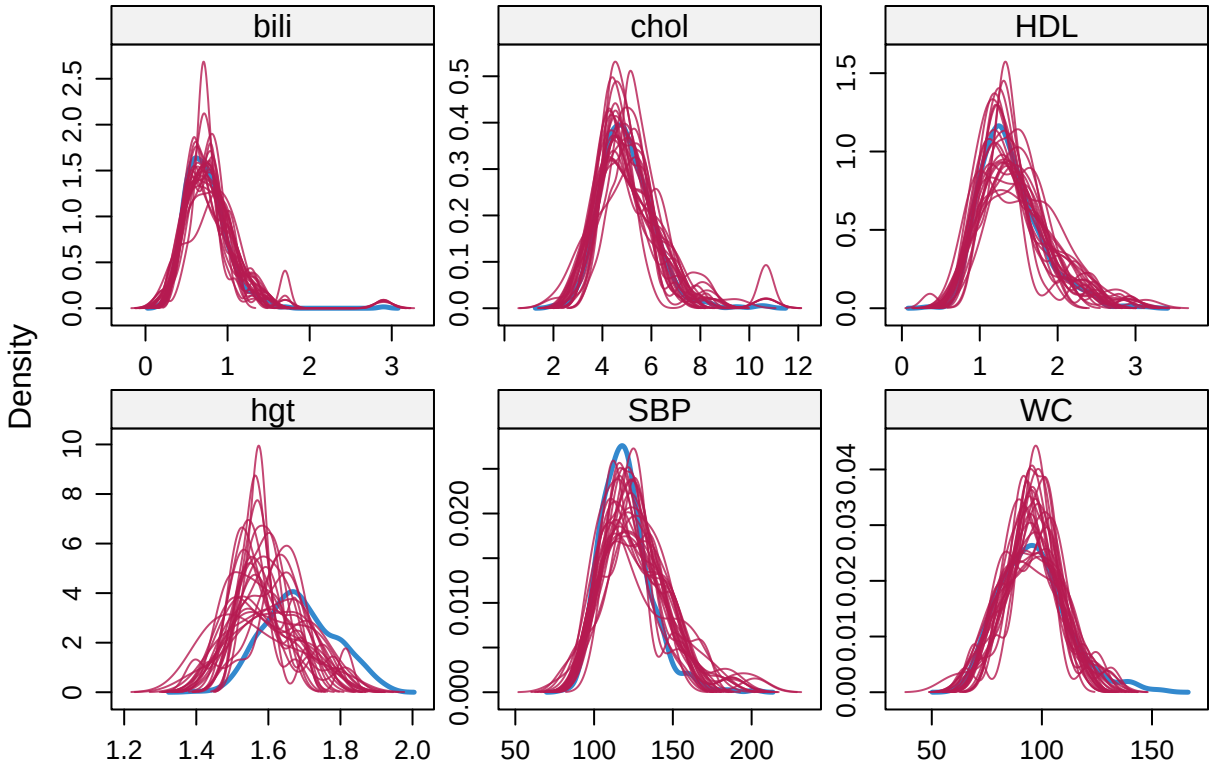
We have no logged events, so mice has had no issues when running the code, and see no obvious problems with convergance. We can also check density plots to double check the distribution of the imputed variables agrres with the observed ones.

```
densityplot(impdata)
```



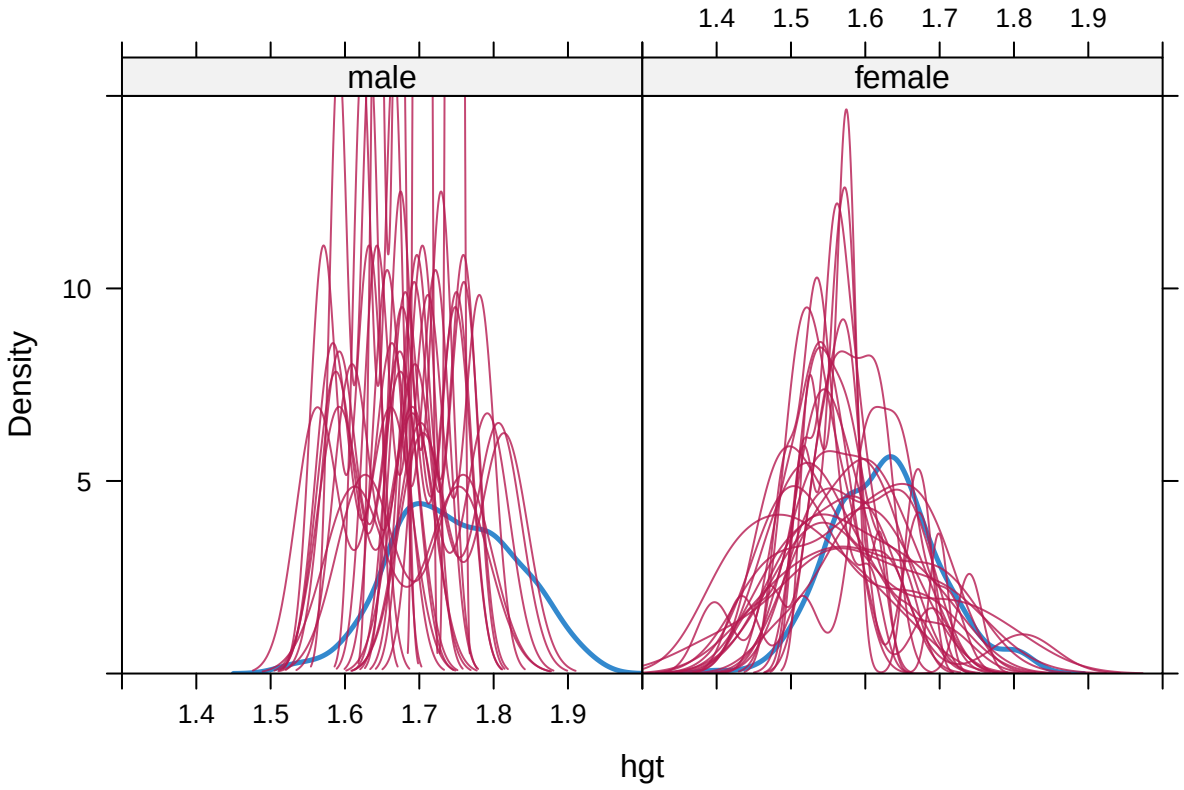
Here we do see an issue, the imputed values of height are well clearly skewed from the observed values. Given these results we should go back to our predictive model and consider using norm instead of pmm. Despite my initial hesitancy our results clearly show that pmm isn't working.

```
#Load the predictive matrix
meth = impdata$meth
#Edit predictor for height
meth['hgt'] = 'norm'
#Also changing for waist circumference to help accuracy
meth['WC'] = 'norm'
#Now rerun imputation and plot again
impdata2 = mice(data2, m = 25, maxit = 20, seed = 1, printFlag = F, method = meth)
densityplot(impdata2)
```



Hmm, this model is still showing issues. It should be noted we were only missing 11 values for height, perhaps they were outliers (or even missing MNAR), we should look a bit closer at what is missing. Perhaps we're missing a correlation, let's look at height and gender.

```
densityplot(impdata2, ~hgt|gender, xlim = c(1.3, 2), ylim =c(0,15))
```



Here, when accounting for gender, we see much more consistent density plots, suggesting our model robust enough to proceed. We will now move on to creating the and finishing and examining the model.

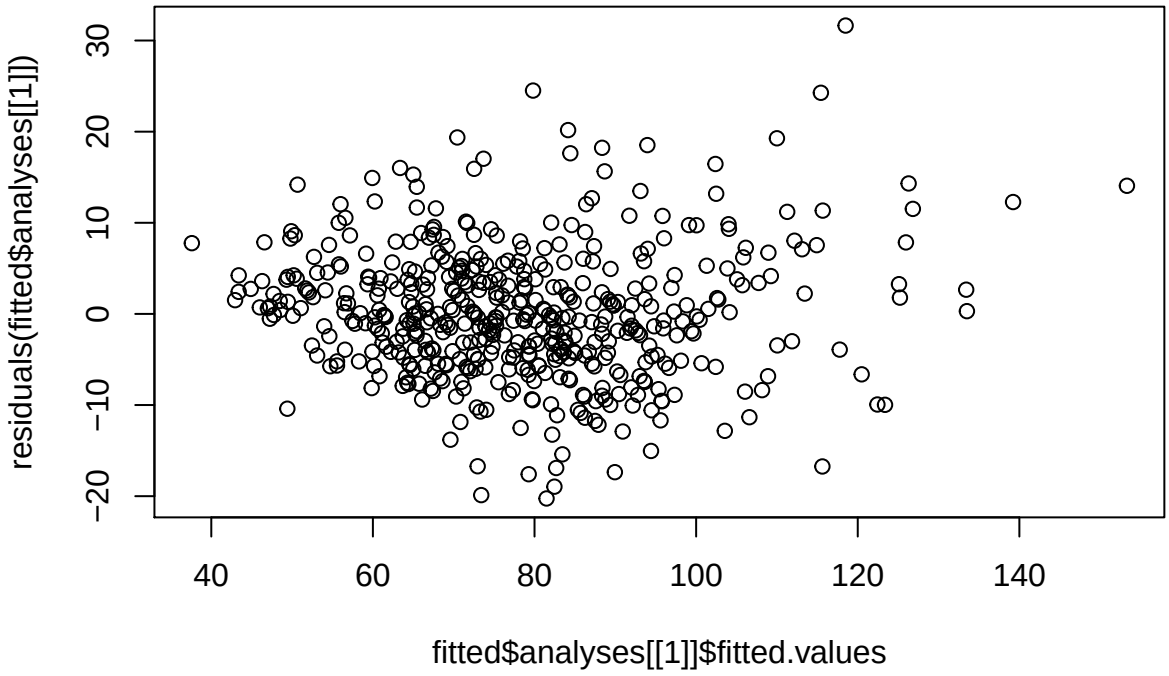
```
fitted = with(impdata2, lm (wgt ~ gender + age + hgt + WC))
summary(fitted$analyses[[1]])
```

```
##
## Call:
## lm(formula = wgt ~ gender + age + hgt + WC)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.252  -4.585  -0.341   3.955  31.642
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -101.94901    7.45924 -13.667  < 2e-16 ***
## genderfemale   -1.36071    0.82303  -1.653   0.0989 .
## age            -0.15216    0.02085  -7.297 1.18e-12 ***
## hgt            52.69178    4.25515  12.383  < 2e-16 ***
## WC              1.02972    0.02215  46.484  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.179 on 495 degrees of freedom
## Multiple R-squared:  0.8575, Adjusted R-squared:  0.8563
## F-statistic: 744.6 on 4 and 495 DF,  p-value: < 2.2e-16
```

Our model.

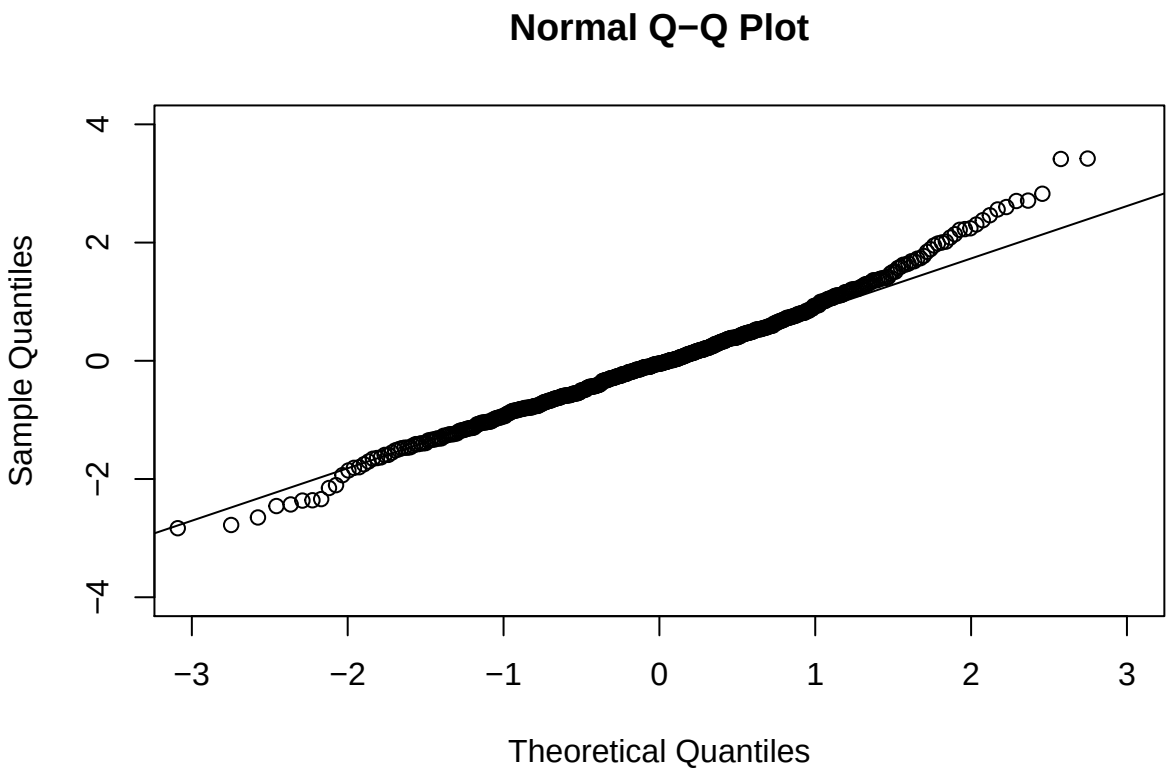
Now we'll check it through plotting.

```
plot(fitted$analyses[[1]]$fitted.values, residuals(fitted$analyses[[1]]))
```



In our first plot (fitted vs residuals) we can see (whilst we start around 0) an increasing variance as the we go rightward along the graph. This would indicate a certain amount of heteroskedacity. This isn't ideal, however by also checking the qq plot we that the model holds for the most part at least.

```
qqnorm(rstandard(fitted$analyses[[1]]), xlim = c(-3, 3), ylim = c(-4, 4))
qqline(rstandard(fitted$analyses[[1]], col = 2))
```



Here our tails only deviate towards the ends, backing up the model, even if it isn't perfect.

Now finally let's give our coefficients

```
summary(pool(fitted), conf.int = T)[c(1,2,7,8)]
```

```
##           term     estimate        2.5 %      97.5 %
## 1  (Intercept) -100.8813248 -115.9072584 -85.8553913
## 2 genderfemale   -1.3318448   -2.9717613   0.3080716
## 3          age   -0.1567008   -0.1985983  -0.1148032
## 4          hgt   52.4760861   43.8822630  61.0699091
## 5           WC    1.0247435    0.9807554   1.0687317
```

Here we see the results of our model after pooling, and that is that.

Have a good Christmas and thank you for an interesting course!