Test Question 1 (Imputation)

Mario Ibanez

April 21, 2016

Introduction

I will write this report for question 1 using R Markdown, which allows the code and report to be combined into one document. The data set consists of readings from three temperature sensors and the assignment is to impute the missing values using four different methods. The results should be plotted and evaluated.

Before beginning, the first step can be evaluating the data set to see what the structure is and how many values are missing. As I mentioned before, the report and code will be contained together in this assignment since the purpose is to demonstrate my work in R.

The code below loads the data from a .csv file in the same directory:

```
# Load data
Data1 <- read.csv(file = "Data1.csv",
                  header = TRUE,
                  na.strings = "NA")[1:100, ]
```

There are 100 data points and over a thousand blank rows, so I manually subsetted it to 100 rows.

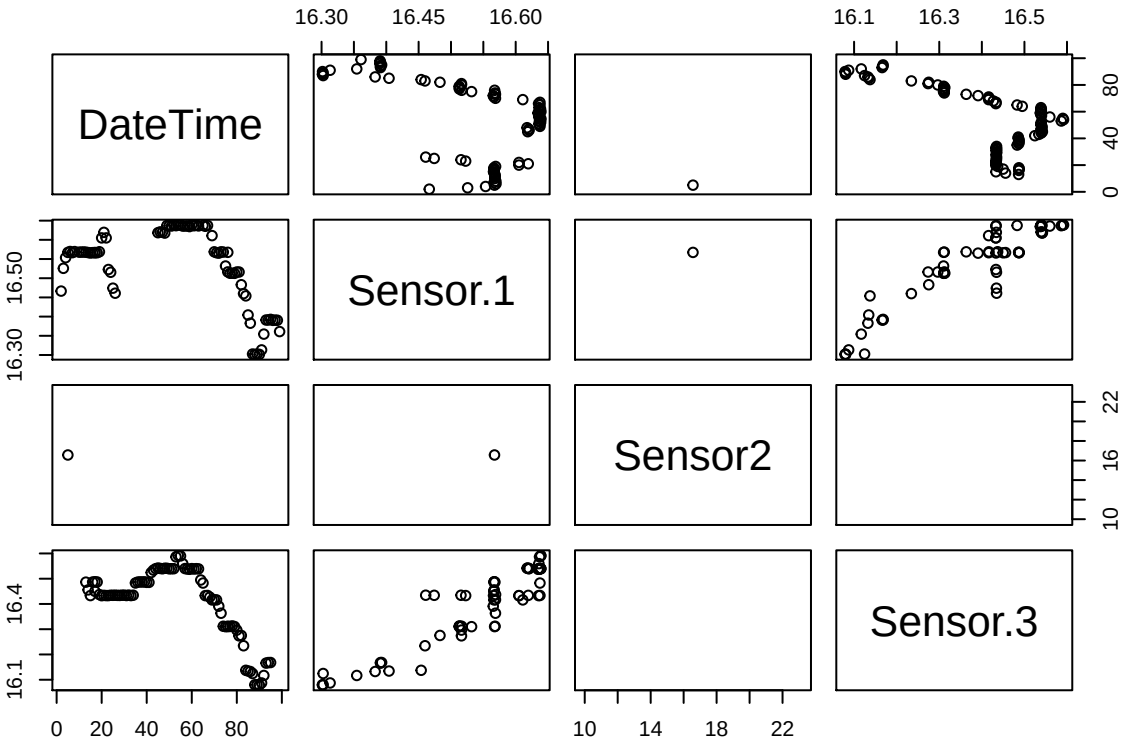
Next, we should look at how many missing values there are in each column.

```
# Return the number of NA values in each column
colSums(apply(X = Data1, FUN = is.na, MARGIN = 2))
```

```
## DateTime Sensor.1  Sensor2 Sensor.3
##        0       21       99       15
```

Out of 100 rows, Sensor2 has 99 missing values. The date column has no missing values, and Sensor.1 and Sensor.3 have about 20% of their values missing. In my opinion, there is not much that can be done with Sensor.3 in this case, other than possibly setting the entire column equal to that 1 present value. Plotting the original data set is also worthwhile:

```
# Scatterplot of all variables
plot(Data1)
```



It is seen immediately that Sensor2 has only one value present. It is also worth checking the amount of "overlap" among the missing values for Sensor.1 and Sensor.3:

```
# Return the rows that have missing values in each column
Data1[, 2:4][rowSums(is.na(Data1[, 2:4])) == 3, ]
```

```
##   Sensor.1 Sensor2 Sensor.3
## 8       NA      NA       NA
```

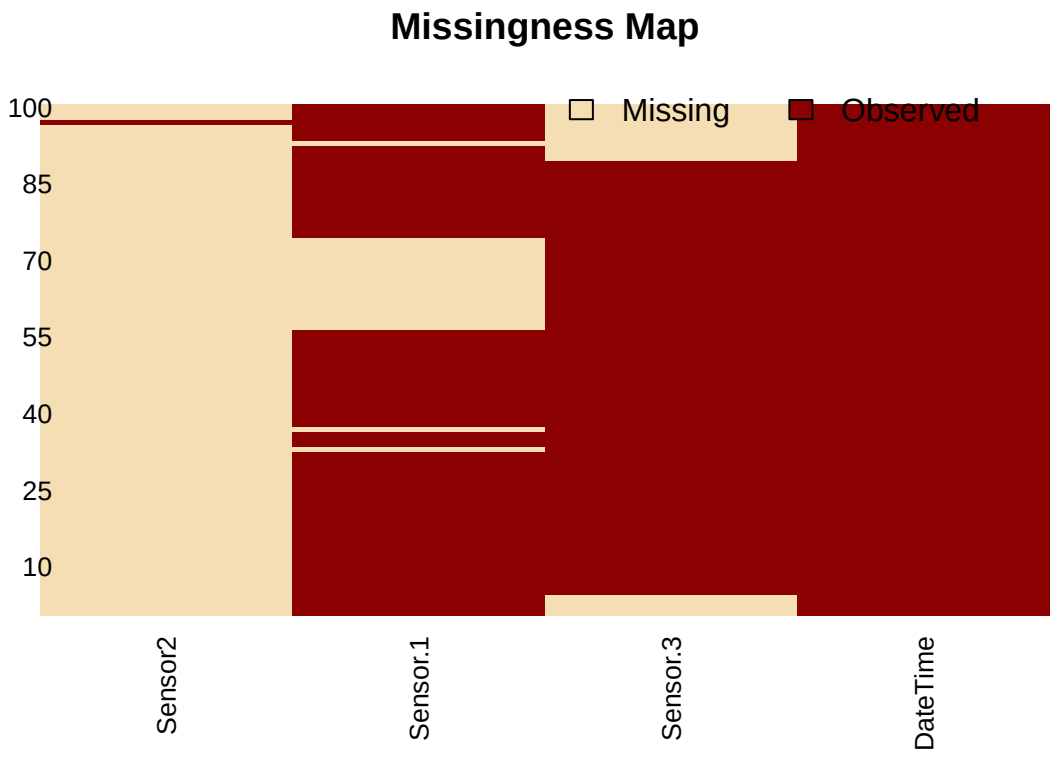
This shows that only one row has NA values in all 3 columns. This is worth noting because some imputation methods could be affected by this. In any case, we can no proceed with the four imputation methods.

Method 1 (Amelia Package)

For the first method, I will try the Amelia package in R. Information about the package can be found at https://cran.r-project.org/web/packages/Amelia/vignettes/amelia.pdf . The name of the package is "Amelia" and would need to be installed beforehand. The authors of the package are two Harvard professors and a UCLA professor.

First we can try the function *missmap()* which creates a plot of the missing values.

```
# Load Amelia package and plot missingness map
library(Amelia)
missmap(Data1)
```

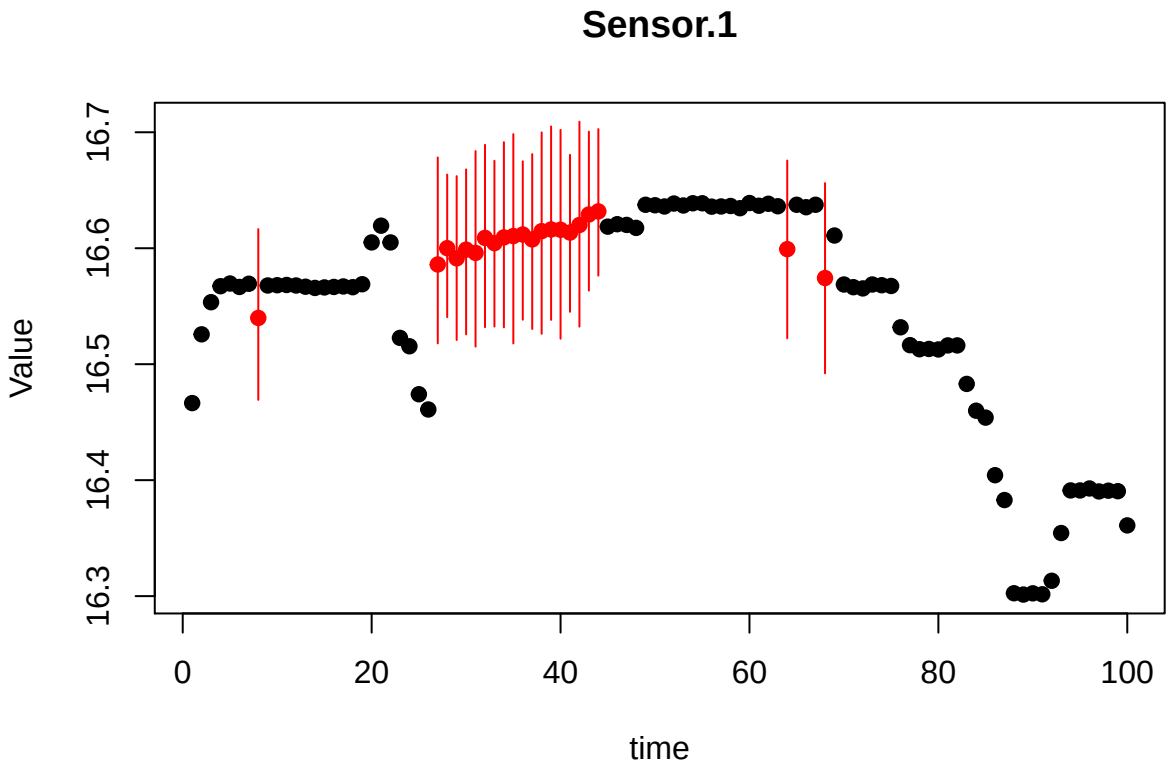


The plot does not come out perfectly, but we get a good visualization of the missing data. Sensor2 only has a single value, and the missing values for Sensor.1 and Sensor.3 overlap at only a single point which is good. It is important that we realize that this is time series data and use an imputation method that takes this into account.

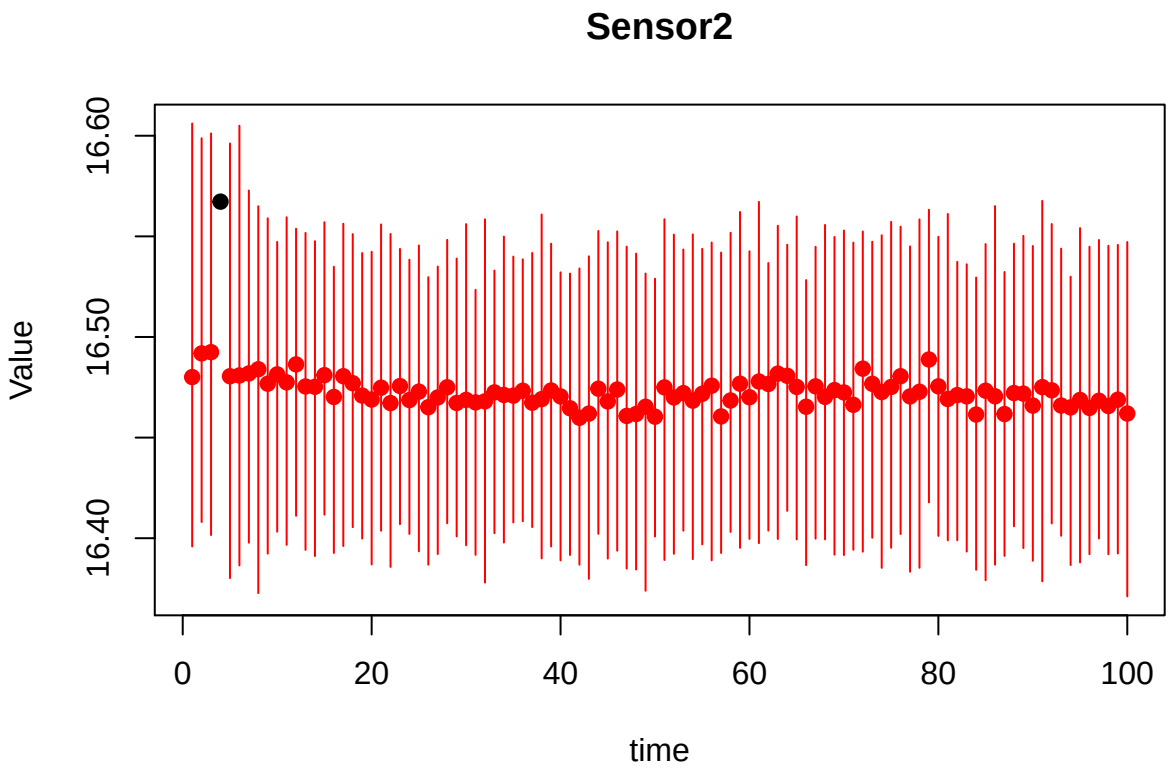
```
# New data set for this imputation method, needs to be reshaped
Data1_amelia <- Data1
Data1_amelia$time <- 1:100
Data1_amelia <- reshape(data = Data1,
                        varying = 2:4 ,
                        direction = "long",
                        v.names = "Value",
                        times = c("Sensor.1", "Sensor2", "Sensor.3"),
                        timevar = "Sensor")

# Impute values using using the fact that it is time series data
Data1_amelia <- amelia(Data1_amelia,
                       ts = "id",
                       polytime = 3,
                       idvars = 1,
                       cs = "Sensor",
                       intercs = TRUE, p2s = 0)
```

We can now look at how the values were imputed:

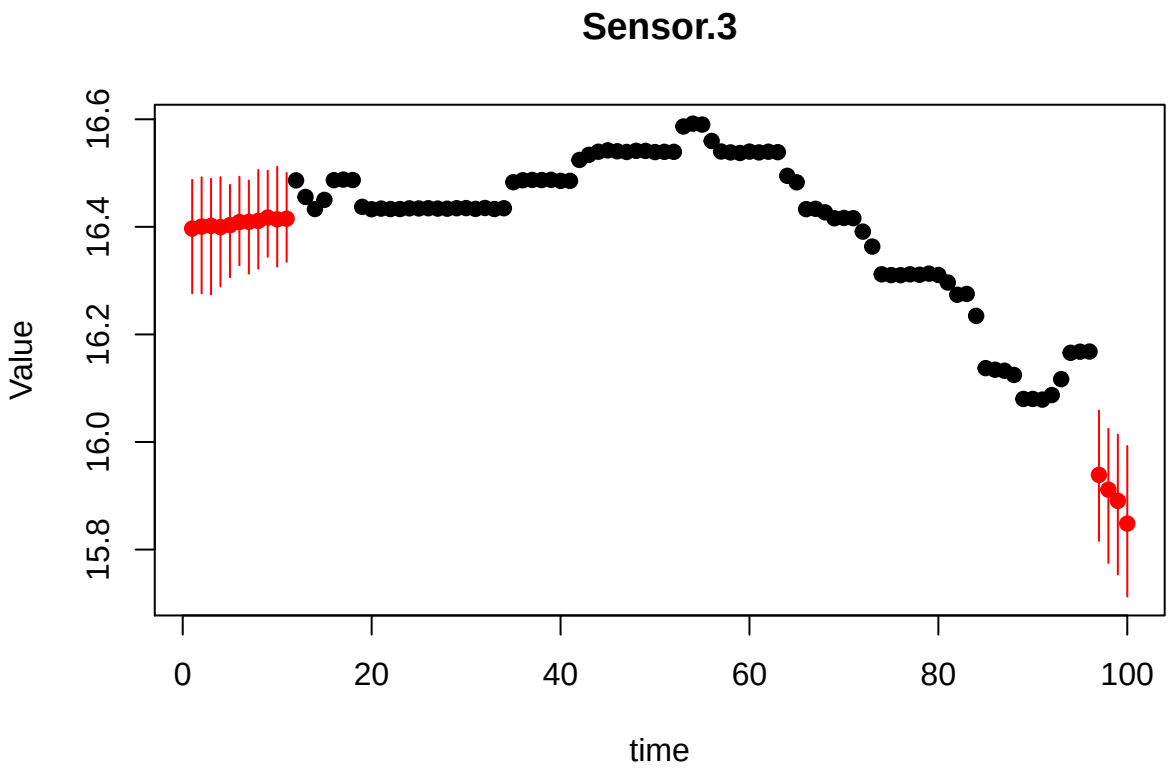
```
tscsPlot(Data1_amelia, var = "Value", cs = "Sensor.1")
```




```
tscsPlot(Data1_amelia, var = "Value", cs = "Sensor2")
```



```r
tscsPlot(Data1_amelia, var = "Value", cs = "Sensor.3")
```



There are many adjustments that can be made within this package but this was just a quick exploration and improvements and further investigation could certainly be done. The red bars are 90% confidence intervals for the imputed values. The function uses the EM algorithm and bootstrapping to do these imputations.

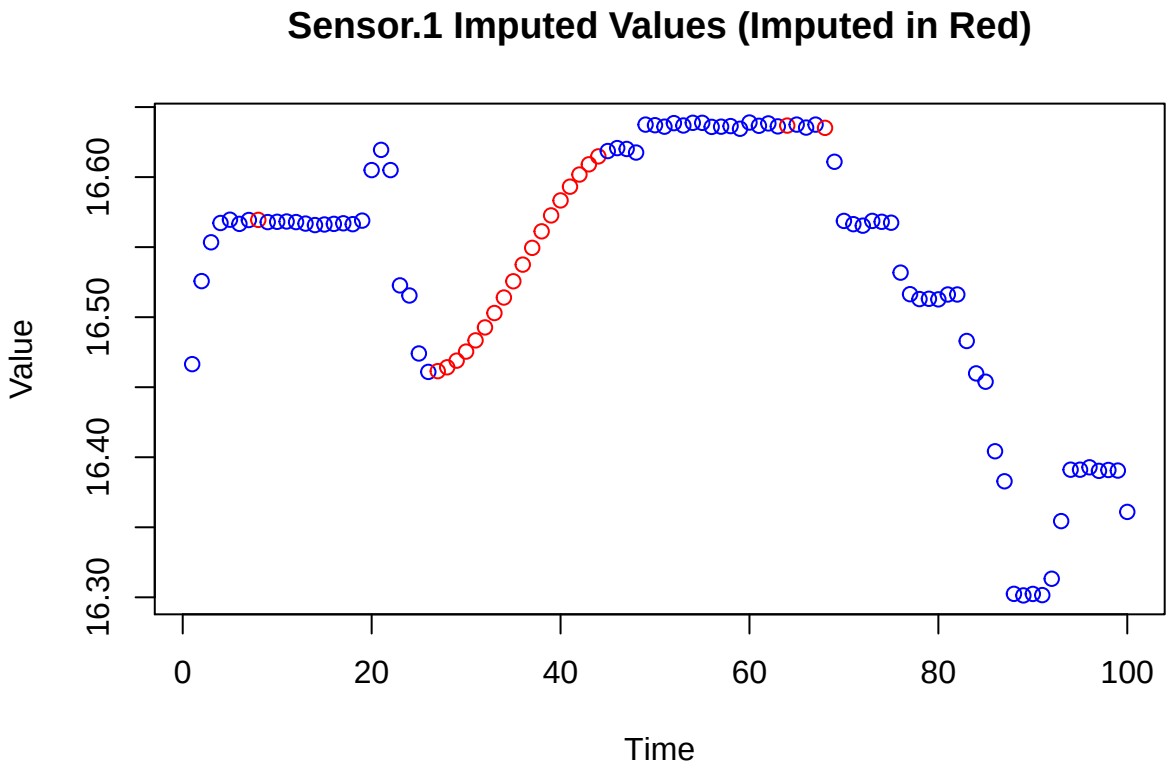
Method 2 (imputeTS Package)

Another package available in R is "imputeTS". It is used especially for imputing missing values in time series data. The function *na.interpolation()* works on time series objects. The code below first makes the sensor data into time series format and then applies the interpolation function to each sensor:

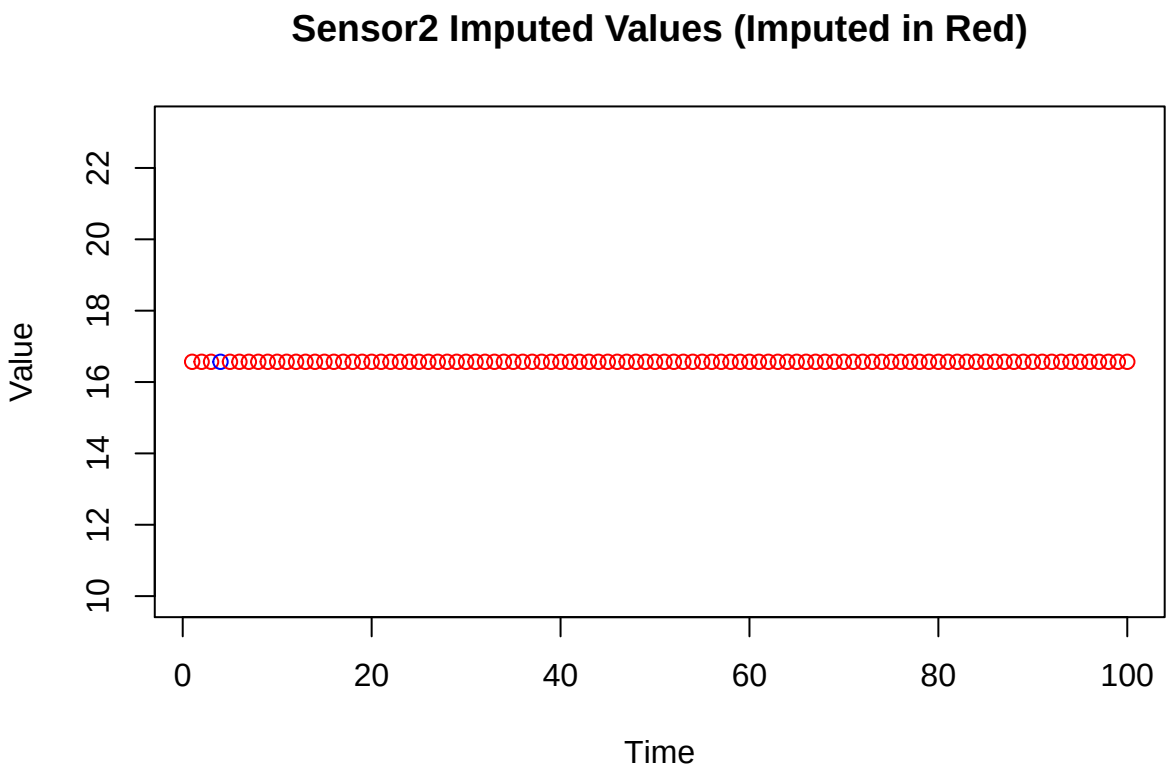
```r
# load package
library(imputeTS)

# Convert to time series, apply function, and convert back to data frame
Data1_imputeTS <- data.frame(lapply(lapply(X = Data1[, 2:4],
                                           FUN = as.ts),
                                    FUN = na.interpolation,
                                    option = "spline"))
Data1_imputeTS <- data.frame(Data1$DateTime, Data1_imputeTS)
Data1_imputeTS[, 2:4] <- lapply(Data1_imputeTS[, 2:4], as.numeric)
```

To check the results, we can plot each of the sensors against the row number (time).

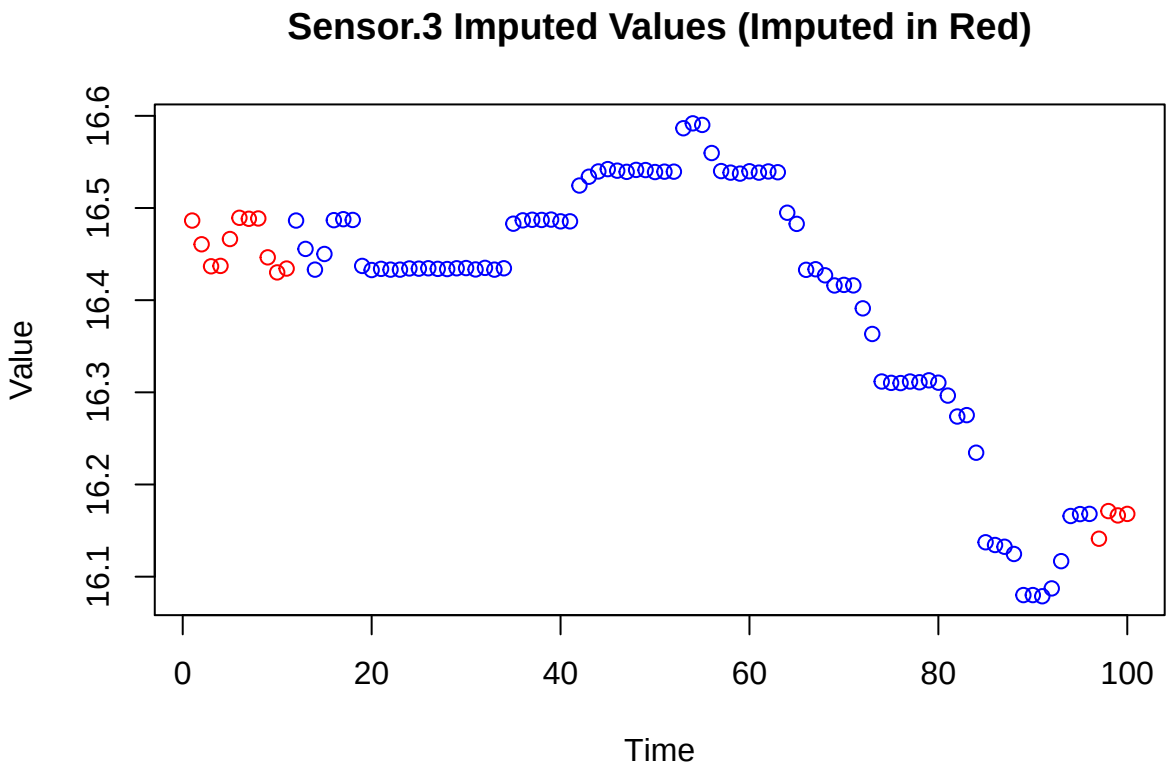
```
plot(x = 1:100,
     y = Data1_imputeTS$Sensor.1,
     col = ifelse(is.na(Data1$Sensor.1), "red", "blue"),
     main = "Sensor.1 Imputed Values (Imputed in Red)",
     xlab = "Time",
     ylab = "Value")
```




```
plot(x = 1:100,
     y = Data1_imputeTS$Sensor2,
     col = ifelse(is.na(Data1$Sensor2), "red", "blue"),
     main = "Sensor2 Imputed Values (Imputed in Red)",
     xlab = "Time",
     ylab = "Value")
```



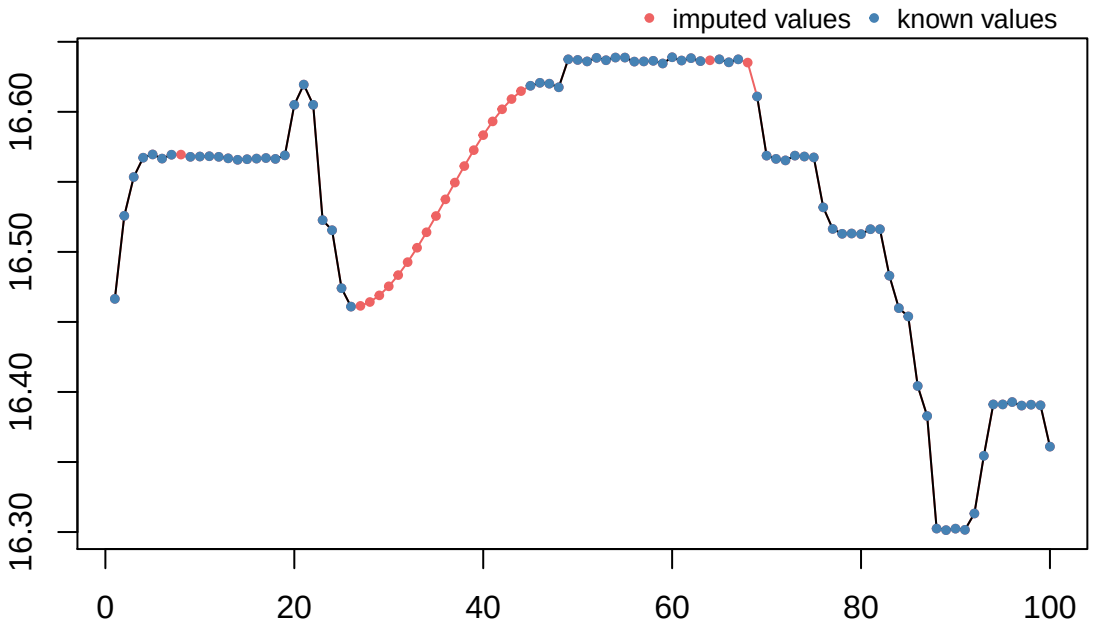

```
plot(x = 1:100,
     y = Data1_imputeTS$Sensor.3,
     col = ifelse(is.na(Data1$Sensor.3), "red", "blue"),
     main = "Sensor.3 Imputed Values (Imputed in Red)",
     xlab = "Time",
     ylab = "Value")
```



Overall, this spline method seems to work well and it is fairly simple to use.

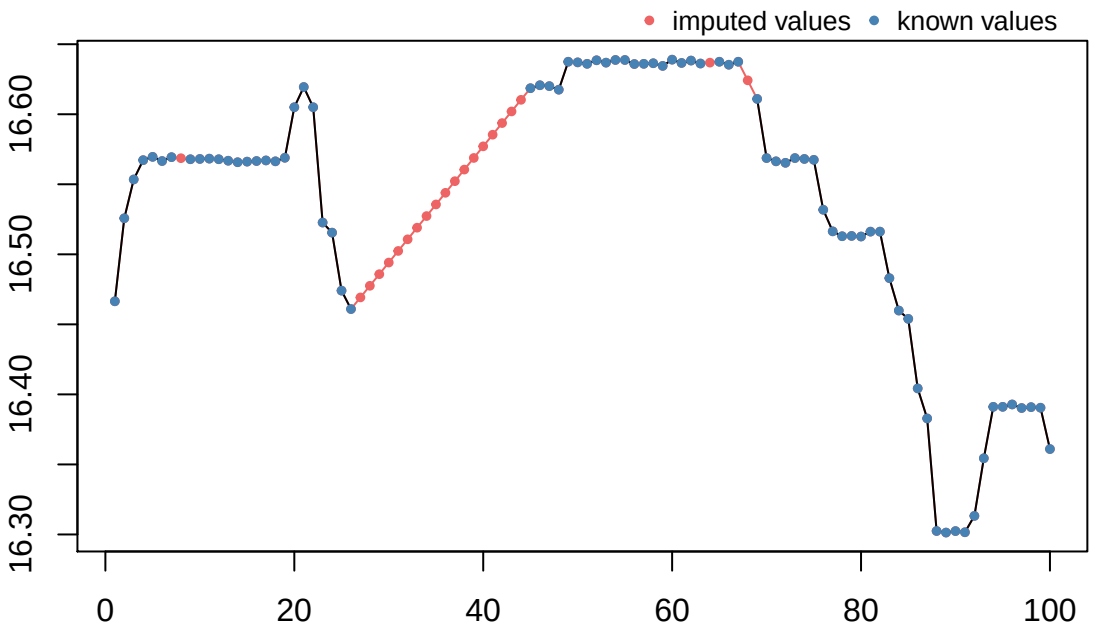
The package also has other capabilities. There is a function available to visualize imputed values for example:

```
vis.imputations(as.ts(Data1$Sensor.1),
                na.interpolation(as.ts(Data1$Sensor.1), option = "spline"))
```



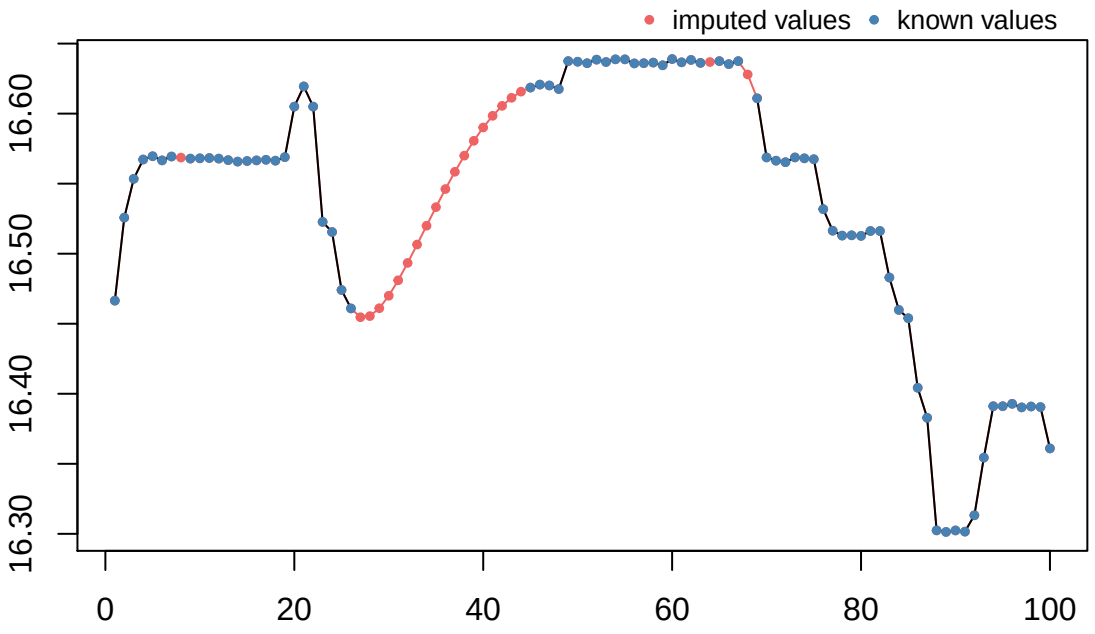
This plot is more visually appealing and is also easier to create than the ones I created by hand above. While we're at it, we can visualize how the other imputations would look on Sensor.1:

```
vis.imputations(as.ts(Data1$Sensor.1),
                na.interpolation(as.ts(Data1$Sensor.1), option = "linear"))
```



Using the option "linear" fills in gaps using a line.

```
vis.imputations(as.ts(Data1$Sensor.1),
                na.interpolation(as.ts(Data1$Sensor.1), option = "stine"))
```



This option produces a smooth curve.

Method 3 (Last observation carried forward - LOCF)

This is another method that is applicable to time series data. As the name of the method implies, missing values are replaced with whatever the last scene value was. The R packages above are capable of doing this, but for variety I will do this manually. One thing to check first is what the first row looks like:

```
head(Data1)
```

```
##          DateTime Sensor.1  Sensor2 Sensor.3
## 1 9/29/2015 23:00 16.46646       NA       NA
## 2 9/29/2015 23:01 16.52575       NA       NA
## 3 9/29/2015 23:02 16.55345       NA       NA
## 4 9/29/2015 23:03 16.56724 16.56724       NA
## 5 9/29/2015 23:04 16.56958       NA       NA
## 6 9/29/2015 23:05 16.56660       NA       NA
```

Unfortunately, there are NA values in the first few rows for both Sensor2 and Sensor.3. I will use the first observation of each column and use that to replace the leading missing values. I will use a double for loop even though it is obviously something to avoid in R when possible (normally this would just be done with the package above, not done by hand anyway).

```
# new copy of data
Data1_locf <- Data1

# double for loop to go across rows and columns to replace missing values
for(j in 2:dim(Data1_locf)[2]){
  # replace any leading NA value
  if(is.na(Data1_locf[1, j])){
    Data1_locf[1, j] <- Data1_locf[min(which(!is.na(Data1_locf[, j]))), j]
  }
  # replace any subsequent missing values
  for(i in 2:dim(Data1_locf)[1]){
    if(is.na(Data1_locf[i, j])){
      Data1_locf[i, j] <- Data1_locf[i-1, j]
    }
  }
}
```

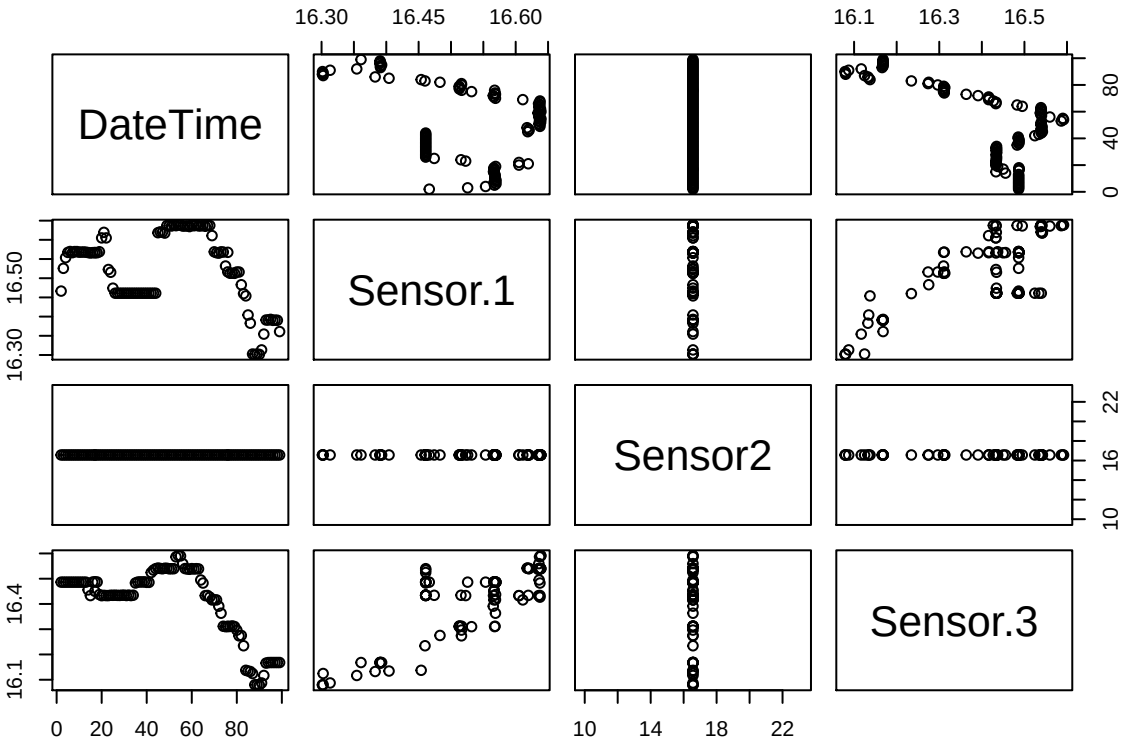
To test that all the NA values were replaced, we can run this command:

```
# Return the number of NA values in each column
colSums(apply(X = Data1_locf, FUN = is.na, MARGIN = 2))
```

```
## DateTime Sensor.1  Sensor2 Sensor.3
##        0        0        0        0
```

Good. We can also run the plot to see the results:

```
plot(Data1_locf)
```



As expected, Sensor2 now has the same value repeated 100 times. In the plots of Sensor.1 and Sensor.3 against time, it is especially easy to see the horizontal pieces that have been added in.

Method 4 (Mean Imputation)

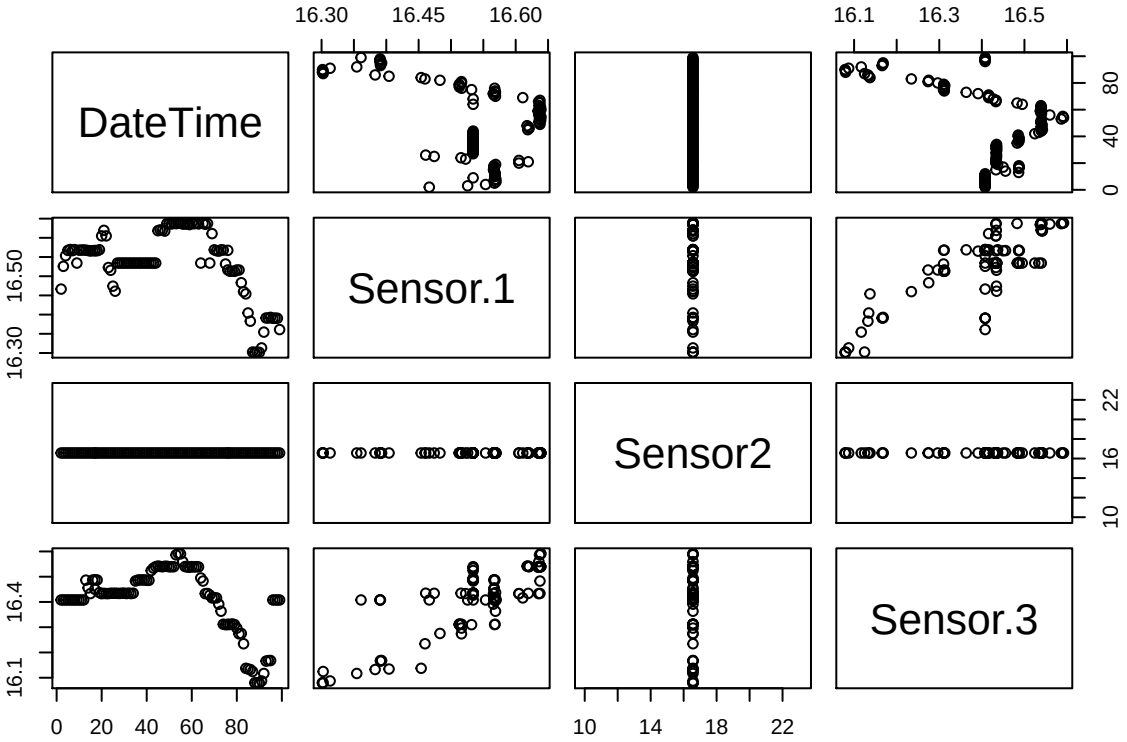
The last method, mean imputation, is also a valid method for time series data, though likely not the best possible choice (methods 1 and 2 are better choices). Nonetheless, here is some interesting and concise R code that accomplishes this task:

```
# R tricks to replace all NA values with their column means
Data1_mi <- data.frame(lapply(Data1, function(x){
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  x}))
```

```
## Warning in mean.default(x, na.rm = TRUE): argument is not numeric or
## logical: returning NA
```

We can look at the same plot as before:

```
plot(Data1_mi)
```



Again, this would not be the best method for time series data.

Conclusion / Comparison

In the assignment sent to me by Xiupeng, it mentioned using prediction metrics like MAE and Std to evaluate the methods. I assume MAE is mean absolute error and that Std is standard deviation. I don't see how either of these would be used to evaluate an imputation. Standard deviation for a data set would be minimized by imputing mean values, but minimizing the standard deviation is not the goal. Visually, for this time series data, the spline method from the the *imputeTS* package seemed to work very well. It would be expected that most sensors' readings would vary smoothly and continuously as time passes.

Since Sensor2 only has 1 value out of 100, the only thing that would make sense to do would be to use that 1 value to fill the 99 missing values. There is no reason to suppose that there is any trend up or down or that it is correlated with any of the other two sensors. If there had been even a few more values present, it would have been possible to infer its behavior or correlation with the other sensors. The *Amelia* package did not set the missing values this way, but *imputeTS* did. Overall I thought the *imputeTS* package was a great choice.