Imputing with imputeTS

Kevin Nolasco

2022-06-12

Load Libraries

```
library(imputeTS) # for imputation functions and dataset
```

```
## Warning: package 'imputeTS' was built under R version 4.1.3
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(glue) # to format printing
```

```
## Warning: package 'glue' was built under R version 4.1.3
```

```
library(tictoc) # to know how long the imputation takes
```

Load Data

```
# load dataset ----
data_with_na <- tsNH4
data_complete <- tsNH4Complete
```

Hypothesis

According to the documentation for imputeTS, the tsNH4 dataset is a time-series NH4 concentration in a waste-water system. Since this is a time series dataset, I believe seasonality will play a big role on the way the dataset behaves. Therefore, a simple imputation method - such as using the mean, median, or mode will not be sufficient to capture the trend of the data. I believe the best method for imputing this dataset will be an interpolation model with spline or Stineman Interpolation.

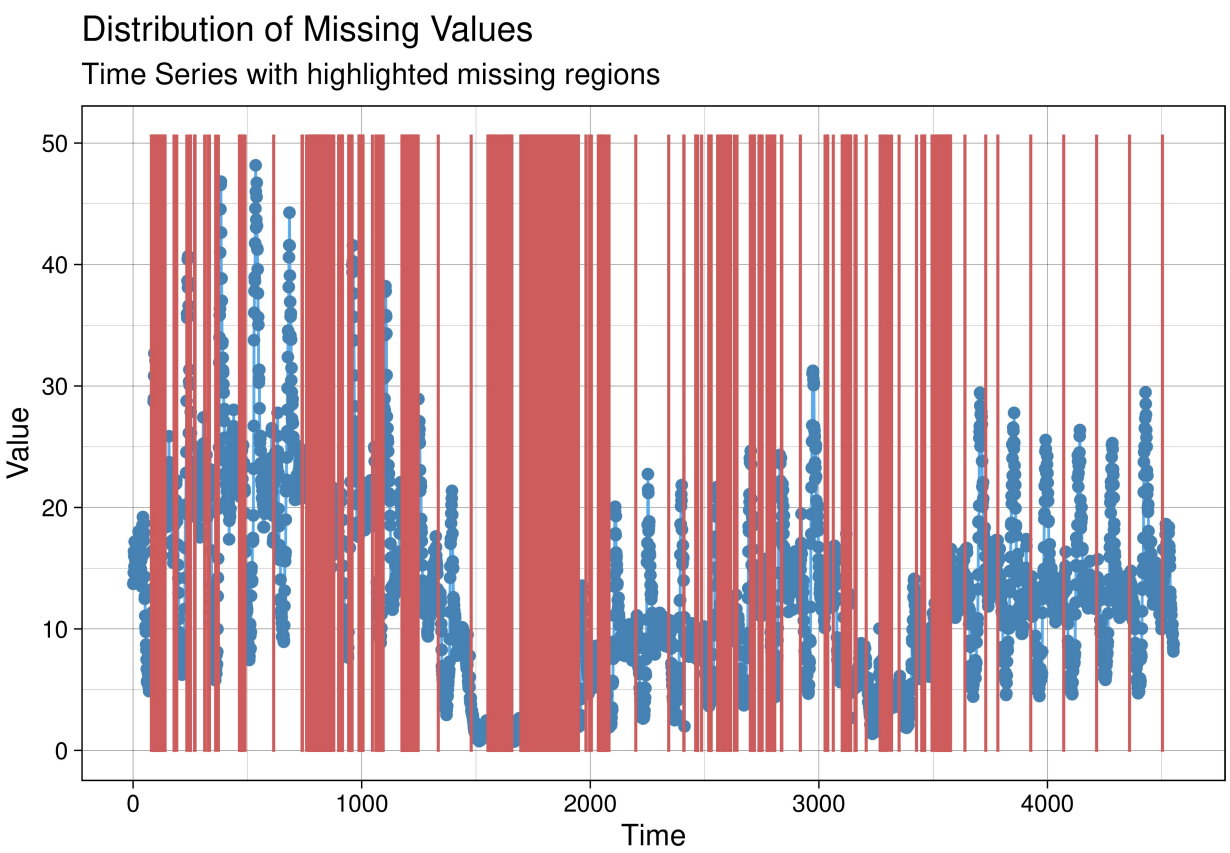
We will test three interpolation functions (na_interpolation, na_kalman, and na_seadec) and we will tweak each parameters for the best result. For example, we will impute using na_interpolation with linear, spline, and stine interpolation.

The results of the imputations will be evaluated using RMSE (root mean squared error).

Data Visualization

Let's see what the dataset looks like, and how many missing values there are.

```
ggplot_na_distribution(data_with_na)
```



We can visually see that there are a lot of missing values. Let's get a concrete set of how many values we are dealing with using the stats package included in imputeTS.

```
statsNA(data_with_na)
```

```
## [1] "Length of time series:"
## [1] 4552
## [1] "-------------------------"
## [1] "Number of Missing Values:"
## [1] 883
## [1] "-------------------------"
## [1] "Percentage of Missing Values:"
## [1] "19.4%"
## [1] "-------------------------"
## [1] "Number of Gaps:"
## [1] 155
## [1] "-------------------------"
## [1] "Average Gap Size:"
## [1] 5.696774
```

```
## [1] "-------------------------"
## [1] "Stats for Bins"
## [1] "  Bin 1 (1138 values from 1 to 1138) :      233 NAs (20.5%)"
## [1] "  Bin 2 (1138 values from 1139 to 2276) :      433 NAs (38%)"
## [1] "  Bin 3 (1138 values from 2277 to 3414) :      135 NAs (11.9%)"
## [1] "  Bin 4 (1138 values from 3415 to 4552) :      82 NAs (7.21%)"
## [1] "-------------------------"
## [1] "Longest NA gap (series of consecutive NAs)"
## [1] "157 in a row"
## [1] "-------------------------"
## [1] "Most frequent gap size (series of consecutive NA series)"
## [1] "1 NA in a row (occurring 68 times)"
## [1] "-------------------------"
## [1] "Gap size accounting for most NAs"
## [1] "157 NA in a row (occurring 1 times, making up for overall 157 NAs)"
## [1] "-------------------------"
## [1] "Overview NA series"
## [1] "  1 NA in a row: 68 times"
## [1] "  2 NA in a row: 26 times"
## [1] "  3 NA in a row: 16 times"
## [1] "  4 NA in a row: 10 times"
## [1] "  5 NA in a row: 8 times"
## [1] "  6 NA in a row: 4 times"
## [1] "  7 NA in a row: 2 times"
## [1] "  8 NA in a row: 3 times"
## [1] "  9 NA in a row: 2 times"
## [1] "  10 NA in a row: 1 times"
## [1] "  11 NA in a row: 1 times"
## [1] "  12 NA in a row: 2 times"
## [1] "  14 NA in a row: 1 times"
## [1] "  16 NA in a row: 1 times"
## [1] "  17 NA in a row: 1 times"
## [1] "  21 NA in a row: 1 times"
## [1] "  25 NA in a row: 1 times"
## [1] "  26 NA in a row: 1 times"
## [1] "  27 NA in a row: 1 times"
## [1] "  32 NA in a row: 1 times"
## [1] "  42 NA in a row: 2 times"
## [1] "  91 NA in a row: 1 times"
## [1] "  157 NA in a row: 1 times"
```

From the stats output, we can see the dataset consists of 4,552 data points with 883 missing values (19.4% of values are missing).

Impute Values

na_interpolation

```
na_interpolation_results <- function(dataset, dataset_no_missing){
  # impute
  tic()
```

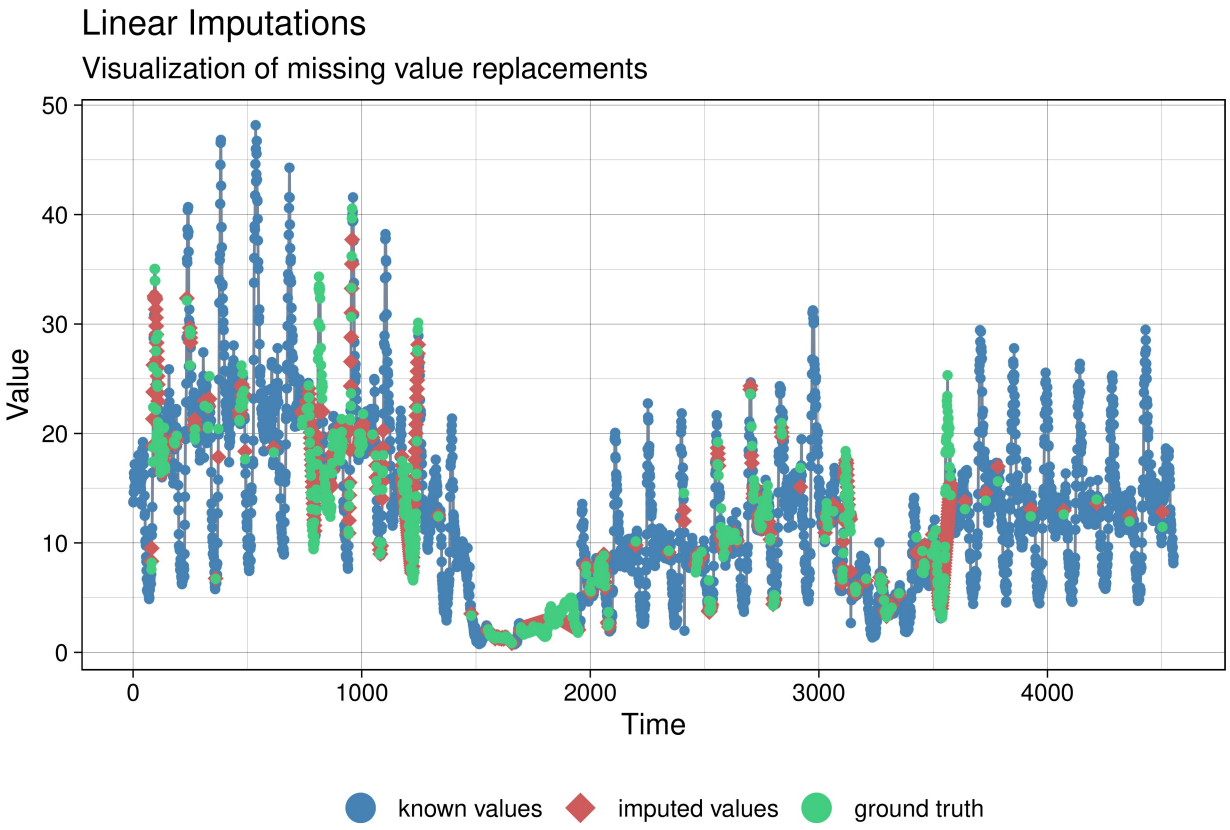
```
  with_linear <- na_interpolation(dataset, option = "linear")
  print("Time to impute using Linear Interpolation:")
  toc()
  tic()
  with_spline <- na_interpolation(dataset, option = "spline")
  print("Time to impute using Spline Interpolation:")
  toc()
  tic()
  with_stine <- na_interpolation(dataset, option = "stine")
  print("Time to impute using Stine Interpolation")
  toc()

  datasets <- list(with_linear, with_spline, with_stine)
  names <- list("Linear", "Spline", "Stine")
  all_rmse <- c()
  # calculate and print rmse's
  for(ind in 1:3){
    name <- names[[ind]]
    data <- datasets[[ind]]
    rmse <- sqrt(mean((data - dataset_no_missing)^2))
    print(glue("RMSE for {name} Imputation : {rmse}"))
    all_rmse <- c(all_rmse, rmse)
  }
  # print method with lowest rmse
  min_ind <- which.min(all_rmse)
  lowest_rmse <- all_rmse[min_ind]
  name_lowest_rmse <- names[[min_ind]]
  print(glue("{name_lowest_rmse} Imputation had the lowest RMSE of: {lowest_rmse}"))
  return(datasets)
}
imputed_datasets <- na_interpolation_results(data_with_na, data_complete)
```

```
## [1] "Time to impute using Linear Interpolation:"
## 0 sec elapsed
## [1] "Time to impute using Spline Interpolation:"
## 0 sec elapsed
## [1] "Time to impute using Stine Interpolation"
## 0 sec elapsed
## RMSE for Linear Imputation : 1.06253169986563
## RMSE for Spline Imputation : 2.05536200992844
## RMSE for Stine Imputation : 1.15842511335834
## Linear Imputation had the lowest RMSE of: 1.06253169986563
```

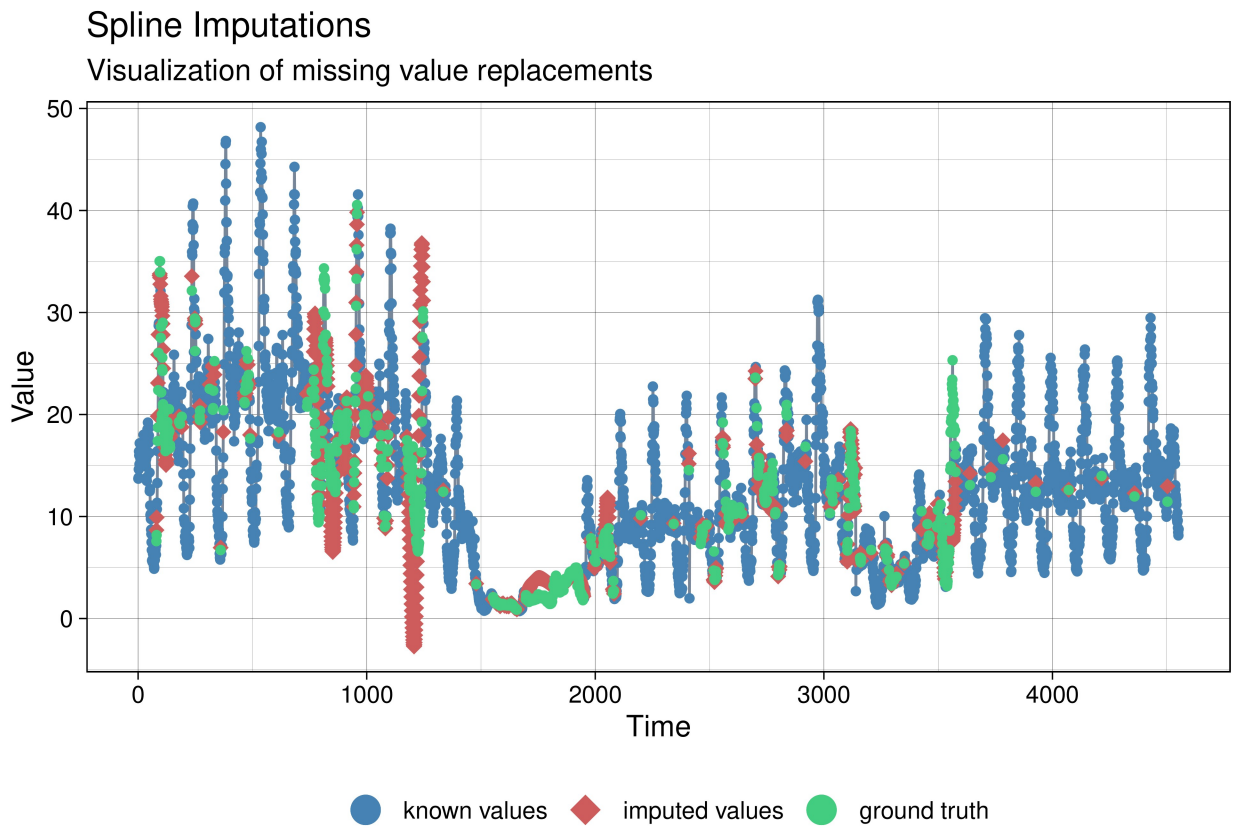
Based on the analysis above, we could see that the linear interpolation worked best for imputing this dataset. My hypothesis was that the spline or Stineman interpolation would work best. From my understanding, spline uses a polynomial function to approximate the next data point, this could why the interpolation did not work well. There is a chance that the interpolating polynomial was estimated with a high degree. Models using high degree polynomials have high variance, which explains the higher RMSE's.

Let's plot the imputed values for each method.

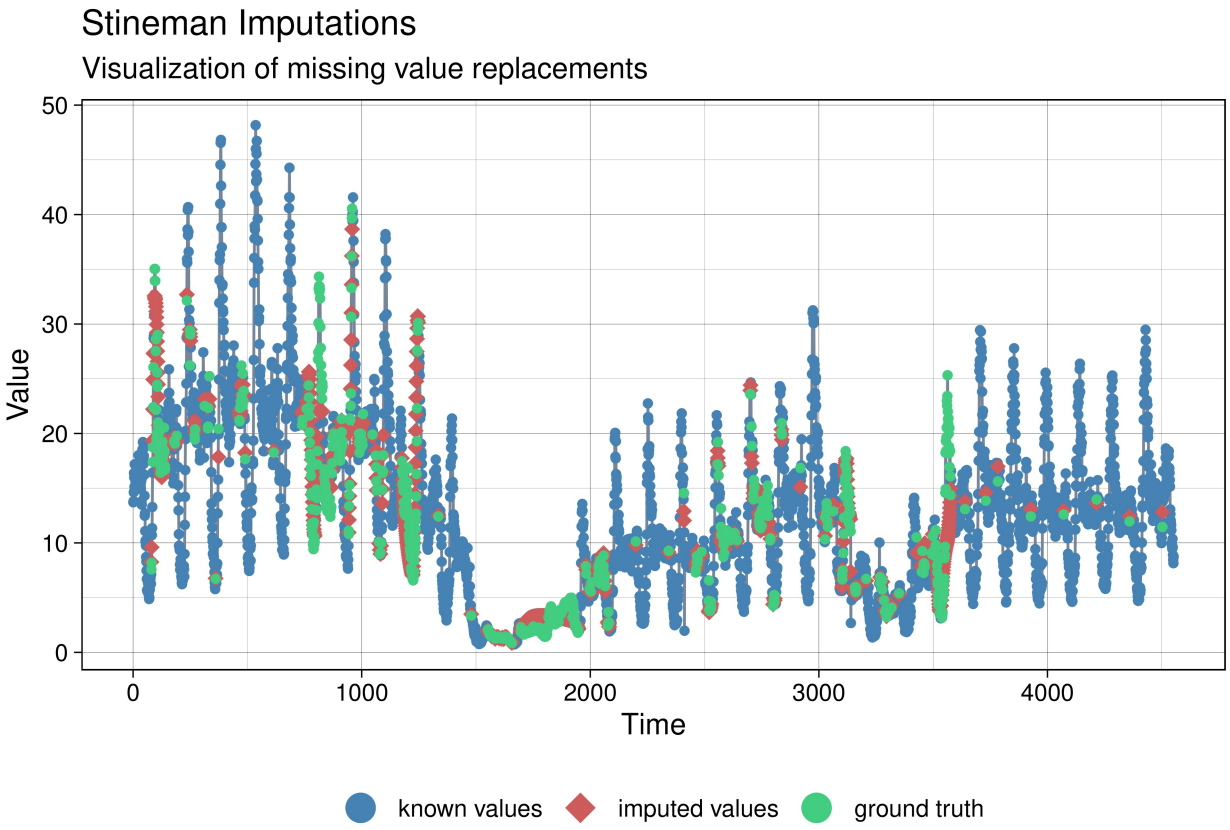
```
ggplot_na_imputations(data_with_na, imputed_datasets[[1]], data_complete, title = "Linear Imputations")
```




```
ggplot_na_imputations(data_with_na, imputed_datasets[[2]], data_complete, title = "Spline Imputations")
```




```
ggplot_na_imputations(data_with_na, imputed_datasets[[3]], data_complete, title = "Stineman Imputations
```



From the plots above, we can see the imputed values for Spline interpolation between x = 1000 and x = 1500, the imputed values are very off from the actual values. This is probably due to the high degree polynomial that we discussed in the section before.

na_kalman

According to the documentation, the kalman methods for imputing work better for datasets that contain seasonality. As we saw above, the dataset does contain some seasonality, so let's see how these imputation techniques do.

NOTE the plan was to run the kalman imputation method using model = "auto.arima" and "StructTS" but the imputations gave no results even after 30 minutes of letting it run. Because of this, I will not be including these methods in this analysis.

na_seadec

Just like the na_kalman methods, the na_seadec method is supposed to work well with datasets that contain seasonality. Let's see how well it performs on our dataset. We will try 5 methods included in the na_seadec package: "interpolation", "locf", "mean", "random", and "ma". We will not be trying the kalman method because it takes too long to run.

```
na_seadec_results <- function(dataset, dataset_no_missing){
  # impute
  tic()
  with_inter <- na_seadec(dataset, algorithm = "interpolation")
```

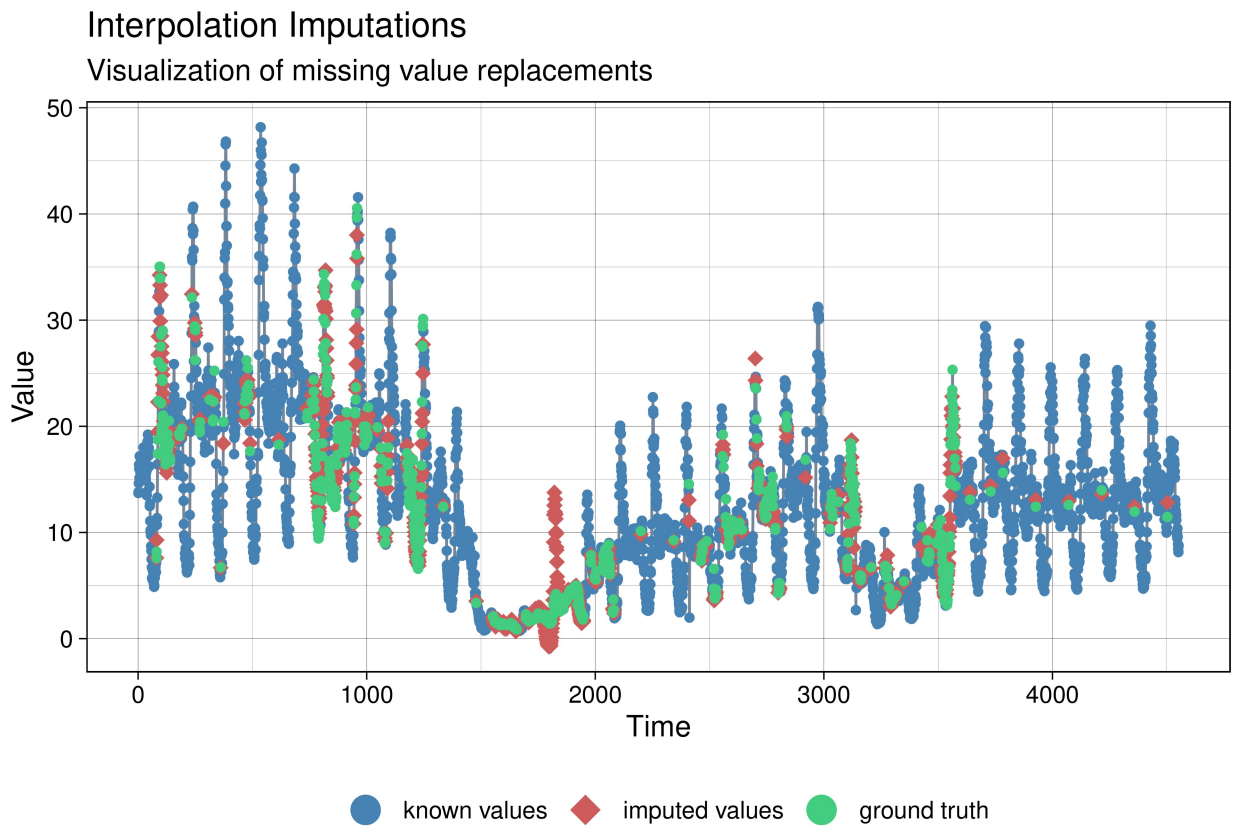
```
  print("Time to impute using Interpolation:")
  toc()
  tic()
  with_locf <- na_seadec(dataset, algorithm = "locf")
  print("Time to impute using LocF:")
  toc()
  tic()
  with_mean <- na_seadec(dataset, algorithm = "mean")
  print("Time to impute using Mean")
  toc()
  tic()
  with_random <- na_seadec(dataset, algorithm = "random")
  print("Time to impute using Random:")
  toc()
  tic()
  with_ma <- na_seadec(dataset, algorithm = "ma")
  print("Time to impute using MA")
  toc()

  datasets <- list(with_inter, with_locf, with_mean, with_random, with_ma)
  names <- list("Interpolation", "LocF", "Mean", "Random", "MA")
  all_rmse <- c()
  # calculate and print rmse's
  for(ind in 1:5){
    name <- names[[ind]]
    data <- datasets[[ind]]
    rmse <- sqrt(mean((data - dataset_no_missing)^2))
    print(glue("RMSE for {name} Imputation : {rmse}"))
    all_rmse <- c(all_rmse, rmse)
  }
  # print method with lowest rmse
  min_ind <- which.min(all_rmse)
  lowest_rmse <- all_rmse[min_ind]
  name_lowest_rmse <- names[[min_ind]]
  print(glue("{name_lowest_rmse} Imputation had the lowest RMSE of: {lowest_rmse}"))
  return(datasets)
}
imputed_datasets_seadec <- na_seadec_results(data_with_na, data_complete)
```

```
## [1] "Time to impute using Interpolation:"
## 0.02 sec elapsed
## [1] "Time to impute using LocF:"
## 0.01 sec elapsed
## [1] "Time to impute using Mean"
## 0 sec elapsed
## [1] "Time to impute using Random:"
## 0.02 sec elapsed
## [1] "Time to impute using MA"
## 0.06 sec elapsed
## RMSE for Interpolation Imputation : 0.82282250456891
## RMSE for LocF Imputation : 1.03269057159474
## RMSE for Mean Imputation : 3.2967081003038
## RMSE for Random Imputation : 7.52602760627937
```

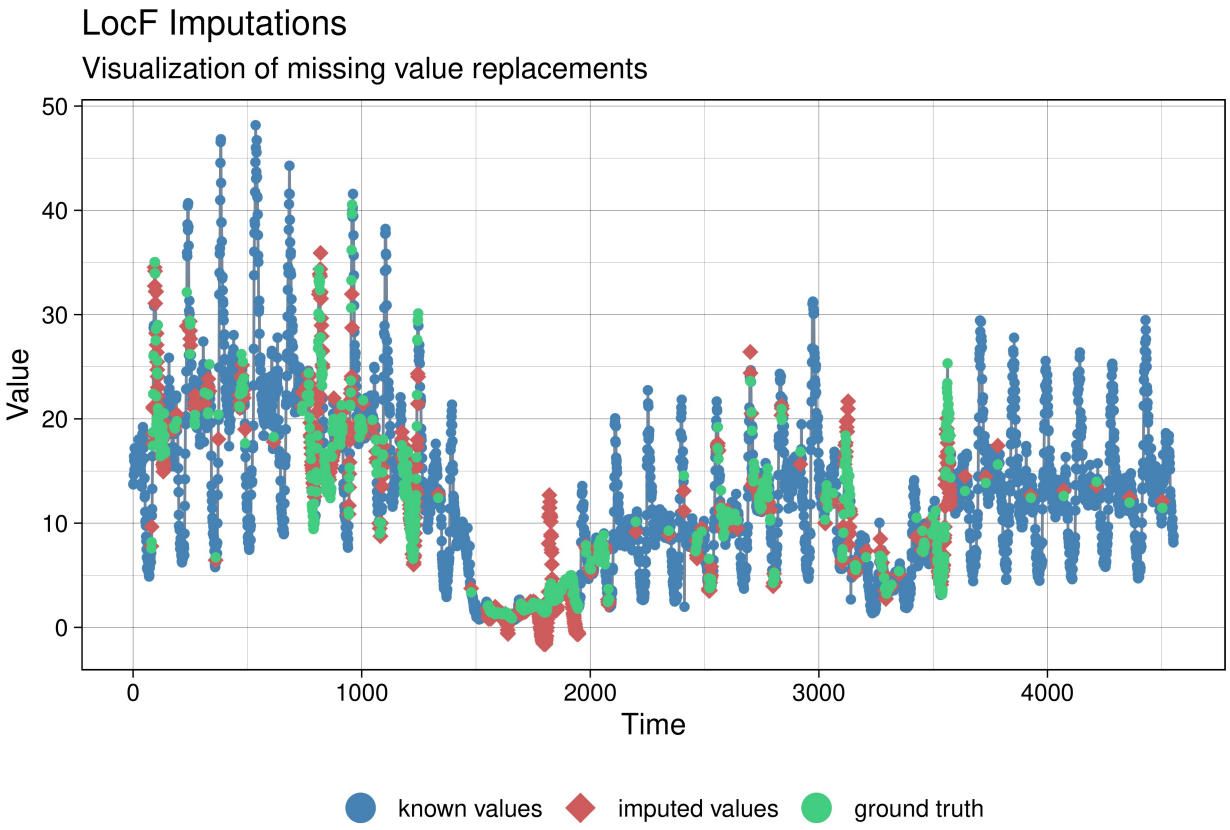
```
## RMSE for MA Imputation : 0.844414095300961
## Interpolation Imputation had the lowest RMSE of: 0.82282250456891
```

According to the results above, regular imputation worked best, with a RMSE score of 0.822, which is roughly 0.238 points more accurate than the linear interpolation in the na_interpolation model. The second best model was the MA model (imputation by weighted moving average), which also performed better than the linear interpolation in the previous model and performed only slightly worse than the regular interpolation method. Finally, we can see that the random method of interplation did not perform well at all. Let's visualize the interpolations.

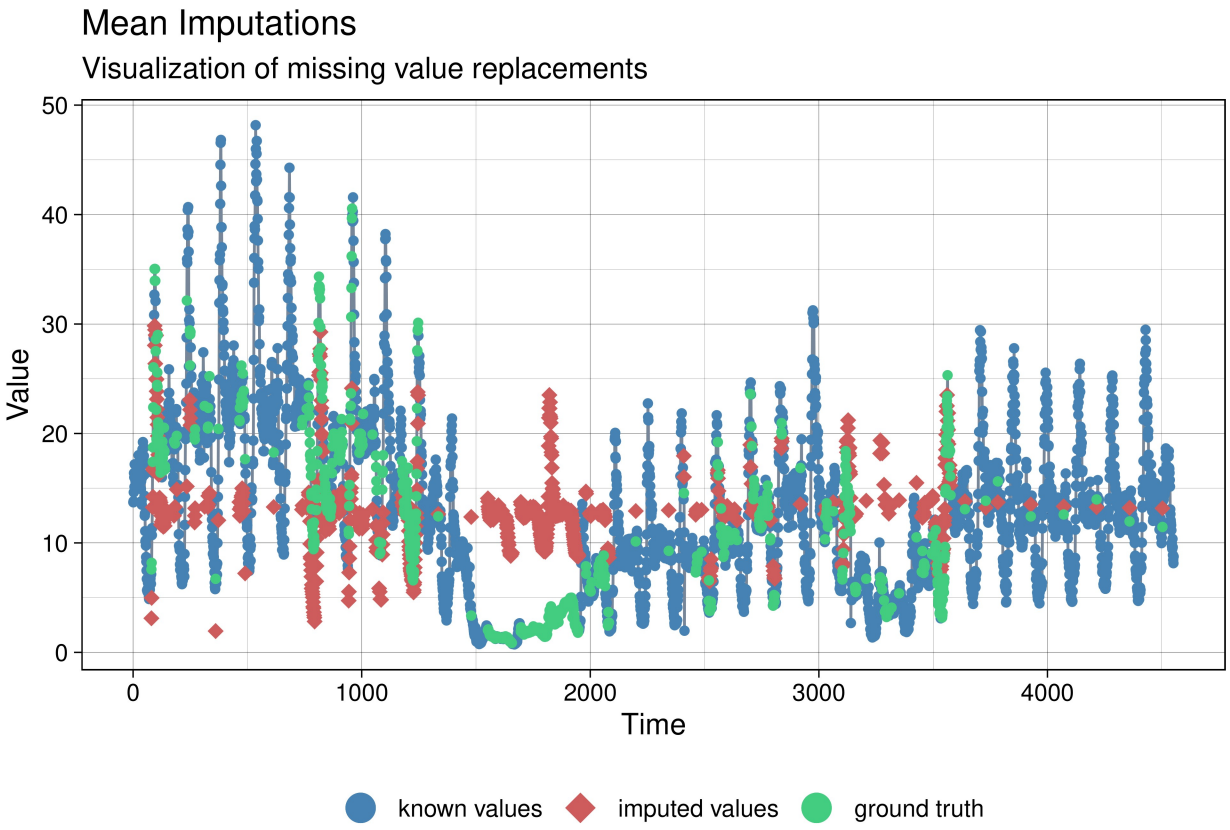
```
ggplot_na_imputations(data_with_na, imputed_datasets_seadec[[1]], data_complete, title = "Interpolation
```



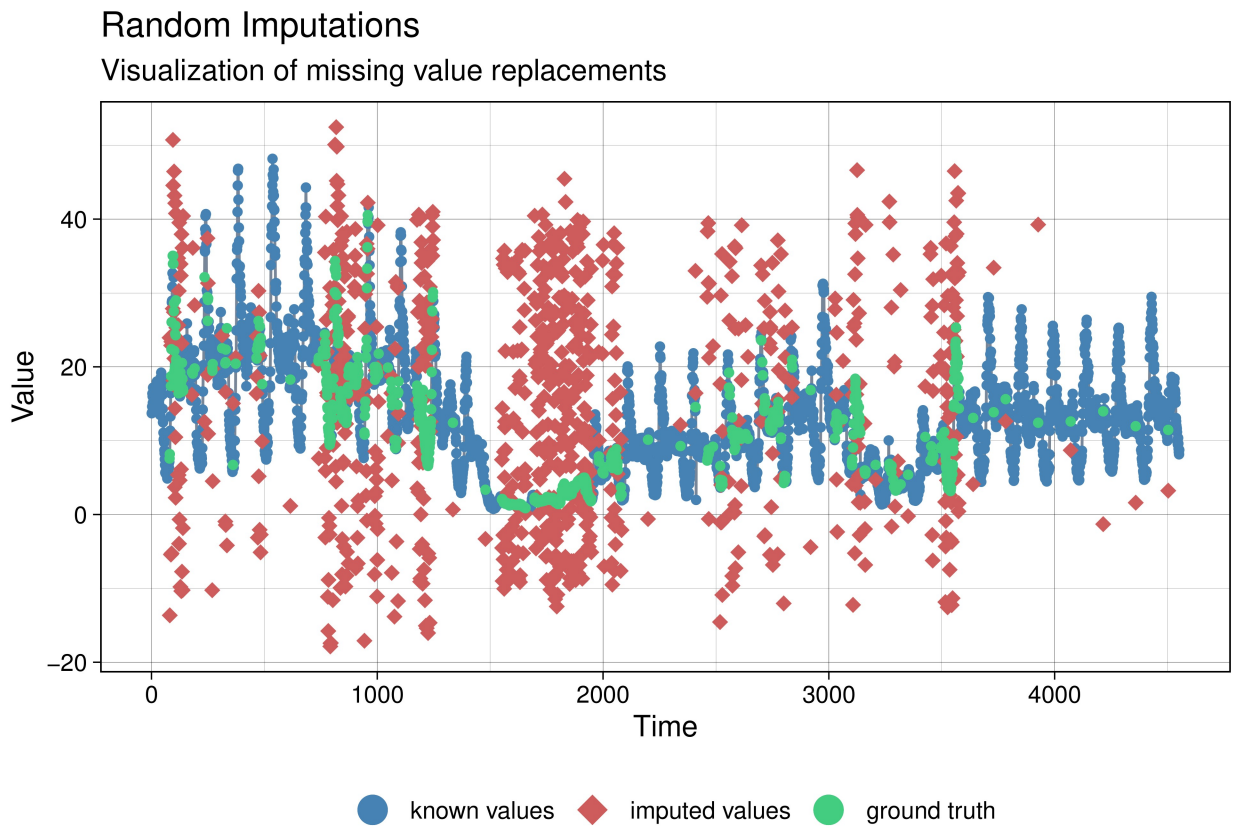

```
ggplot_na_imputations(data_with_na, imputed_datasets_seadec[[2]], data_complete, title = "LocF Imputati
```



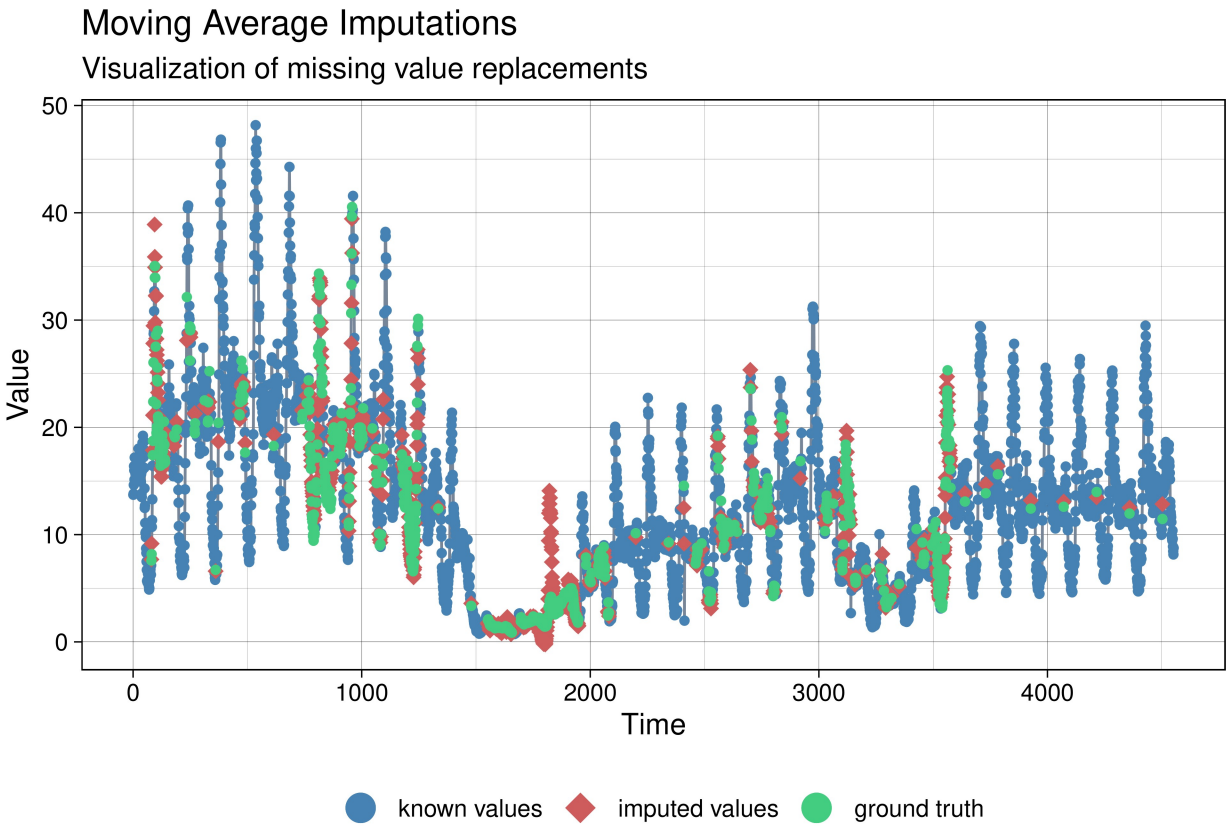

```
ggplot_na_imputations(data_with_na, imputed_datasets_seadec[[3]], data_complete, title = "Mean Imputati
```




```
ggplot_na_imputations(data_with_na, imputed_datasets_seadec[[4]], data_complete, title = "Random Imputa
```




```
ggplot_na_imputations(data_with_na, imputed_datasets_seadec[[5]], data_complete, title = "Moving Averag
```



If we consider the plots above, we can see that regular interpolation and Moving Average Imputations performed very similarly except between x = 1500 and x = 2000. The Moving Average imputations actually hugged the true value more than the regular interpolations, but had a spike in mistakes around x = 1800. This could be the reason that regular interpolation performed better on average.

Conclusion

Based on the results from this analysis, my hypthesis that spline interpolation would work best was incorrect. Spline interpolation is a model with low bias and high variance which led to the big errors in the predicted values. Linear interpolation is a high bias and low variance model that reduces the size of the errors. Even then, the weighted average interpolation performed very well comparatively and only made big mistakes when the bin of missing values was large. One way to improve this is by interpolating using weighted moving average when the chunk of missing values is small, and using linear interpolation when the chunk of missing values is large.