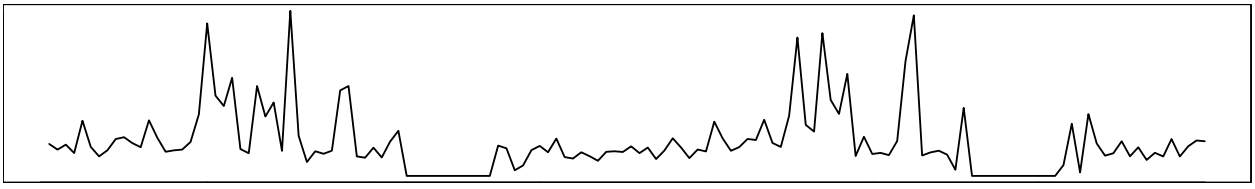
beecount-vignette

bee.count



Data Set

Our dataset was produced by an AI using digital particle image velocimetry or DPIV [Willert and Gharib, 1991] to count the level of bee activity outside a hive marked R_4_5. These csv files are part of the output of Dr. Sarbajit Mukherjee during his dissertation at Utah State University. We have received permission of Dr. Kulyukin to use these files for time series analysis. The counts correspond to a 28 second period recorded approximately every 15 minutes throughout the day. The monitors shut-down at night and went off line several times throughout the season leaving significant coverage gaps.

bee.data

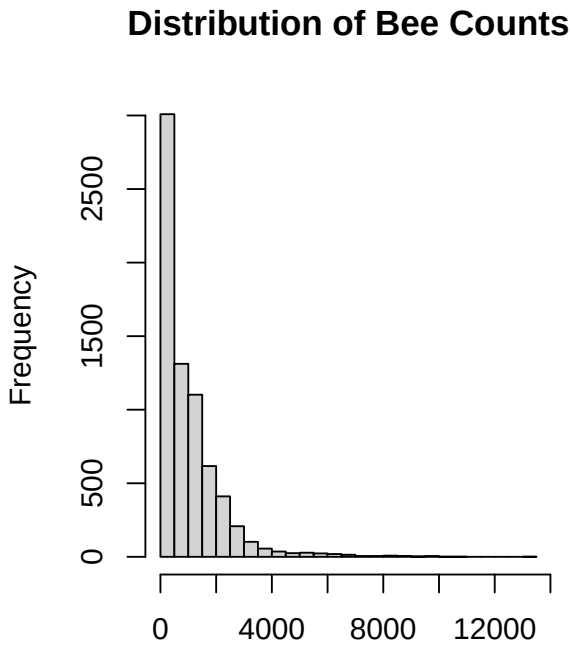
This library contains an R data file named bee.data. It is a collection of bee motion counts calculated by DPIV and paired with the weather conditions organized by timestamp.

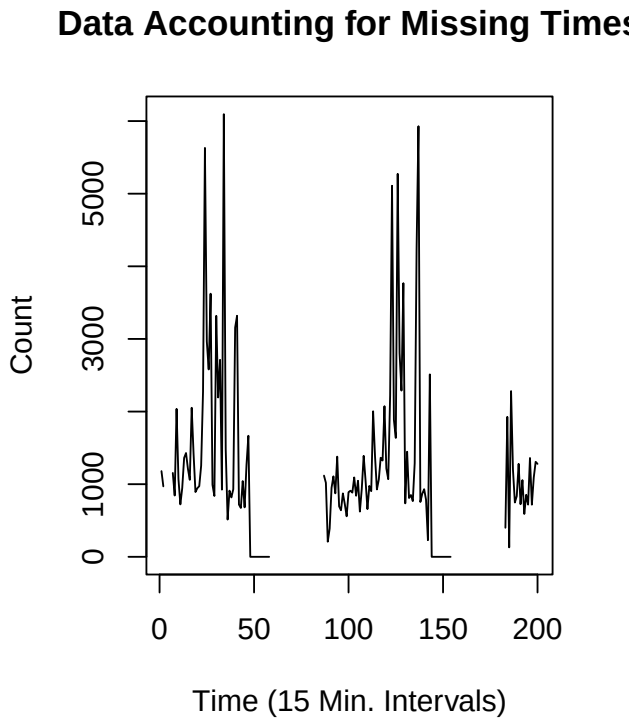
Long Description of bee.data

```
#> Bee Count Data
#>
#> Description:
#>
#>      The bee dataset was produced by an AI using digital particle image
#>      velocimetry or DPIV to count the level of bee activity outside a
#>      hive marked R_4_5. These csv files are part of the output of Dr.
#>      Sarbajit Mukherjee during his dissertation at Utah State
#>      University. We have received permission of Dr. Kulyukin to use
#>      these files for time series analysis.  The counts correspond to a
#>      28 second period recorded approximately every 15 minutes
#>      throughout the day. The monitors shutdown at night and went off
#>      line several times throughout the season leaving significant
#>      coverage gaps.  Weather was also collected from Utah State
#>      University's Environmental Observatory to correspond with the same
#>      days and times as the bee data.
#>
#>      A dataset containing the counts of bees alongside weather.
#>
#> Usage:
#>
#>      bee.data
```

```
#>
#> Format:
#>
#>      A data.table with 9896 rows and 15 variables:
#>
#>      DATE The date and time using the MDT timezone.
#>
#>      MONTH Numeric month.
#>
#>      DAY Day of the month.
#>
#>      HOUR Hour of the day. 0 corresponds to midnight.
#>
#>      MINUTE Minutes within the hour.
#>
#>      TOTAL_COUNT How many bees are present.
#>
#>      UPWARD How many bees are facing upward at the time of the count.
#>
#>      DOWNWARD How many bees are downward at the time of the count.
#>
#>      LATERAL How many bees are lateral at the time of the count.
#>
#>      WINDOW Window (in minutes) of time during the count.
#>
#>      OVELAP Any overlap (in minutes) of consecutive windows.
#>
#>      TEMP Air temperature in degrees F.
#>
#>      WIND Average windspeed in the hour (m/s).
#>
#>      WET Percentage of surface wetness (due to precipitation)
#>
#>      PRECIP Total precipitation (in the hour) in mm.
#>
#> Source:
#>
#>      <URL: https://climate.usu.edu/mchd/>
```

The R_4_5 monitor worked reasonable well during the 2017 season. However, the data collected from is was heavily right skewed and missing multiple values.





Time series

Time series data sets are most often indexed by time and store a single value for each index. A distinctive feature of time series is that the mean and variance may often be misleading as there is a significant auto correlation between a point and the preceding or following points. This makes linear models unsatisfactory and instead the data scientist must use the more poorly understood techniques of time series analysis.

Purpose

The purpose of the collected data set and associated tools is to provide a frame-work for the data scientist to better understand the processing involved in preparing a data set for time series analysis and interpreting the results.

Functions

base_impute

The base_impute function replaces missing values to a value indicated to by the programmer.

Long Description of base_impute

```
#> Basic (naive) imputation for missing values in a data.table.
#>
#> Description:
#>
#>      Basic (naive) imputation for missing values in a data.table.
```

```
#>
#> Usage:
#>
#>      base_impute(dt, type = "median", impute_col, group_by = "")
#>
#> Arguments:
#>
#>       dt: A data.table with a column of with missing values.
#>
#>     type: The type of imputation to be done; options are 'mean',
#>           'median', and 'zero'.
#>
#> impute_col: The column on which to perform imputation.
#>
#> group_by: The columns to group the imputation method by.
#>
#> Examples:
#>
#>      base_impute(bee.data, "median", "TOTAL_COUNT", c("MONTH", "DAY"))
```

sparse_ar

The sparse_ar function computes an autoregressive integrated moving average without the user selecting all the needed parameters of the stats package arima function.

Long Description of spare_ar

```
#> Sparse AR Model with Auto-Selected Coefficients
#>
#> Description:
#>
#>      Sparse AR Model with Auto-Selected Coefficients
#>
#> Usage:
#>
#>      sparse_ar(x, p_max = 30, pct_ci = 0.95, margin = 0)
#>
#> Arguments:
#>
#>        x: A time series object
#>
#>    p_max: The maximum number of coefficients to be compute
#>
#>   pct_ci: What percentage of confidence interval to be used for
#>           coefficient consideration.
#>
#>   margin: An additional margin to increase sparsity.
#>
#> Examples:
#>
#>      data <- base_impute(bee.data[1:100], impute_col = "TOTAL_COUNT")
#>      x <- data$IMPUTED_VALS
#>      model <- sparse_ma(x)
```

sparse_arma

The sparse_arma function computes an autoregressive moving average while limiting the variables needed from the user.

Long Description of sparse_arma

```
#> Sparse ARMA Model with Auto-Selected Coefficients
#>
#> Description:
#>
#>      Sparse ARMA Model with Auto-Selected Coefficients
#>
#> Usage:
#>
#>      sparse_arma(x, p_max = 10, q_max = 10, pct_ci = 0.95, margin = 0)
#>
#> Arguments:
#>
#>        x: A time series object
#>
#>    p_max: The maximum number of coefficients for the AR part of the
#>           model
#>
#>    q_max: The maximum number of coefficients for the MA part of the
#>           model
#>
#>   pct_ci: What percentage of confidence interval to be used for
#>           coefficient consideration.
#>
#>   margin: An additional margin to increase sparsity.
#>
#> Examples:
#>
#>      data <- base_impute(bee.data[1:100], impute_col = "TOTAL_COUNT")
#>      x <- data$IMPUTED_VALS
#>      model <- sparse_arma(x)
```

sparse_ma

The sparse_ma function computes an moving average for the user.

Long Description of sparse_ma

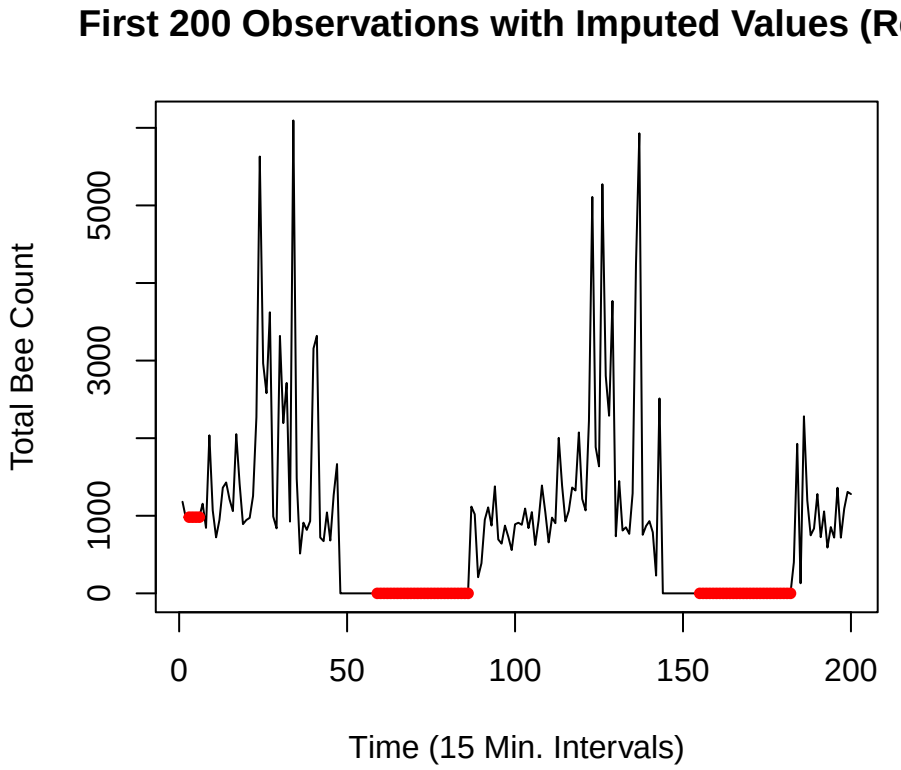
```
#> Sparse MA Model with Auto-Selected Coefficients
#>
#> Description:
#>
#>      Sparse MA Model with Auto-Selected Coefficients
#>
#> Usage:
#>
#>      sparse_ma(x, q_max = 30, pct_ci = 0.95, margin = 0)
#>
#> Arguments:
#>
#>        x: A time series object
```

```
#>
#>    q_max: The maximum number of coefficients to be compute
#>
#>   pct_ci: What percentage of confidence interval to be used for
#>           coefficient consideration.
#>
#>   margin: An additional margin to increase sparsity.
#>
#> Examples:
#>
#>      data <- base_impute(bee.data[1:100], impute_col = "TOTAL_COUNT")
#>      x <- data$IMPUTED_VALS
#>      model <- sparse_ma(x)
```

Tutorial

After the bee data is read in, the data must be inputed to be usable. After consideration, The hours between midnight and 6:00 am were decided to be imputed as a zero activity level, and other missing values to be the median for that day of the month.

```
plot(ts(data$IMPUTED_VALS[1:200]), ylab = 'Total Bee Count',
     main = "First 200 Observations with Imputed Values (Red)",
     xlab = 'Time (15 Min. Intervals)')
lines(missing_idxs, data$IMPUTED_VALS[missing_idxs], col = 'red',
      type = 'p', pch = 20)
```

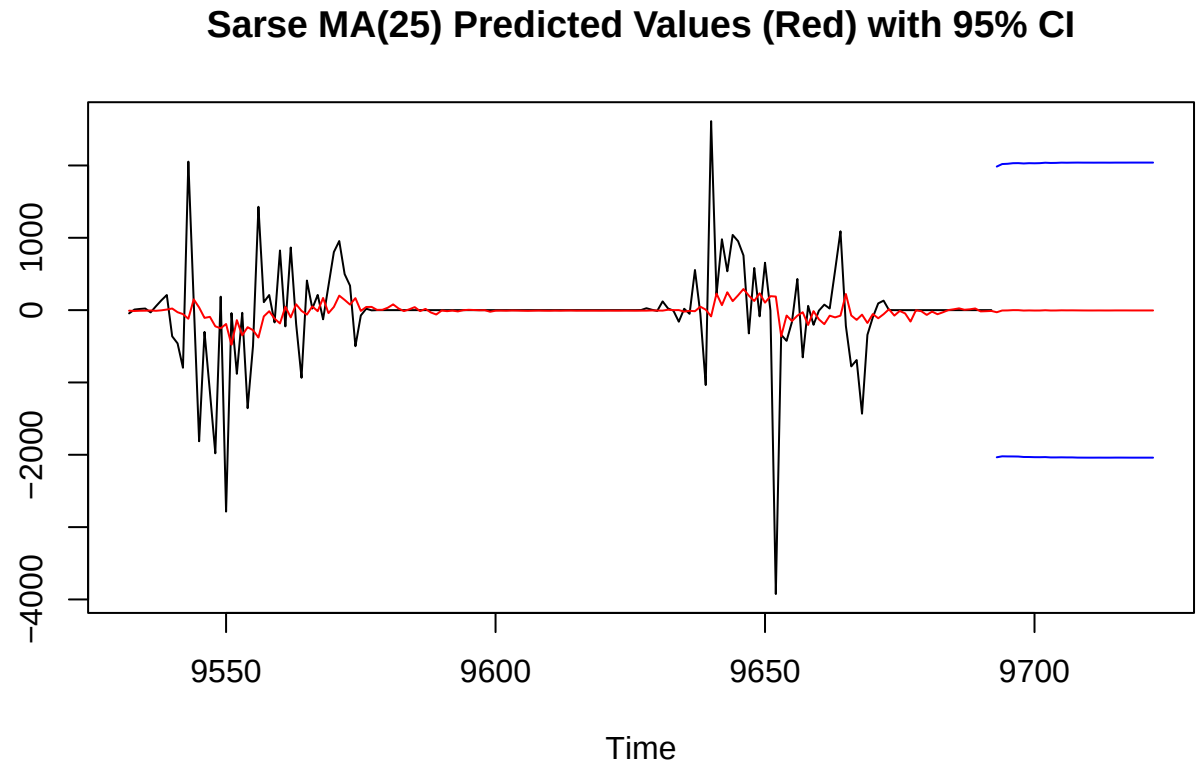


Next we take the data and separate it into training and testing groups. In this case, we chose the first three days of October to be our test group.

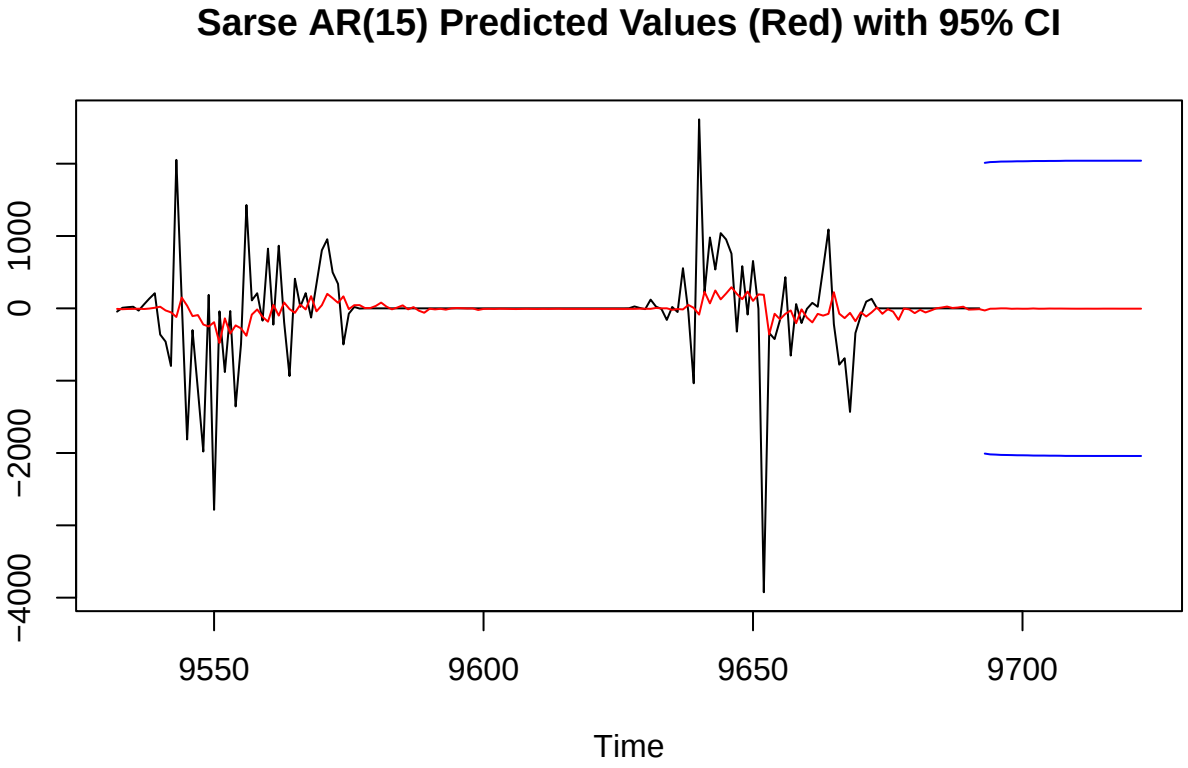
```r
# The training set is June - September
train <- data[1:9658]
train_count <- ts(train$IMPUTED_VALS)
train_count_96 <- diff(train_count, lag = 96)

# The test set is the first three days of October
test  <- data[9659:9894]
test_count <- ts(test$IMPUTED_VALS)
test_count_96 <- diff(test_count, lag = 96)
```

First we try to make predictions on the moving average approach. From the figure, we can see that there is more to this series fits a trend line.

```
#>
#>  Box-Ljung test
#>
#> data:  model$residuals
#> X-squared = 10.732, df = 36, p-value = 1
#>
#>  Box-Ljung test
#>
#> data:  model$residuals
#> X-squared = 13.003, df = 36, p-value = 0.9998
#>
#>  Box-Ljung test
#>
#> data:  model$residuals
#> X-squared = 32.418, df = 36, p-value = 0.6397
```

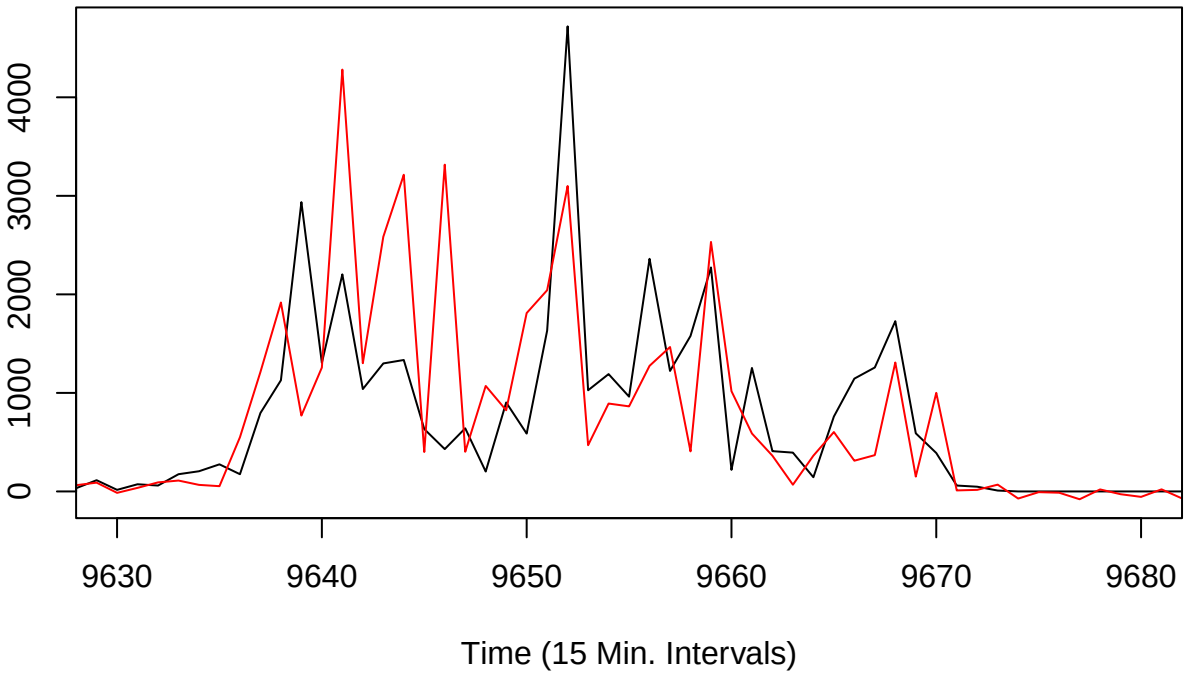


We can also use the ar model to predict the future values.



Lastly, the moving average with lag 25 can be used to make some close predictions of our final day of testing.

```
ma_orig <- diffinv(plot_vals, lag = 96, differences = 1,
                   xi = train_count[9402:9497])

ts.plot(ts(train_count[9402:9562], start = 9532), ma_orig,
        gpars = list(col = c('black', 'red'),
                     main = "MA(25) Model on Original Count Scale (Tail End)",
                     xlab = 'Time (15 Min. Intervals)',
                     xlim = c(9630, 9680)))
```


MA(25) Model on Original Count Scale (Tail End)
0
1000
2000
3000
4000
9630
9640
9650
9660
9670
9680
Time (15 Min. Intervals)