Sensible: An R Package to Make Sense of your Sensor Data

Requirements

- R v4+
- dplyr
- readr
- stringr
- stats
- ggplot2
- ggridges
- ggpmisc
- tidyr
- viridis
- rlang
- lubridate
- clifro
- lubridate
- oceanwaves
- oce
- owhlR *from github*
- mesowest *from github*

Installation

```
remotes::install_github("rfrancolini/sensible")
```

About Sensible

This package is built to aid in the processing and summarizing of various environmental sensors. Initially created to streamline the QA/QC-ing of the following sensors - (1) Onset's HOBO Temperature Logger, (2) Lowell Instrument's Tilt Current Meter, (3) Dataflow Systems' Odyssey Xtreem PAR Logger, and (4) the Open Wave Height Logger - the functions can be used on any sensors that have similar data structure. All of the previously mentioned sensors often have their own programs that pre-process data and output them in a csv format. However, one detail that the sensor-specific program may not process is when the sensor was turned on to start recording versus when the sensor was deployed. This package allows the user to define what time points they want to analyze the data for, eliminating noise at the start and stop of the data files.

This package also has multiple plotting functions to aid in the visualization of the data. Scatter plots with smoothed lines or line graphs are included for most sensors. Additionally, the temperature data can also be plotted as a ridgeline plot, and the tilt current meter data can be plotted as a wind rose diagram as a more dynamic way to visualize these data. Even if data has not been processed with the `read_...()` functions, plotting functions can be used as long as they are structured the same way as the output of the `read_...()` functions.

In addition to processing and summarizing the data gathered via the sensors, this package has functions that will generate a linear model using the sensor data and corresponding satellite data. In carrying out our

research using our sensors deployed at each site, we realized that while the sensors were providing us a more precise at-depth measurement at our specific sites than satellite data may provide (due to the coarse-grain nature of satellite data), we were lacking many days of data due to having to remove the sensors in the winter months, malfunctioning sensors, or batteries expiring partway through deployment. Instead of scrapping the sensor data we had in hand, we opted to create a model correlating satellite data for a specific site and the data gathered from the sensor deployed at this site. With the generated model, we were then able to predict what the sensor would have registered on the days where it was not in the water. This way, we were able to still use site-specific "at-depth" measurements, and have year-round data to implement in downstream applications.

Temperature Data (Onset HOBO Temperature Loggers)

This is to be used after initial downloading of data onto a computer using the HOBOware program.

Function: `read_hobotemp()`

This will function on any HOBO data that has been downloaded in csv format (not hproj format), or temperature data that has the same initial format as HOBO temperature logger data, and output a csv. Options include (1) whether you want to define a site name (if not, it will pull it from the file name), (2) if you want to define the start/stop times, have it automatically remove the first and last day (default setting), or keep all of the data, and (3) if you want to write a file with a specific output name.

Read Example Data

```
library(sensible)
x <- read_hobotemp()
x
```

```
## # A tibble: 9,025 x 5
##    Reading DateTime             Temp Intensity Site
##      <int> <dttm>              <dbl>     <dbl> <chr>
##  1      53 2021-05-15 00:04:32  8.18         0 LittleDrisko
##  2      54 2021-05-15 00:19:32  8.18         0 LittleDrisko
##  3      55 2021-05-15 00:34:32  8.08         0 LittleDrisko
##  4      56 2021-05-15 00:49:32  8.18         0 LittleDrisko
##  5      57 2021-05-15 01:04:32  8.08         0 LittleDrisko
##  6      58 2021-05-15 01:19:32  8.08         0 LittleDrisko
##  7      59 2021-05-15 01:34:32  7.98         0 LittleDrisko
##  8      60 2021-05-15 01:49:32  7.88         0 LittleDrisko
##  9      61 2021-05-15 02:04:32  7.88         0 LittleDrisko
## 10      62 2021-05-15 02:19:32  7.88         0 LittleDrisko
## # i 9,015 more rows
```

Read Example Data with User Defined Site

```
library(sensible)
x <- read_hobotemp(site = "Little Drisko Island")
x
```

```
## # A tibble: 9,025 x 5
##    Reading DateTime             Temp Intensity Site
##      <int> <dttm>              <dbl>     <dbl> <chr>
##  1      53 2021-05-15 00:04:32  8.18         0 Little Drisko Island
##  2      54 2021-05-15 00:19:32  8.18         0 Little Drisko Island
##  3      55 2021-05-15 00:34:32  8.08         0 Little Drisko Island
##  4      56 2021-05-15 00:49:32  8.18         0 Little Drisko Island
##  5      57 2021-05-15 01:04:32  8.08         0 Little Drisko Island
##  6      58 2021-05-15 01:19:32  8.08         0 Little Drisko Island
##  7      59 2021-05-15 01:34:32  7.98         0 Little Drisko Island
##  8      60 2021-05-15 01:49:32  7.88         0 Little Drisko Island
##  9      61 2021-05-15 02:04:32  7.88         0 Little Drisko Island
## 10      62 2021-05-15 02:19:32  7.88         0 Little Drisko Island
## # i 9,015 more rows
```

Read Example Data With User Defined Start/Stop Dates

```
ss <- as.POSIXct(c("2021-05-20", "2021-06-01"), tz = "UTC")
xud <- read_hobotemp(clipped = "user", startstop = ss)
xud
```

```
## # A tibble: 1,152 x 5
##    Reading DateTime             Temp Intensity Site
##      <int> <dttm>              <dbl>     <dbl> <chr>
##  1     533 2021-05-20 00:04:32  8.38         0 LittleDrisko
##  2     534 2021-05-20 00:19:32  8.48         0 LittleDrisko
##  3     535 2021-05-20 00:34:32  8.48         0 LittleDrisko
##  4     536 2021-05-20 00:49:32  8.48         0 LittleDrisko
##  5     537 2021-05-20 01:04:32  8.58         0 LittleDrisko
##  6     538 2021-05-20 01:19:32  8.58         0 LittleDrisko
##  7     539 2021-05-20 01:34:32  8.68         0 LittleDrisko
##  8     540 2021-05-20 01:49:32  8.68         0 LittleDrisko
##  9     541 2021-05-20 02:04:32  8.78         0 LittleDrisko
## 10     542 2021-05-20 02:19:32  8.78         0 LittleDrisko
## # i 1,142 more rows
```

Function: `summarize_hobotemp()`

Create a dataframe summarizing the temperature data, will group by site if more than one present.

Summarize data, csv with one site

```
sum1 <- summarize_hobotemp(x)
print.data.frame(sum1)
```

```
##                   Site mean.temp           first.day            last.day
## 1 Little Drisko Island  11.30466 2021-05-15 00:04:32 2021-08-17 00:04:32
##   max.temp       max.temp.date min.temp       min.temp.date
## 1   16.046 2021-08-08 22:34:32    7.682 2021-05-15 07:34:32
```

Summarize data, csv with more than one site

```
sum2 <- summarize_hobotemp(example_ridge_data())
```

```
## Rows: 26018 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): Site
## dbl  (3): Reading, Temp, Intensity
## dttm (1): DateTime
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
print.data.frame(sum2)
```

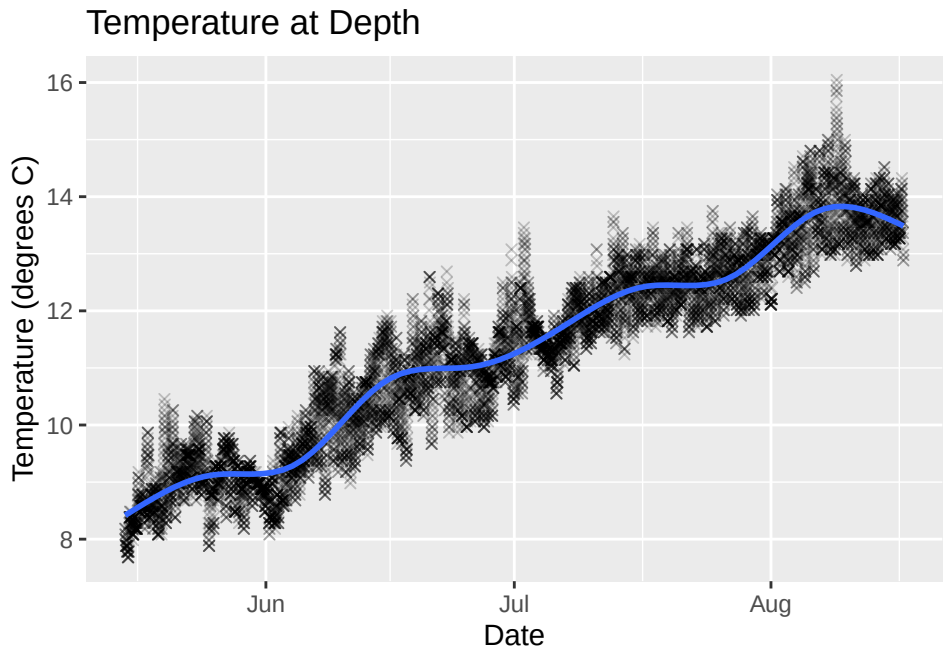
```
##           Site mean.temp           first.day            last.day max.temp
## 1     Cape Liz  12.95992 2021-06-05 00:11:44 2021-07-20 23:56:44   17.379
## 2  Damariscove  12.58830 2021-06-05 00:04:55 2021-07-20 23:49:55   17.475
## 3 Halfway Rock  12.35469 2021-06-05 00:14:09 2021-07-20 23:59:09   17.760
## 4 Isle Au Haut  11.61809 2021-06-05 00:00:00 2021-07-21 00:00:00   13.846
## 5     Marshall  11.38961 2021-06-10 00:00:00 2021-07-21 00:00:00   13.269
##         max.temp.date min.temp       min.temp.date
## 1 2021-07-18 20:41:44    9.571 2021-06-21 09:56:44
## 2 2021-07-16 19:04:55    8.680 2021-06-09 12:49:55
## 3 2021-07-18 18:14:09    9.077 2021-06-18 09:59:09
## 4 2021-07-08 10:00:00    8.779 2021-06-08 07:00:00
## 5 2021-07-16 18:45:00    9.077 2021-06-10 05:15:00
```

Function: `draw_temp_scatter_plot()`

This function will read in a csv of your HOBO data and draw a scatter plot with an overlapping trendline.

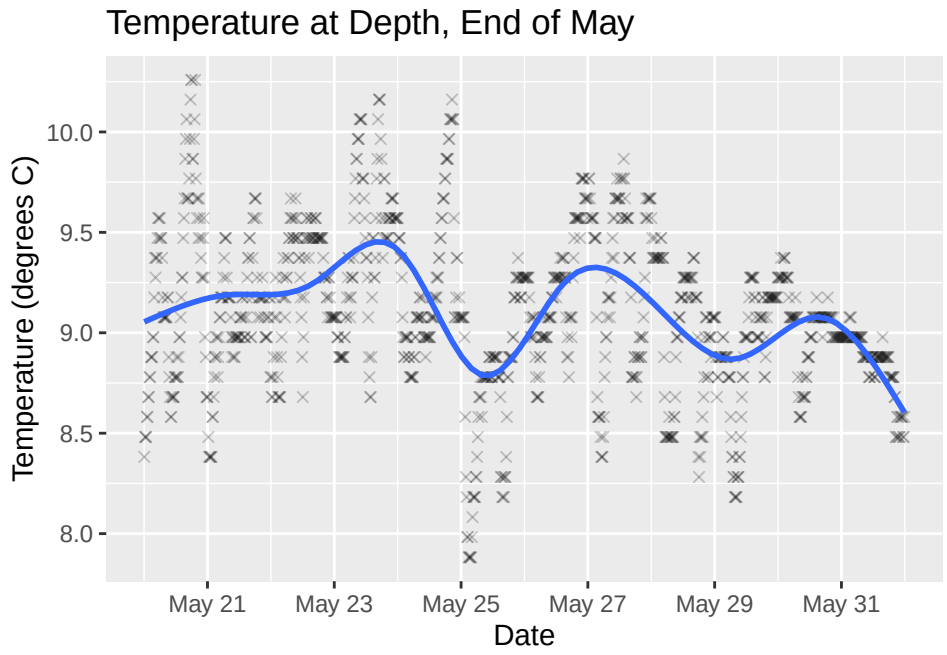
Draw Example Scatter Plot

```
tempplot_x <- draw_temp_scatter_plot(x)
```



Draw Example Scatter Plot with User Defined Start/Stop Dates and Title

```
tempplot_xud <- draw_temp_scatter_plot(xud, main = "Temperature at Depth, End of May")
```

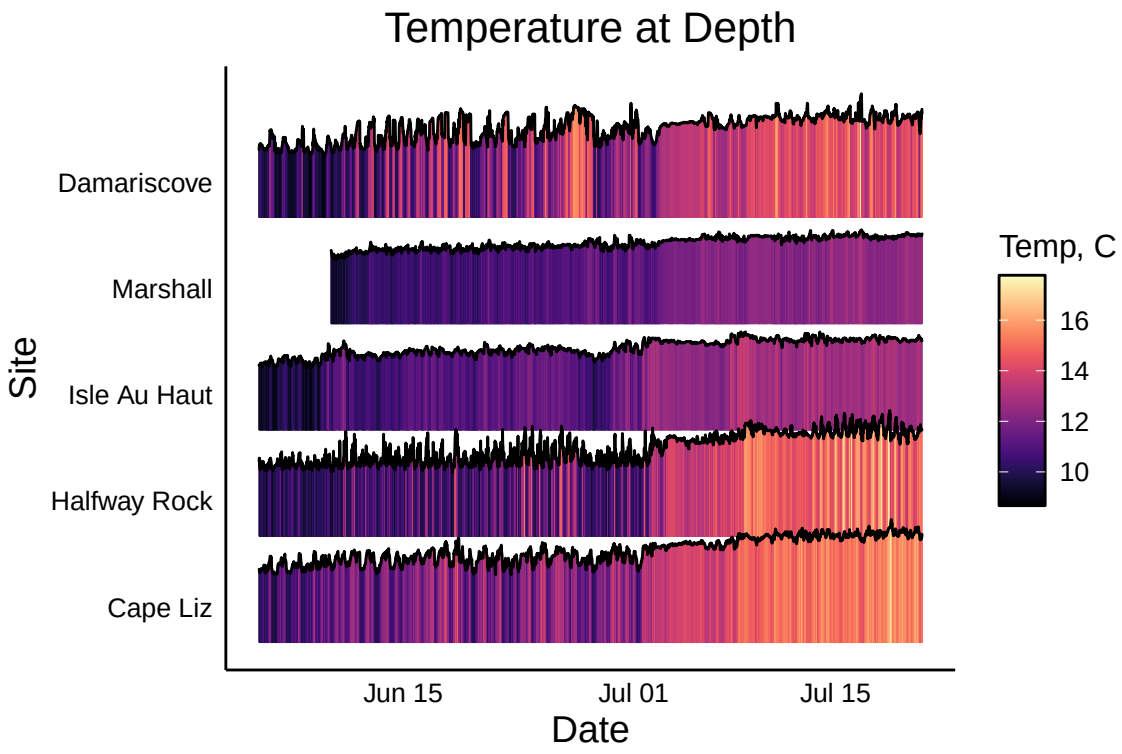


Function: `draw_ridgeline_plot()`

This function will create a color-coded ridgeline plot, adapted from here. This will have a line representing the temperature, as well as the space underneath the line colored in using a temperature gradient color scheme. Sites will appear in order they are in dataframe, unless "ordered" option is used.

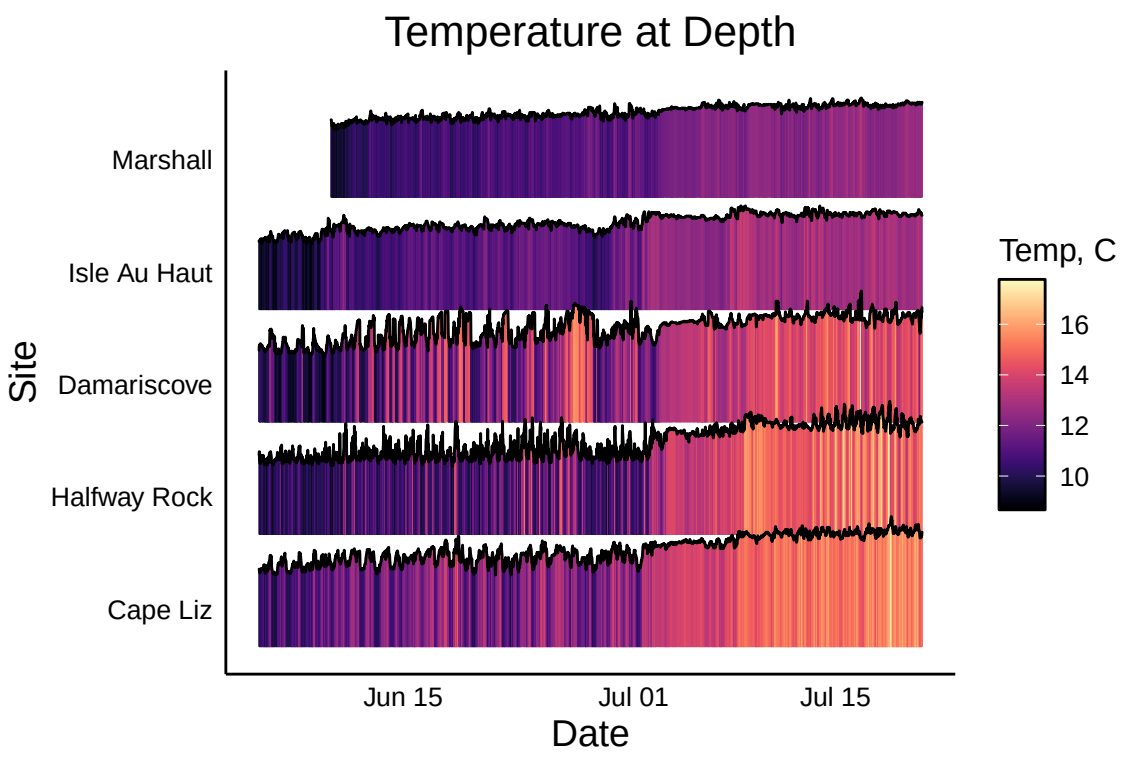
Draw example ridgeline plot

```
ridgelineplot <- draw_ridgeline_plot()
```



Draw example ridgeline plot with specified site order (south to north)

```
SiteOrder <-  c("Cape Liz", "Halfway Rock", "Damariscove", "Isle Au Haut", "Marshall")
ridgelineplot0 <- draw_ridgeline_plot(ordered = SiteOrder)
```

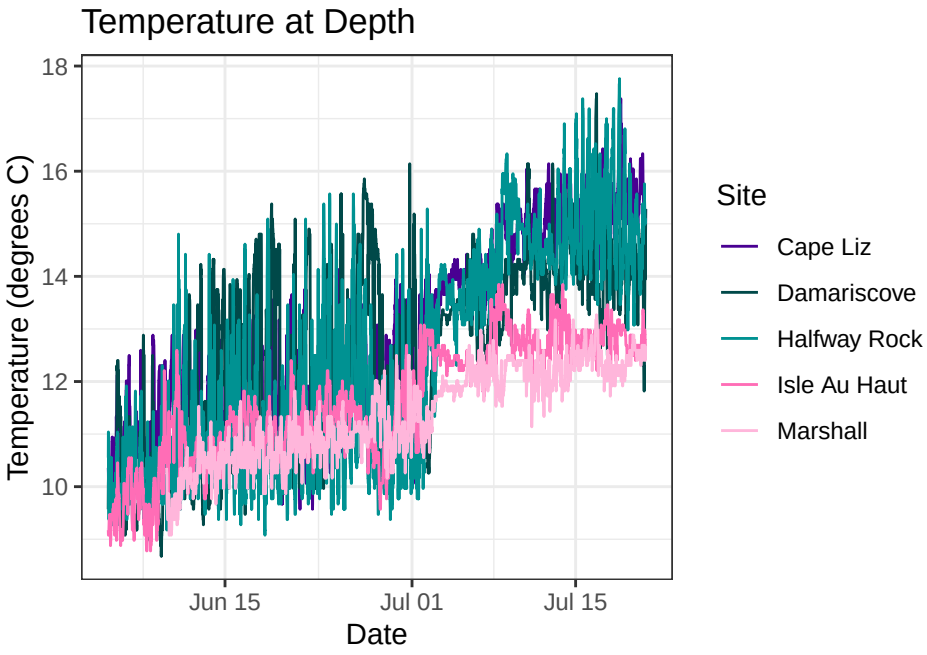


Function: `draw_temp_line_plot()`

This function will create a line plot, with a colorblind friendly color pallette, and can currently take up to 15 sites at once. Sites will appear in legend (or will be facetted upon) in order they are in dataframe, unless "ordered" option is used.

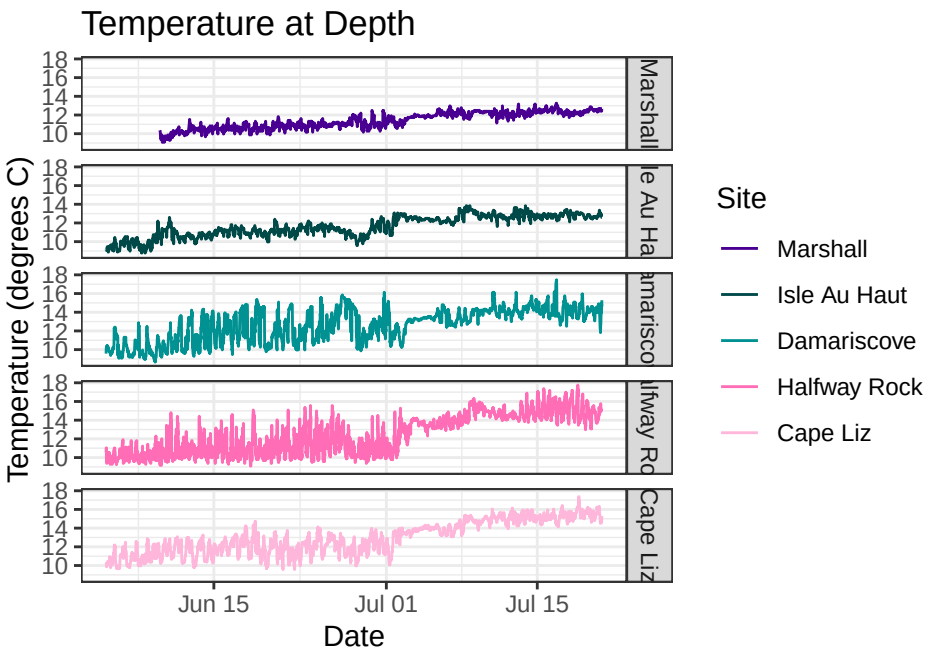
Draw example line plot

```
lineplot <- draw_temp_line_plot()
```



Draw example line plot facetted with specified site order (south to north)

```
SiteOrder <-  c("Cape Liz", "Halfway Rock", "Damariscove", "Isle Au Haut", "Marshall")
lineplot0 <- draw_temp_line_plot(ordered = SiteOrder, facet = "Site")
```



Tilt-Current Meter Data (Lowell Instruments Tilt-Current Meter)

This is to be used after initial downloading of data onto a computer using the Domino program.

Function: `read_tiltometer()`

This function will take in raw tilt current meter data and output a csv. Options include (1) whether you want to define a site name (if not, it will pull it from the file name), (2) if you want to define the start/stop times, have it automatically remove the first and last day (default setting), or keep all of the data, and (3) if you want to write a file with a specific output name. Example for the basic function is below, to see examples implementing defined start/stop times or specific sites, see read_hobotemp() documentation above.

Read Example Data

```
library(sensible)
x <- read_tiltometer()
x
```

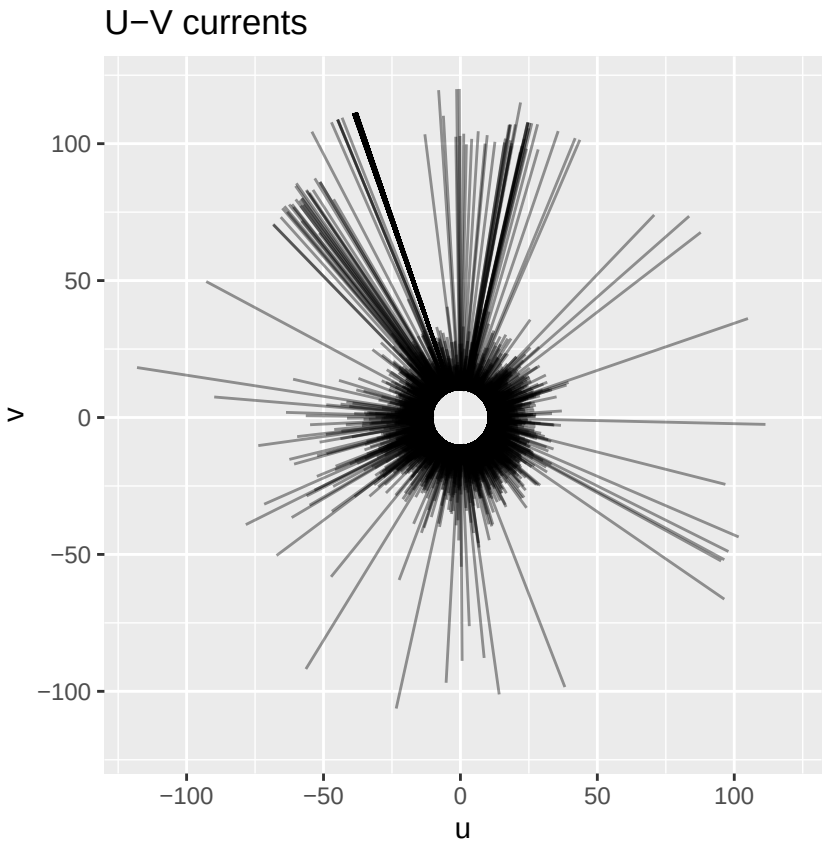
```
## # A tibble: 87,121 x 5
##    DateTime            speed   dir     v     u
##    <dttm>              <dbl> <dbl> <dbl> <dbl>
##  1 2022-04-16 00:00:00  3.76  18.5  3.57  1.2
##  2 2022-04-16 00:02:00  7.52 320.   5.77 -4.82
##  3 2022-04-16 00:04:00  7.07 353.   7.02 -0.87
##  4 2022-04-16 00:06:00  5.26 118.  -2.47  4.65
##  5 2022-04-16 00:08:00  8.7  348.   8.5  -1.85
##  6 2022-04-16 00:10:00  8.45 228.  -5.62 -6.31
##  7 2022-04-16 00:12:00  8.22 117.  -3.68  7.35
##  8 2022-04-16 00:14:00  6.61 165.  -6.37  1.75
##  9 2022-04-16 00:16:00  0.91 355.   0.91 -0.08
## 10 2022-04-16 00:18:00  2.41  56.6  1.32  2.01
## # i 87,111 more rows
```

Function: `draw_uv()`

This will produce a basic UV plot of the tilt current data.

Draw example UV plot

```
uv <- draw_uv(x)
```

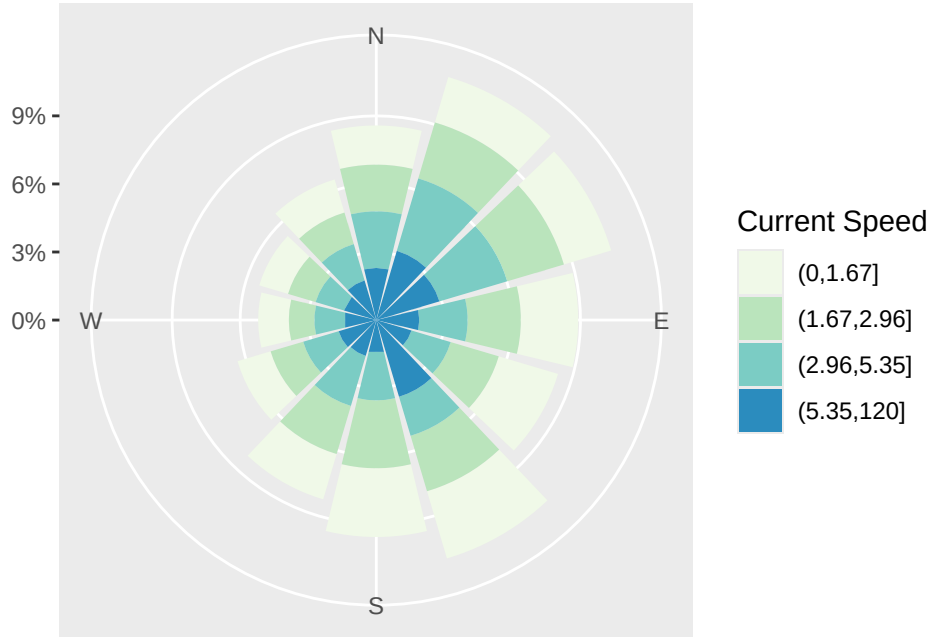


Function: `tiltometer_rose()`

This will produce a windrose plot of the tilt current data

Draw example windrose plot

```
tiltometer_rose(x, speed.cuts = "quantile-4")
```



Wave Logger Data (Open Wave Height Loggers)

This is to be used after the initial downloading of the zip files for an OWHL. The code to process the data is based on owhlR. To complete the processing of wave height data, the following three functions must be run sequentially, as it requires reading in the raw wave data, reading in air pressure data (based on mesowest), and using these two data streams to calculate wave pressure. For the ease of an example, we use a previously dowloaded air pressure dataset in the following example, however, the "read_airpressure()" function will aid in grabbing data from the mesowest dataset.

Functions: `read_wavelogger()`, `read_airpressure()`, `interp_swpressure()`, `wave_stats()`

Read Example Data

```
library(sensible)
x <- read_wavelogger()
head(x)
```

```
## # A tibble: 6 x 3
##   DateTime            Pressure.mbar TempC
##   <dttm>                      <dbl> <dbl>
## 1 2021-05-16 00:00:00         1556.  7.91
## 2 2021-05-16 00:00:00         1556.  7.9
## 3 2021-05-16 00:00:00         1556.  7.89
## 4 2021-05-16 00:00:00         1556   7.89
## 5 2021-05-16 00:00:01         1556   7.89
## 6 2021-05-16 00:00:01         1556.  7.89
```

Read Air Pressure Data Example

```
a <- example_airpressure()
head(a)
```

```
##              DateTime sea_pressure.mbar
## 1 2021-05-16 00:00:00           1022.05
## 2 2021-05-16 00:05:00           1022.05
## 3 2021-05-16 00:10:00           1022.05
## 4 2021-05-16 00:15:00           1022.05
## 5 2021-05-16 00:20:00           1022.06
## 6 2021-05-16 00:25:00           1022.06
```

Calculate Wave Statistcs

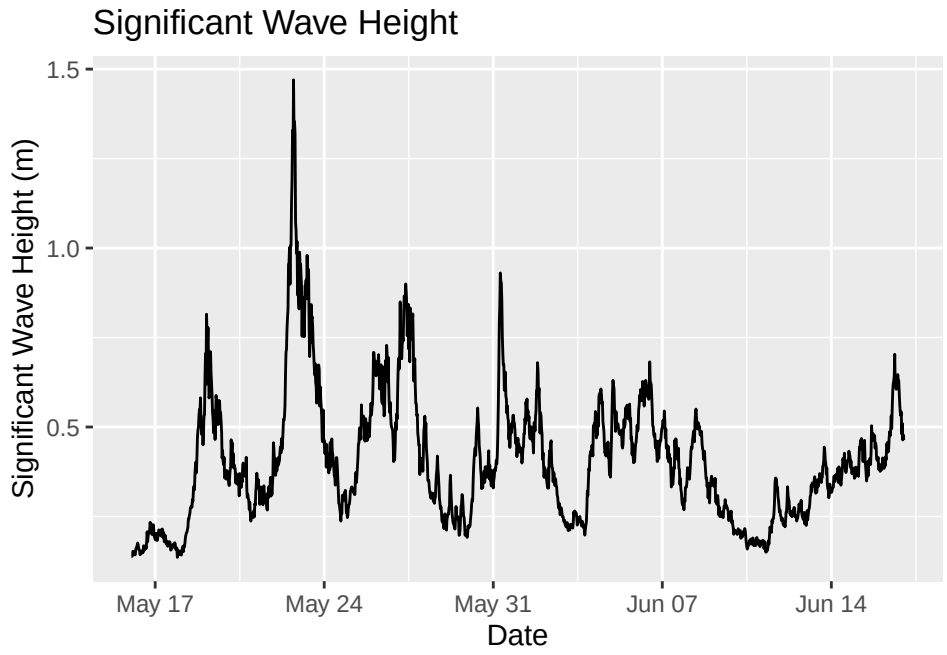
```
i <- interp_swpressure(wavelogger = x, airpressure = a)

w <- wave_stats(wavelogger = mbar_to_elevation(wavelogger = i))

head(w)
```

```
##          h       Hm0        Tp          m0    T_0_1    T_0_2      EPS2
## 1 5.311525 0.1391045 11.428571 0.001209379 7.467398 6.625700 0.5198161
## 2 5.400873 0.1385070 11.612903 0.001199012 7.563648 6.702864 0.5228120
## 3 5.577315 0.1414778 10.434783 0.001250998 7.327396 6.485607 0.5257691
## 4 5.830439 0.1529073 12.413793 0.001461291 7.138421 6.348096 0.5142914
## 5 6.168788 0.1464905 12.000000 0.001341216 6.627873 5.910600 0.5073796
## 6 6.532976 0.1492757  8.674699 0.001392702 6.299749 5.659775 0.4888088
##        EPS4            DateTime
## 1 0.7483700 2021-05-16 00:30:00
## 2 0.7543657 2021-05-16 01:00:00
## 3 0.7467871 2021-05-16 01:30:00
## 4 0.7373126 2021-05-16 02:00:00
## 5 0.7135647 2021-05-16 02:30:00
## 6 0.6908076 2021-05-16 03:00:00
```

Function: `wavespec_plot()`

Significant Wave Height Example Plot

```
wave_plot <- wavespec_plot(w)
```



PAR Data (PAR Odyssey Xtreem)

This is to be used after initial downloading of data onto a computer using the Xtract program.

Function: `read_parXtreem()`

This function will take in PAR data and output a csv. Options include (1) whether you want to define a site name (if not, it will pull it from the file name), (2) if you want to define the start/stop times, have it automatically remove the first and last day (default setting), or keep all of the data, and (3) if you want to write a file with a specific output name. Example for the basic function is below, to see examples implementing defined start/stop times or specific sites, see read_hobotemp() documentation above.

Read Example Data

```
library(sensible)
x <- read_parXtreem()
x
```

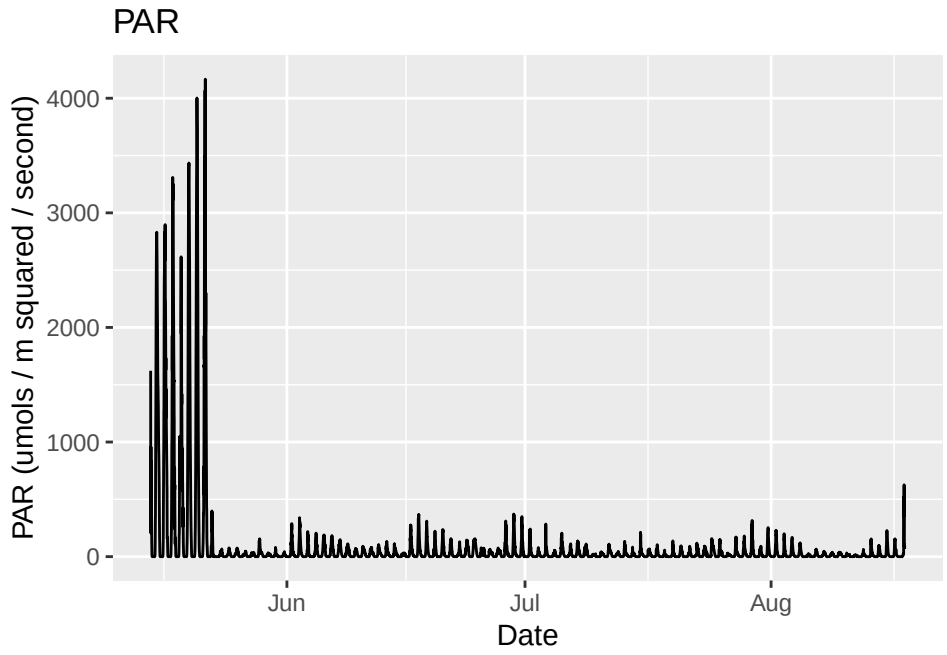
```
## # A tibble: 9,121 x 5
##     Temp   PAR ID           logDateTime DateTime
##    <dbl> <dbl> <chr>              <dbl> <dttm>
##  1  7.19 1622. FE23BC74DC01  1621022400 2021-05-14 20:00:00
##  2  7.13  739. FE23BC74DC01  1621023300 2021-05-14 20:15:00
##  3  7.13  206. FE23BC74DC01  1621024200 2021-05-14 20:30:00
##  4  7.19  957. FE23BC74DC01  1621025100 2021-05-14 20:45:00
##  5  7.19  443. FE23BC74DC01  1621026000 2021-05-14 21:00:00
##  6  7.25  635. FE23BC74DC01  1621026900 2021-05-14 21:15:00
##  7  7.25  834. FE23BC74DC01  1621027800 2021-05-14 21:30:00
##  8  7.38  695. FE23BC74DC01  1621028700 2021-05-14 21:45:00
##  9  7.44  433. FE23BC74DC01  1621029600 2021-05-14 22:00:00
## 10  7.5   364. FE23BC74DC01  1621030500 2021-05-14 22:15:00
## # i 9,111 more rows
```

Function: `draw_par_plot()`

This function will read in a csv of your PAR data and draw a line plot.

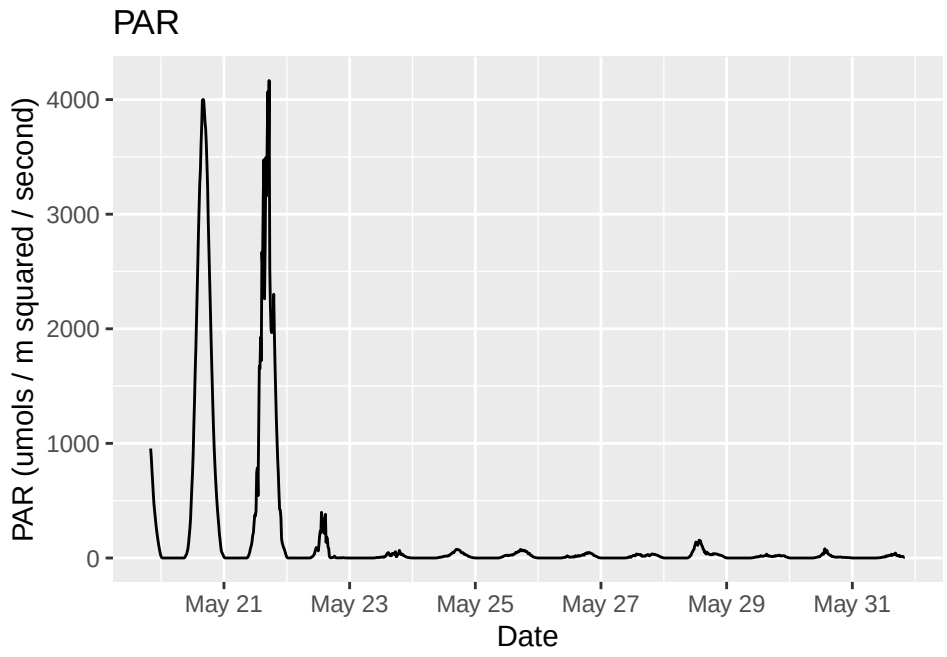
Draw Example Plot

```
parplot_x <- draw_par_plot(x)
```



Draw Example Plot User Defined Start/Stop Dates

```
ss <- as.POSIXct(c("2021-05-20", "2021-06-01"), tz = "UTC")
xud <- read_parXtreem(clipped = "user", startstop = ss)

parplot_xud <- draw_par_plot(xud)
```



Generating Sensor-Satellite Models

The generation of these models is dependent on the user supplying satellite data they have downloaded and formatted as two columns - (1) date/time, (2) environmental parameter of interest. We have previously used this on temperature data using MURSST satellite dataset and ECMWF wave dataset. To complete the model generating and prediction process, the following three functions need to be run sequentially.

Functions: `create_model_data()`, `create_model()`, `predict_data()`

```
modeldat <- create_model_data()
```

```
## Rows: 30104 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (1): Temp
## dttm (1): DateTime
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1384 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (1): temp
## dttm (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(modeldat)
```

```
## # A tibble: 6 x 3
##   DateTime            sensor satellite
##   <dttm>               <dbl>     <dbl>
## 1 2023-01-01 00:00:00   8.54      6.78
## 2 2023-01-02 00:00:00   8.42      6.34
## 3 2023-01-03 00:00:00   8.44      6.16
## 4 2023-01-04 00:00:00   8.43      6.35
## 5 2023-01-05 00:00:00   8.30      6.56
## 6 2023-01-06 00:00:00   8.02      5.96
```

```
model <- create_model(modeldata = modeldat)
```

```
summary(model)
```

```
##
## Call:
## stats::lm(formula = sensor ~ satellite, data = modeldata.Train)
##
## Residuals:
```

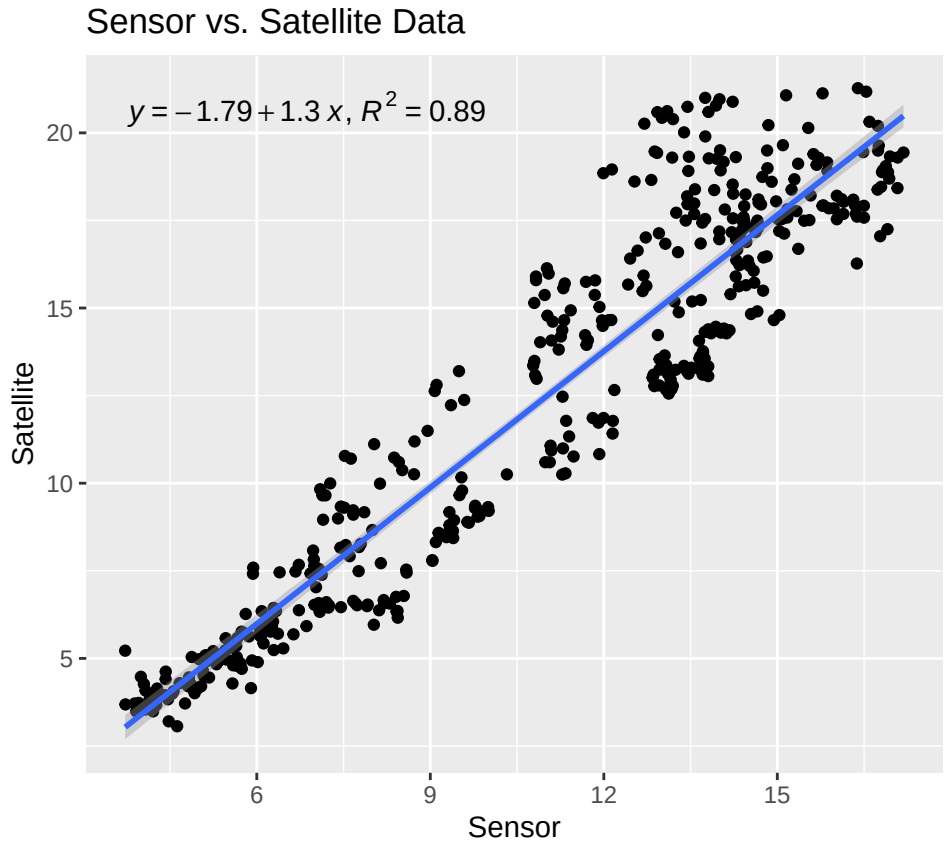
```
##     Min      1Q  Median      3Q     Max
## -3.4345 -0.8899 -0.1959  1.1662  2.8000
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.52823    0.17525   14.43   <2e-16 ***
## satellite    0.67160    0.01294   51.89   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.329 on 361 degrees of freedom
## Multiple R-squared:  0.8818, Adjusted R-squared:  0.8814
## F-statistic:  2692 on 1 and 361 DF,  p-value: < 2.2e-16
```

```
modelresult <- predict_data(model = model, modeldata = modeldat)
head(modelresult)
```

```
##     DateTime   sensor satellite predicted
## 1 2023-01-01 8.541375  6.782007  7.083055
## 2 2023-01-02 8.416854  6.337000  6.784186
## 3 2023-01-03 8.437542  6.162012  6.666663
## 4 2023-01-04 8.433583  6.353998  6.795602
## 5 2023-01-05 8.302479  6.563989  6.936633
## 6 2023-01-06 8.023813  5.960992  6.531658
```

Function: `draw_satsensor_plot()`

This will plot your satellite data and sensor data to see the relationship between the two datasets



Function: `draw_model_plot()`

This will plot your two datastreams over time, as well as your predicted data from the model