Weekly Progress Report

Agrotech Live

2025-04-18

Abstract

In an ongoing effort to achieve food sovereignty, Wiggle Labs has developed Agrotech Live to monitor soil health of different plants and crops. This tool collects four data points; Temperature, Moisture, Light and Conductivity from sensors placed near a subject. Training data for this program are ideal conditions for the subject, and performance of the experiment is based on how close collected sensor (testing) data is to the input care training data. Input data is identical in structure to the testing data (collected during a session), except it represents only ideal conditions for a subject.

This report examines relationships between the session and training data features over the past 7 days, and forecasts the next 3 days. Knowing these insights can advise on how to better adjust an environment for a subject during the session. Clustering will be used to prepare experiment data for classifying the analyzed session as "Above,"Below" or "Within" and ideal range.

System Understanding

The program collects feature data via BLE (Bluetooth Low Energy) through the sensors mentioned previously. This feature data is used to generate raw session data in `sesh.json`. It is also saved to individual daily files in `/files/read_files`.

Using `batch-builder.py` found in `/tools`, input parameters for a subjects' ideal environment can be generated. If any input file is loaded to the `tools`, it can also be auto-inserted with the command `batch-builder.py < input-file.txt`.

Data Understanding

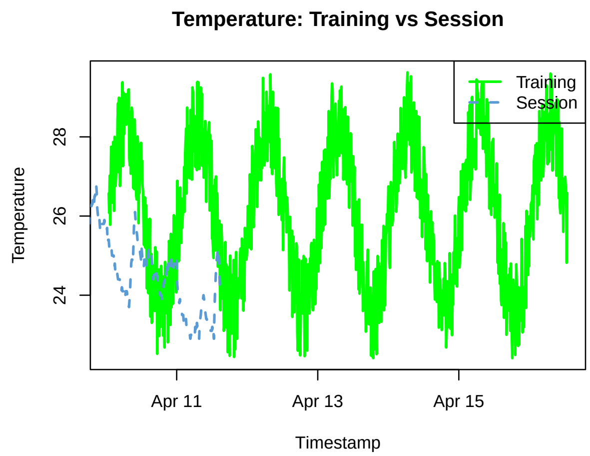
Temperature is measured in Celsius (°C) and includes a tenth decimal (eg. "17.5"). It generally affects soil activity and how nutrients are absorbed. Moisture is measured as a percentage, and depending on the subject's needs, thresholds will vary. Light is collected in lux where 1 =lux is equal to 1 lumen/m^2. Conductivity is measured in µS/cm which is a electrical conductivity. It is essentially how easily a current passes through the soil.

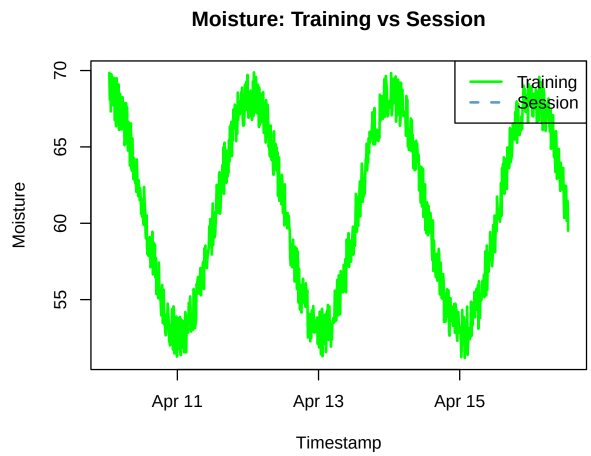
There are often patterns such as Temperature increasing as light does, or decreasing as Moisture increases. These kinds of relationships are to be expected from the laws of nature In later versions of this report, it is planned to train this analysis to learn these patterns so that insights are more clearly visible.

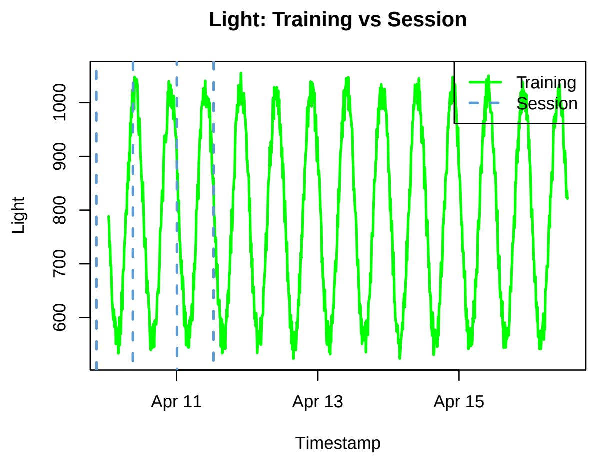
Stage I: Time Series, Correlation, Covariance and Heatmap

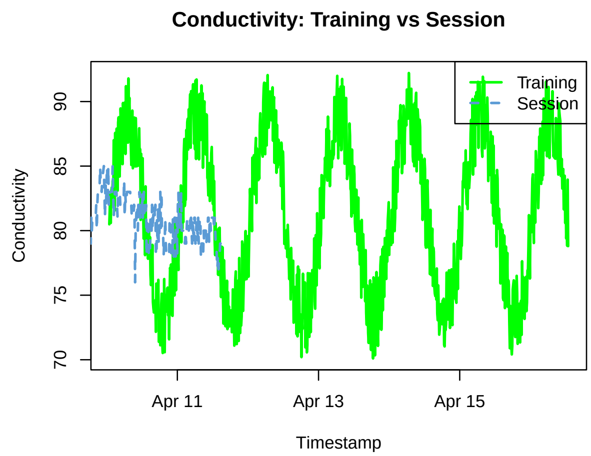
Time Series Comparison

The following graphs represent how the session and training data has changed over time. They are aligned with each other on Timestamp over the past 7 days. Input data determines the shape of training graph in this time series analysis.







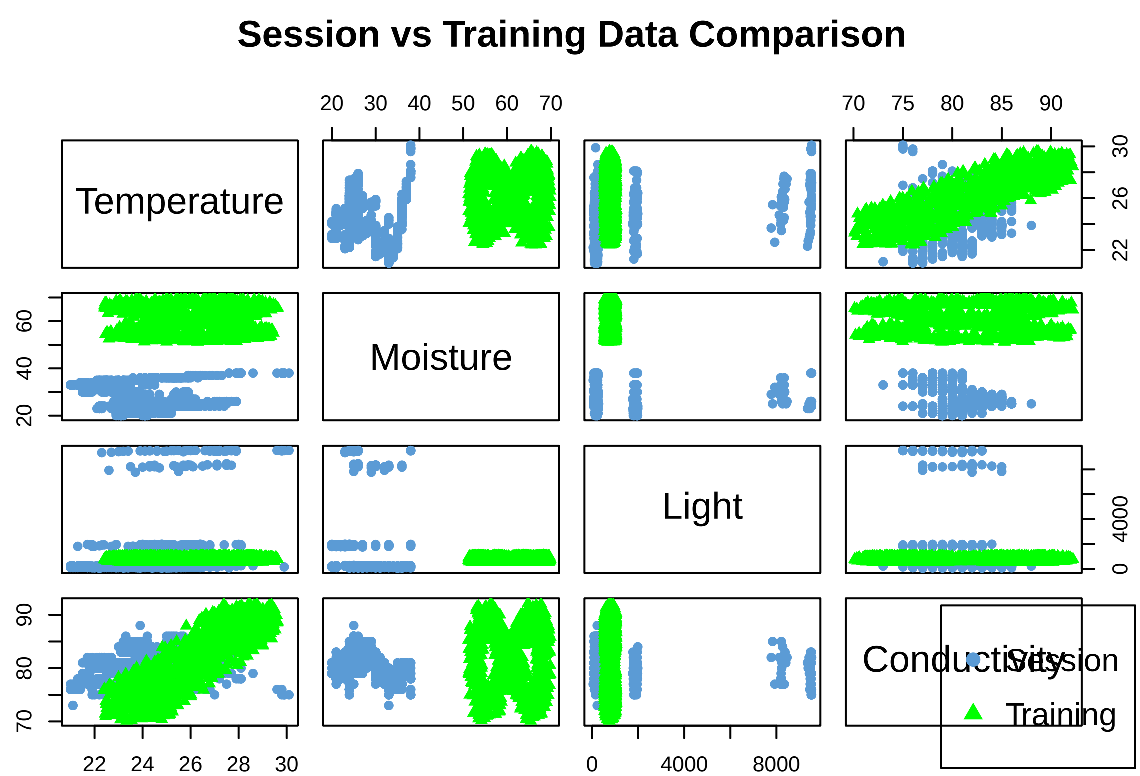


Correlation and Covariance Matrices

In this correlation matrix, score of 1.0 or -1.0 represents a perfect (positive or negative) self-correlation and values closer to 0 show less to no correlation. The matrix above reflects the same relationships as the covariance matrix.

```
## CROSS-COVARIANCE MATRIX (Session vs Training)

##                Temperature     Moisture        Light Conductivity
## Temperature      0.9615548    1.4337150     9.004801     3.377057
## Moisture         0.6396479   -3.0793977   -51.078258     1.875694
## Light         -206.5048534 2101.2970940 83063.171110  -435.946003
## Conductivity     0.8880845   -0.1147215   -59.493709     3.097508

##
## CROSS-CORRELATION MATRIX (Session vs Training)

##               Temperature     Moisture       Light Conductivity
## Temperature    0.32783027  0.158592364  0.03300803   0.35027163
## Moisture       0.07647516 -0.119450951 -0.06565774   0.06822333
## Light         -0.04712967  0.155595144  0.20381826  -0.03026833
## Conductivity   0.20344965 -0.008526914 -0.14653593   0.21587714
```

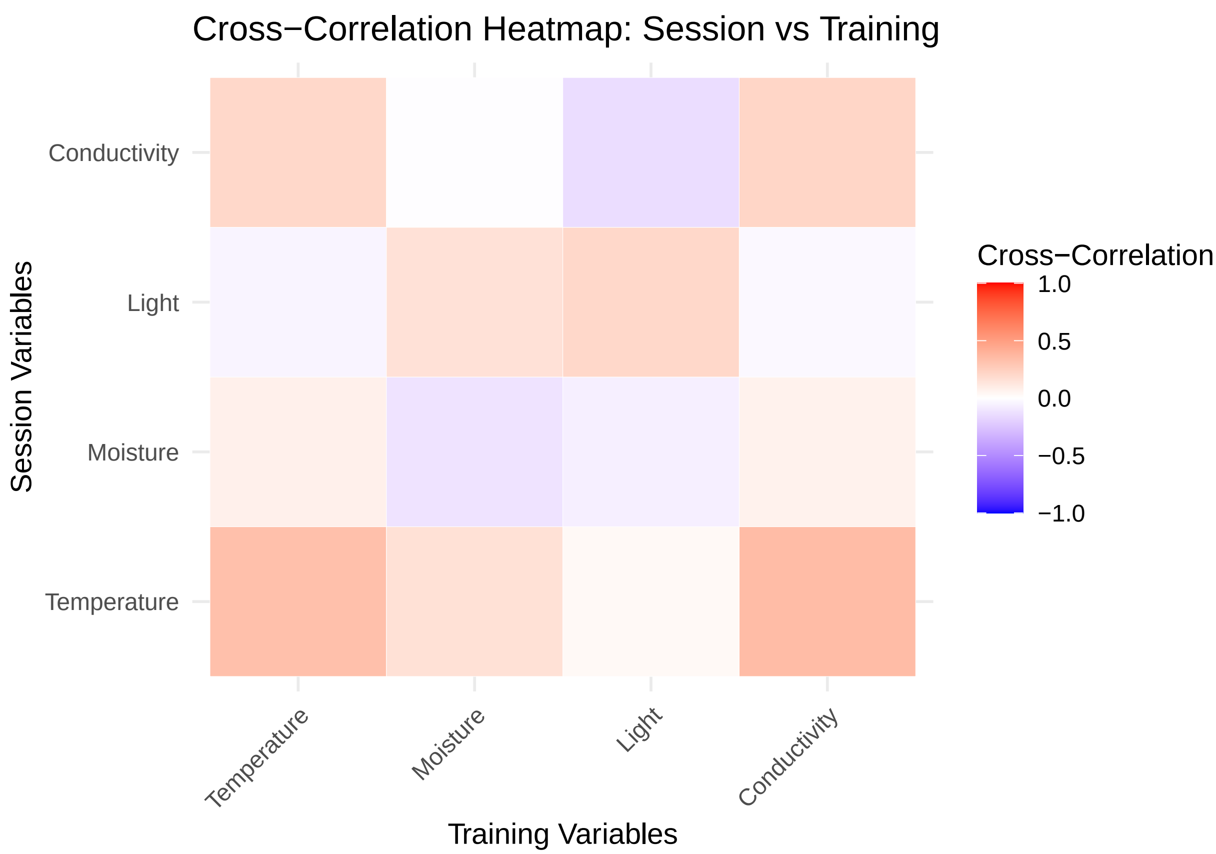
Plotting variables against each other.



Up until this point (excluding the matrices) we have been treating the session and training data as independent data sets for comparison purposes. In the next stage, instead of running similarity algorithms within each variable, both the independent (`session_data`) and the dependent (`training_data`) must be used together in order to produce valuable insights about their relationship.

Cross Correlation Heat Map

The heatmap below shows cross-correlations between session variables (y-axis) and training variables (x-axis). The color scale represents correlation strength, with red indicating positive correlation (up to +1.0), blue indicating negative correlation (down to -1.0), and white/pale colors representing weak or no correlation (around 0).



Stage II: Similarities and RSME

In this stage the goal is to measure the similarities and differences between our session and training data. From there, we'll b3 able to classify and label the features for K means clustering in Stage 3.

Preprocessing

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:reshape2':
##
##     dcast, melt

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

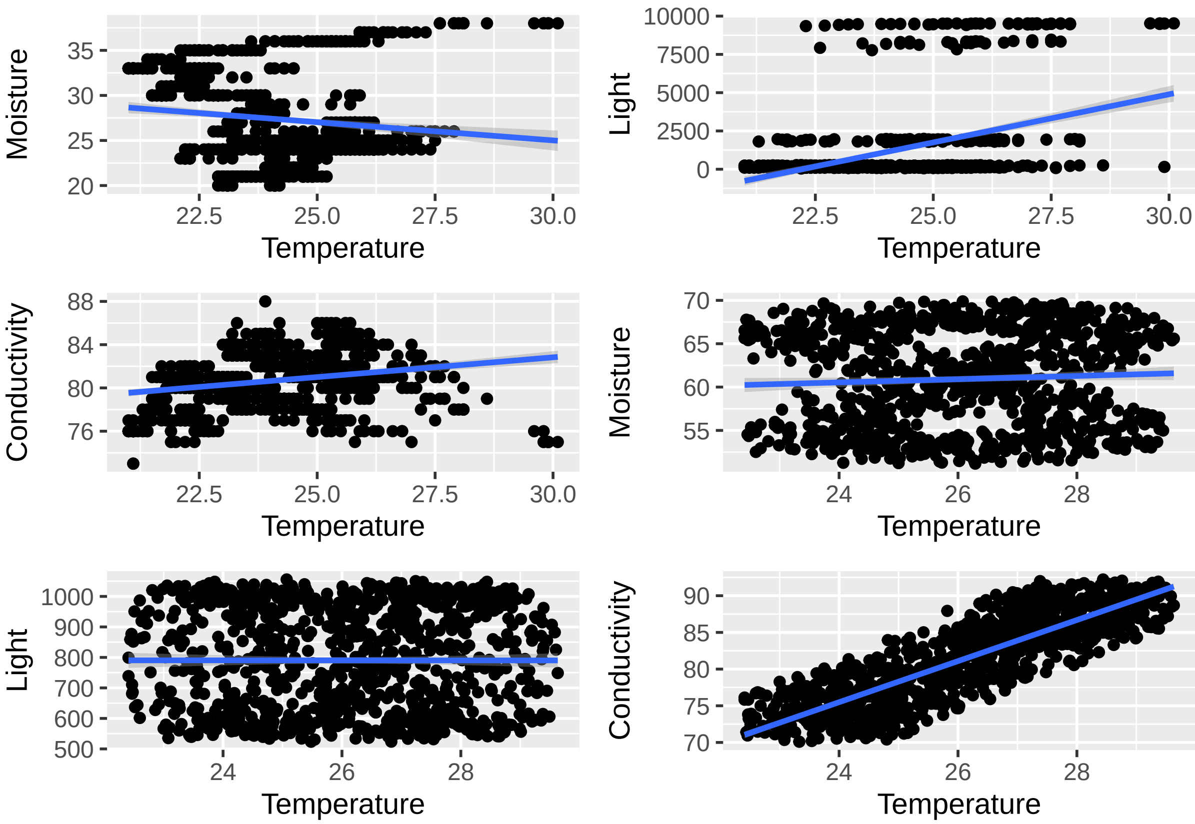
```r
library(dplyr)

training_dt <- as.data.table(training_data)
session_dt <- as.data.table(session_data)
# head()
```

Cleaning the Data

To clean up the data, we'll do some indexing by setting up keys. Then to organize it, we put the fields we want to use for our analysis into their own data frames. Converting Temperature, Moisture, Light and Conductivitry to numeric will ensure no problems when plotting.

Linear Regression Model & Errors

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```



This plot is nice, but can only tell us so much about the data. Running a `summary()` will give us more information.

Session Summaries

AGT session data

```
##
## Call:
## lm(formula = training_data$Moisture ~ session_data$Moisture)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.1044  -5.3056   0.3244   5.0218  10.4116
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           64.80503    1.06704  60.733  < 2e-16 ***
## session_data$Moisture -0.14123    0.03839  -3.679 0.000248 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.484 on 935 degrees of freedom
## Multiple R-squared:  0.01427,    Adjusted R-squared:  0.01321
## F-statistic: 13.53 on 1 and 935 DF,  p-value: 0.0002476
```

Residual Standard Error

Residuals represent the differences between the actual and predicted values of our dependent (response) variable. A high RSE indicates a weak model for this prediction.

Multiple R

Multiple R, also called the correlation coefficient, measures the strength and direction of a relationship between variables. On a scale of -1 to +1, values closer to -1 or +1 represent perfect negative or positive correlation. 0 means no correlation at all.

Multiple R2 Error

R squared tells us the proportion for variance in the dependent variable explained by the predictor variable. It can be on a scale of 0 to 1. A value closer to 1 represents greater variance, while closer to 0 represents less variance. Returning a whole 0 or 1 means none or perfect variance respectively.

Adjusted R2 Error

This error is a modified version of the above R-squared error that is able to accommodate for multiple predictors in a regression model.

Stage III: Modeling, Classification and Metrics

In this next section, we will use the KNN algorithm to find the best K for a label. As this is a placeholder for future use, a good question to use this for is:

How can we classify a subjects health by having thresholds of poor, unsatisfactory, neutral, satisfactory or excellent?

To measure this hypothetical threshold, we would use how closely or different the training data is from testing. This was explored in the previous stage, and is required to be able to have a label to predict for in KNN.

```
## Session Cluster Distribution (%):
```

```
##
##        1        2
##  8.32444 91.67556
```

```
##
## Training Cluster Distribution (%):
```

```
##
##        1        2
## 52.40128 47.59872
```

```
##
## Session Cluster Centers (scaled):
```

```
##   Temperature     Moisture      Light Conductivity
## 1    1.223590 -0.009060060  3.2021365  -0.50693761
## 2   -0.111106  0.000822683 -0.2907644   0.04603159
```

```
## 
## Training Cluster Centers (scaled):
```

```
##   Temperature   Moisture        Light Conductivity
## 1   0.7941567  0.1221828  0.009808624    0.8176824
## 2  -0.8742846 -0.1345106 -0.010798283   -0.9001840
```

```
## 
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
## 
##     combine
```

In order to begin preprocessing and find the right k, we will only keep the relevant features and label the cluster. This analysis uses the session data as testing data and the generated ideal conditions as the training data.

We will use two approaches. First with the built in K nearest neighbor functions with R and then manually calculating the accuracy of each K values.

Method 1: Built in knn() Function

```
# Select numeric features and the cluster label
features_train <- training_data[, c("Temperature", "Moisture", "Light", "Conductivity")]
labels_train <- as.factor(training_data$Cluster)

test_features <- session_data[, c("Temperature", "Moisture", "Light", "Conductivity")]
labels_test <- as.factor(session_data$Cluster)
```

Normalizing the data will help scale larger values down to a comparable size.

```
# Normalize (scale) the features
features_train_scaled <- as.data.frame(scale(features_train))
test_features_scaled <- as.data.frame(scale(test_features))
```

For our train/test split,

```
train_features <- features_train_scaled
train_labels <- labels_train
```

Now we'll proceed with modeling KNN and evaluating its performance. Before we start though, this cross correlation data needs to be fitted a bit more in order to pass through the KNN model.

```
# Make sure train and test sets have the same number of rows as their respective labels
min_train_rows <- min(nrow(train_features), length(train_labels))
min_test_rows <- min(nrow(test_features), length(labels_test))

# Trim all data to match
train_features <- train_features[1:min_train_rows, ]
train_labels <- train_labels[1:min_train_rows]
test_features <- test_features[1:min_test_rows, ]
labels_test <- labels_test[1:min_test_rows]

# Convert labels to factors after trimming
train_labels <- as.factor(train_labels)
labels_test <- as.factor(labels_test)
```

```
library(class)

# Training the KNN model
k_value <- 3  # You can experiment with different values of k
knn_model <- knn(train_features, test_features, train_labels, k = k_value)
# Evaluate the model
confusion_matrix <- table(Predicted = knn_model, Actual = labels_test)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Accuracy on Session Data (Test Data):", accuracy)
```

```
## Accuracy on Session Data (Test Data): 0.0832444
```

Method 2: Manual K Value Accuracy Computation

K Nearest Neighbor (KNN) works by computing the Euclidean distance between the test and training points. Then after selecting the proper k that is the shortest distance, it assigns those closest neighbors most common label to the test point.

Using `knn()` the `class` package automatically computes the Euclidean Distance between two points. We will adjust in the input parameters to ingest the train and test data we have already prepared.

```
# Inputs for manual model
train_data <- features_train_scaled
test_data <- test_features_scaled
train_labels <- as.numeric(labels_train)
test_labels <- as.numeric(labels_test)
```

```
euc_dis <- function(p1, p2) {
  sqrt(sum((p1 - p2)^2))
}
```

In this next section we're implementing the KNN Classifier manually to train the target data. At this stage, the classifier logic is being defined below.

Here is where we'll compute accuracy for the session data using manual KNN.

```
k_values <- seq(1, 15, 2)  # or however many k's you want

cat("Train samples:", nrow(train_data), "Label count:", length(train_labels), "\n")
```

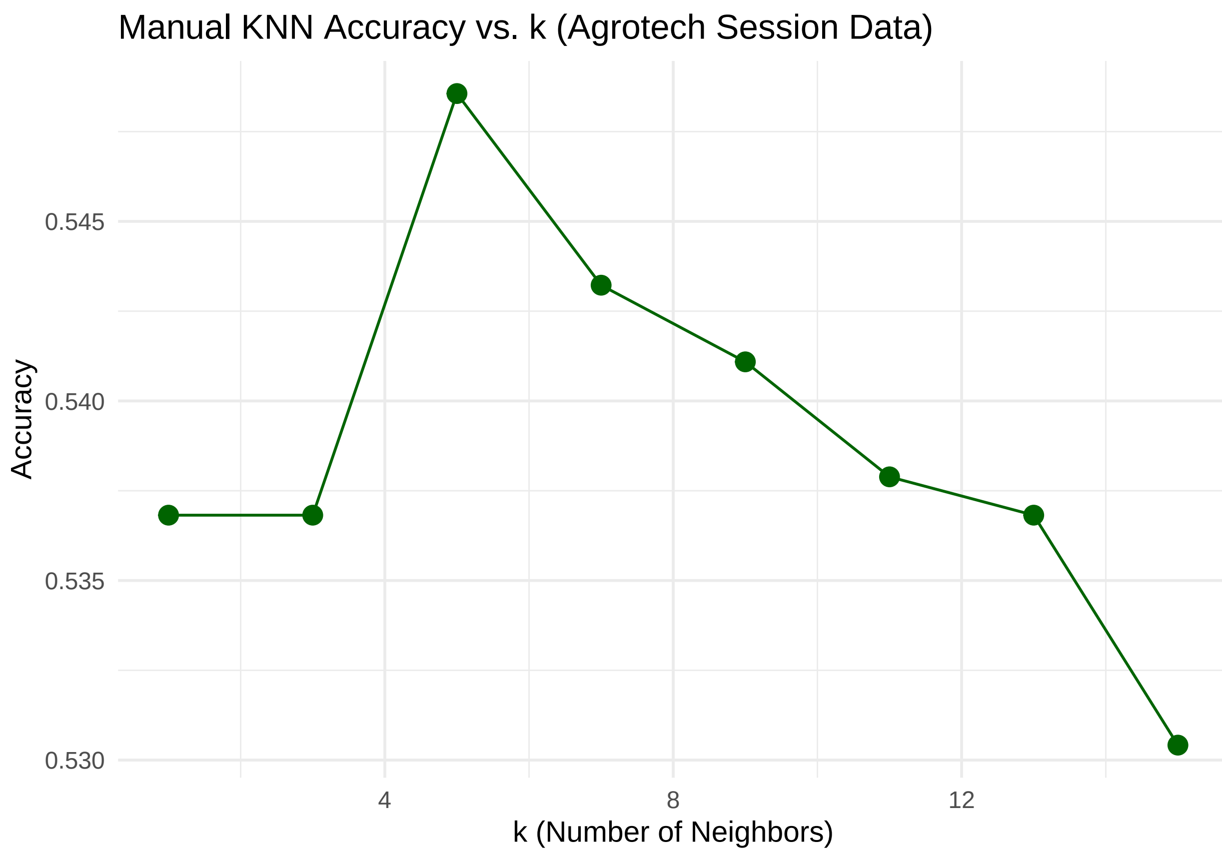
```
## Train samples: 937 Label count: 937
```

```
accuracy_results <- c()

for (k in k_values) {
  predictions <- knn(train = train_data, test = test_data, cl = train_labels, k = k)
  acc <- mean(predictions == test_labels)
  accuracy_results <- c(accuracy_results, acc)
  cat("k =", k, ", Accuracy =", round(acc * 100, 2), "%\n")
}
```

```
## k = 1 , Accuracy = 53.68 %
## k = 3 , Accuracy = 53.68 %
## k = 5 , Accuracy = 54.86 %
## k = 7 , Accuracy = 54.32 %
## k = 9 , Accuracy = 54.11 %
## k = 11 , Accuracy = 53.79 %
## k = 13 , Accuracy = 53.68 %
## k = 15 , Accuracy = 53.04 %
```

The accuracy of this model varies based in the k value used. Plotting them will provide a better idea of which to use to get the best accuracy.

```r
accuracy_data <- data.frame(
  k_values = k_values,
  accuracy = accuracy_results,
  dataset = "Session/Training"
)

ggplot(accuracy_data, aes(x = k_values, y = accuracy)) +
  geom_line(color = "darkgreen") +
  geom_point(size = 3, color = "darkgreen") +
  labs(title = "Manual KNN Accuracy vs. k (Agrotech Session Data)",
       x = "k (Number of Neighbors)", y = "Accuracy") +
  theme_minimal()
```



Stage IV: Forecasting

Time Series Analysis and Forecasting

```
## Number of unique days in dataset: 8

## Date range: 20182 to 20189

## Average observations per day: 117.12
```

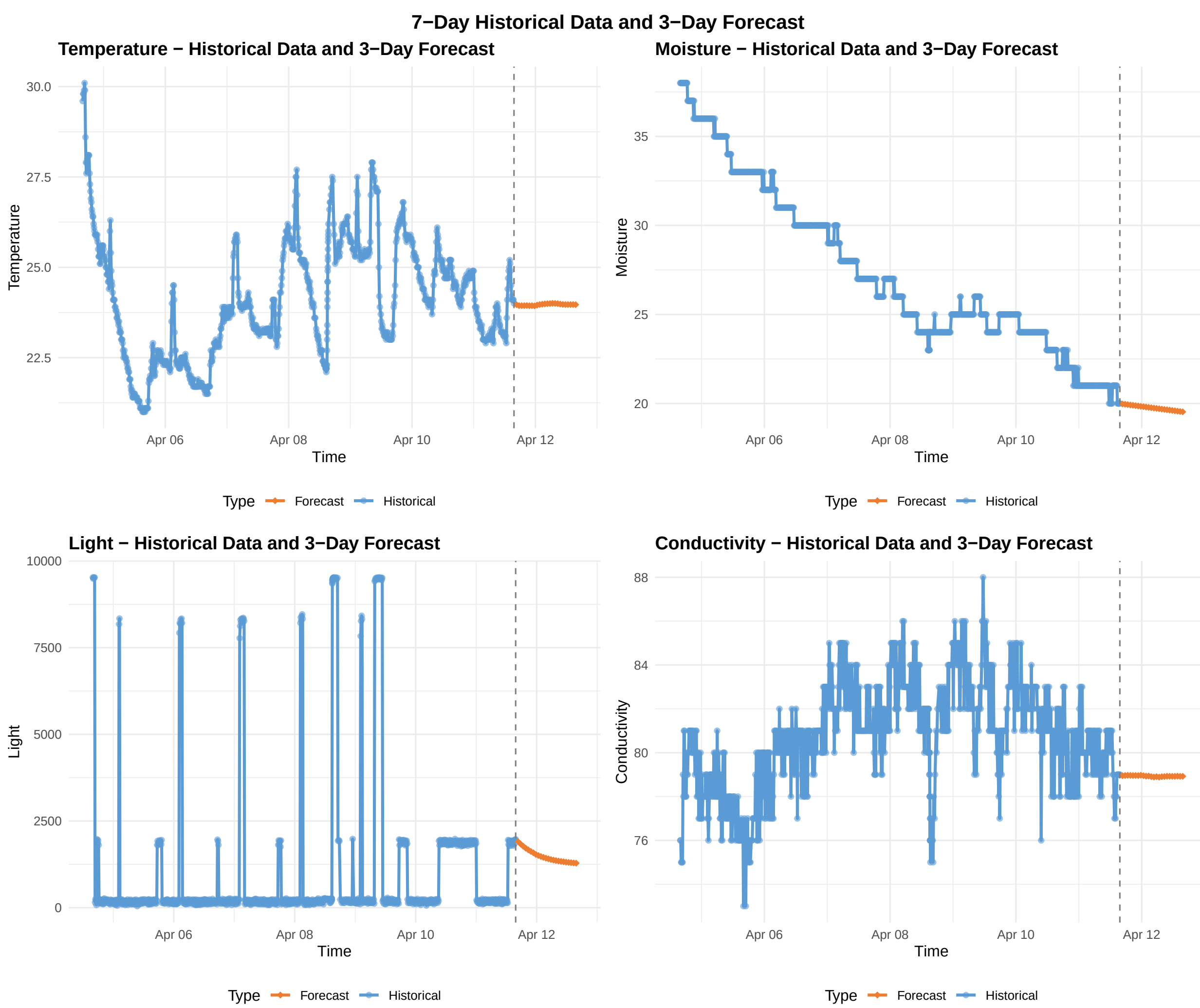
Creating Forecast Models

For each sensor variable, we'll use an appropriate time series forecasting method. We'll evaluate ARIMA, ETS (Exponential Smoothing), and Prophet models to find the best approach for our data.

Visualization of Historical Data and Forecasts

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



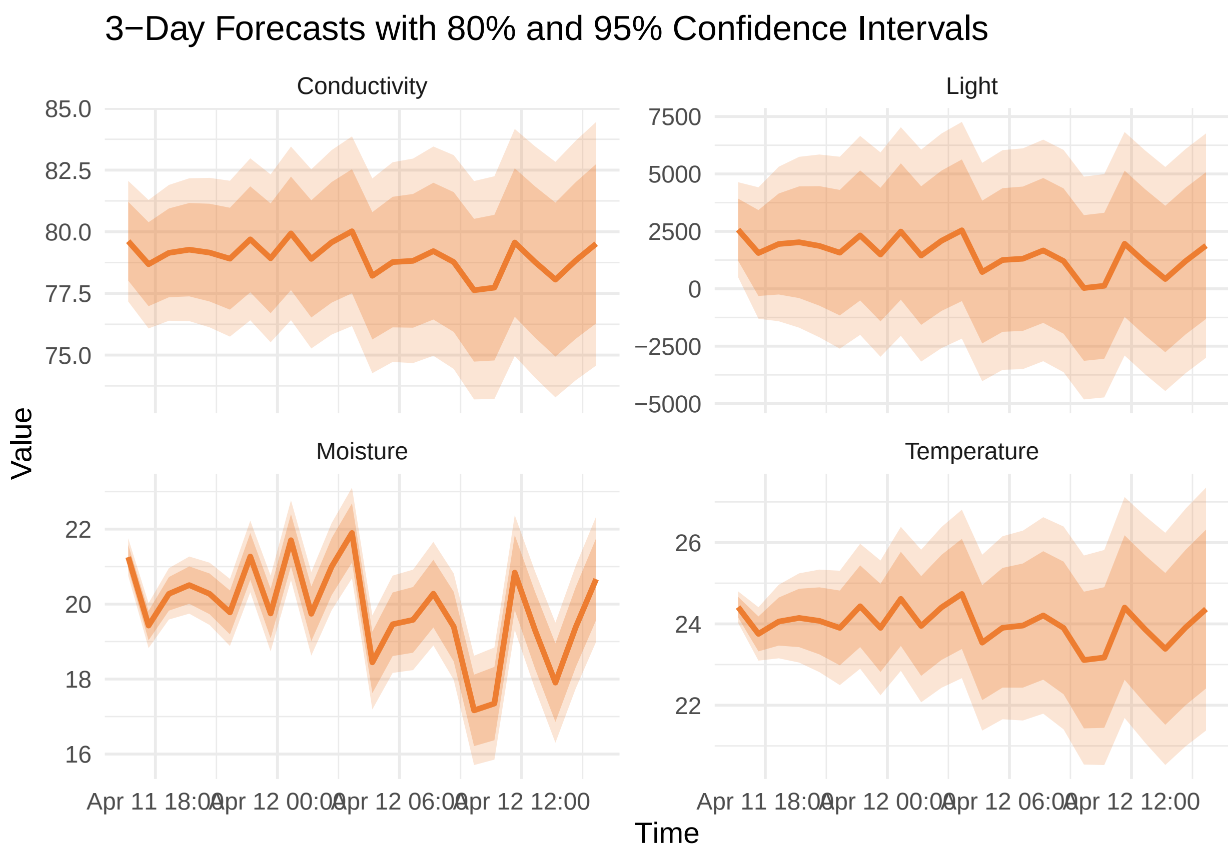
Forecast Accuracy and Confidence Intervals

```
##
##  Temperature Forecast Accuracy Metrics:
##                        ME      RMSE       MAE         MPE      MAPE      MASE
## Training set -0.003616977 0.1969759 0.1301407 -0.01469972 0.5355604 0.1164202
##                     ACF1
## Training set 0.006182006
```

```
##
##  Moisture Forecast Accuracy Metrics:
##                        ME      RMSE       MAE          MPE      MAPE      MASE
## Training set 4.556916e-05 0.2540073 0.1063922 -0.001173416 0.4101601 0.1646374
##                    ACF1
## Training set -0.0117104
```

```
##
##  Light Forecast Accuracy Metrics:
##                    ME   RMSE      MAE       MPE     MAPE     MASE         ACF1
## Training set -10.1262 1052.6 313.7139 -95.74347 106.4918 0.181823 -0.008651945
```

```
##
##  Conductivity Forecast Accuracy Metrics:
##                       ME     RMSE       MAE          MPE     MAPE      MASE
## Training set 0.008766796 1.244855 0.9799116 -0.008150268 1.215079 0.5272005
##                      ACF1
## Training set -0.006767934
```



Forecast Table Summary

Table 1: Daily Forecast Summary for the Next 3 Days

Day	Variable	Min	Mean	Max
1	Temperature	23.93	23.97	24.00
2	Temperature	NA	NA	NA
3	Temperature	NA	NA	NA
1	Moisture	19.53	19.75	19.97
2	Moisture	NA	NA	NA
3	Moisture	NA	NA	NA
1	Light	1284.85	1480.36	1902.90
2	Light	NA	NA	NA
3	Light	NA	NA	NA
1	Conductivity	78.89	78.93	78.97
2	Conductivity	NA	NA	NA
3	Conductivity	NA	NA	NA

Comparison to Training (Ideal) Data

```
##   Temperature      Moisture         Light Conductivity
##      26.13298      60.93523     790.28120      81.46064
```

```
##       Variable Ideal_Mean Ideal_Min Ideal_Max
## 1  Temperature   26.13298        NA        NA
## 2     Moisture   60.93523        NA        NA
## 3        Light  790.28120        NA        NA
## 4 Conductivity   81.46064        NA        NA
```

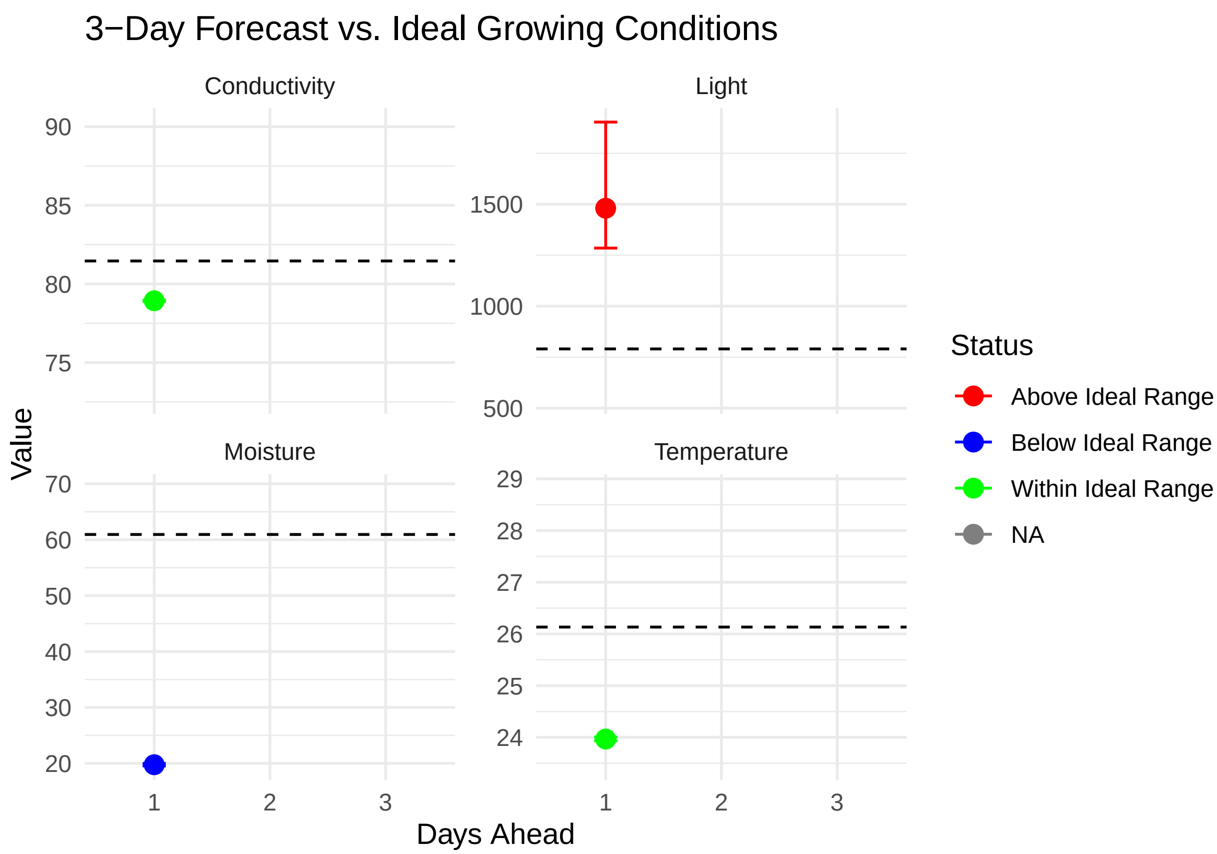
```
## Ideal Conditions (from Training Data):
```

```
##       Variable Ideal_Mean Ideal_Min  Ideal_Max
## 1  Temperature   26.13298  23.44616   28.81979
## 2     Moisture   60.93523  52.65403   69.21643
## 3        Light  790.28120 540.38063 1040.18176
## 4 Conductivity   81.46064  72.62890   90.29238
```

Table 2: Forecast Comparison to Ideal Conditions

Variable	Day	Mean	Ideal_Mean	Status
Conductivity	1	78.93	81.46	Within Ideal Range
Conductivity	2	NA	81.46	NA
Conductivity	3	NA	81.46	NA
Light	1	1480.36	790.28	Above Ideal Range
Light	2	NA	790.28	NA
Light	3	NA	790.28	NA
Moisture	1	19.75	60.94	Below Ideal Range
Moisture	2	NA	60.94	NA
Moisture	3	NA	60.94	NA
Temperature	1	23.97	26.13	Within Ideal Range
Temperature	2	NA	26.13	NA
Temperature	3	NA	26.13	NA

```
## Warning: Removed 8 rows containing missing values or values outside the scale range
## (‘geom_point()‘).
```



Stage V: Discussion