Project Walkthrough

Lucas Major, Logan Gall

2023-05-09

Introduction

This R markdown file will walk through all parts of the project. First we clean the data and create a few different files. After cleaning, we created a plot for data exploration. After, we implemented the three model types into a simulation verification program. Finally, we created the simulation program that takes user input. For more information in each process, please reference the report paper.

Libraries

We ended up using the following libraries in our project.

```
library(reticulate) ##For using R and Python code chunks in this file
library(readr) ##Read CSV's
library(dplyr)
library(lubridate)
library(ggplot2)
library(randomForest)
library(neuralnet)
library(caTools)
```

Data Cleaning

Our initial data files were “PhaseII.Data_Corrected.xlsx” for the soil moisture data set and “2022_TROE Daily ET Export.csv” as the weather output data. The soil moisture data was immediately & manually converted to CSV for easier import and analysis.

Our first data cleaning was to fix the date string mismatch between the two data sets. We actually did this in Python since it was quick to implement and conserved data types. The “Fix_ET_Date.py” file does this process. This will output a new file “ET_FIXED_DATE.csv”. The below code chunk is a copy of this file. The reticulate package prints return values to console, so I added ‘test =’ in front of the filewriter functions, if this breaks the code remove ‘test =’ and ignore printed numbers.

```
##Data Cleaning File##
##This script is used to fix a date string discrepency between the daily ET data and Phase II da
ta.
## This allows us to join the two data sets in later R analysis.

##This program works by reading in the daily ET data,
## and adjusting the date string format to match the Phase II data's format.
## ET date format: %M/%d/%yyyy ; PhaseII date format: %MM/%dd/%yy
##  ET date has no leading zeros with month and day, and 4 digit year instead of 2 digit.

import csv
newdat = []
header = []

#Read data with csv
path_to_input_file = r"2022_TROE Daily ET Export.csv"

with open(path_to_input_file, 'r') as file:
  csvreader = csv.reader(file)
  #pull header for later use. assumes header starts with "Date"
  for row in csvreader:
    if (row[0][0] == "D"):
        header.append(row)
    else: #Adjusts date based on basic string location checks
        datestring = row[0]
        if row[0][0] == '1':
            pass
        else:
            datestring = '0' + row[0]
        if datestring[4] == '/':
            datestring = datestring[0:3] + '0' + datestring[3:]

        datestring = datestring[0:6] + datestring[8:]
        row[0] = datestring
        newdat.append(row)

#Write to new file
path_to_output_file = r"ET_FIXED_DATE.csv"
with open(path_to_output_file, 'w', newline = '') as file:
    filewriter = csv.writer(file, delimiter = ',')
    test = filewriter.writerow(header[0])
    for i in range(len(newdat)):
        thisrow = newdat[i]
        test = filewriter.writerow(thisrow)
```

After the ET data is fixed, we opened the files “PhaseII.Data_Corrected_CSV.csv” and “ET_FIXED_DATE.csv”. For the first file, we removed any days with missing MeanVWC values and removed many of the rows that were not used in further analysis. We then merged the two files together with a short chunk of R code to join by the ‘date’ string. This outputs to the “PhaseII_ET.csv”.

```
dat1 = read.csv("PhaseII.Data_Corrected_CSV.csv")
##Remove missing values and unused columns
dat1 = dat1[!is.na(dat1$Mean.VWC),]
dat1 = subset(dat1, select=c(Date,Plot.No,Irrigated.After,Mean.VWC))

##Merge files
weather = read.csv("ET_FIXED_DATE.csv")
PhaseII_ET = merge(x=weather, y=dat1, by = "Date")
PhaseII_ET = subset(PhaseII_ET,select=c(Date, CumulativeRainfall,CumNet,Plot.No,Irrigated.After,
Mean.VWC))
write.csv(PhaseII_ET, "PhaseII_ET.csv", row.names=FALSE)
```

Now the data is combined. Now we need to add the next VWC observation and add a few more variables to the data set. The “Add_Next_Observation.py” AND ““Add_Next_ObservationR.R” files perform the same task of adding new data values. The R file is run in the below code chunk. This file adds the variables: ‘nextVWC_Observation’, ‘daysUntilNextObservation’, ‘CWB’ (Cumulative Water Balance), ‘NonZeroRain’ (True/False if Rain > 0), ‘CRain’ (Cumulative rain between observations), ‘CET’ (Cumulative ET between observations), and ‘CIrr’ (Cumulative Irrigation between observations). This outputs to the file “PhaseII_ET_WithNExtWater_temp.csv”.

```
#library(readr) ##Read CSV's
#library(dplyr)
#library(lubridate)

#Read csv file.
df <- read_csv("PhaseII_ET.csv")
```

```
## Rows: 1437 Columns: 6
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): Date, Irrigated.After
## dbl (4): CumulativeRainfall, CumNet, Plot.No, Mean.VWC
##
## 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.
```

```
#Convert date to datetime objects, add two new columns
df$Date <- mdy(df$Date)
df$nextVWC_Observation <- 0.0
df$daysUntilNextObservation <- 0
df$CWB <- 0.0
df$NonZeroRain <- FALSE
df$CRain <- 0.0
df$CET <- 0.0
df$CIrr <- 0.0

#Iterate through all unique plots
for (i in unique(df$`Plot.No`)){
  sample <- df[df$`Plot.No` == i,]
  dates <- unique(sample$Date) #Get all unique dates for a given plot

  #Iterate through all unique dates minus 1
  for (j in 1:(length(unique(sample$Date))-1)){
    #Find next observation, find number of days, update observation columns.
    datediff <- as.integer(abs(difftime(dates[j], dates[j+1], units = "days")))
    thisIndex <- which(df$`Plot.No` == i & df$Date == dates[j])
    nextIndex <- which(df$`Plot.No` == i & df$Date == dates[j+1])
    df$nextVWC_Observation[thisIndex] <- df$`Mean.VWC`[nextIndex]
    df$daysUntilNextObservation[thisIndex] <- datediff

    thisWater <- df$CumNet[thisIndex]
    nextWater <- df$CumNet[nextIndex]
    diff <- nextWater - thisWater
    diffCopy <- nextWater - thisWater
    if(df$Irrigated.After[thisIndex] == "Yes"){
      diff <- diff + 0.25
      df$CIrr[thisIndex] <- 0.25
    }

    thisRain <- df$CumulativeRainfall[thisIndex]
    nextRain <- df$CumulativeRainfall[nextIndex]
    if (nextRain - thisRain > 0){
      df$NonZeroRain[thisIndex] <- TRUE
    }

    df$CRain[thisIndex] <- (nextRain - thisRain)
    df$CWB[thisIndex] <- diff
    df$CET[thisIndex] <- diffCopy - (nextRain - thisRain)
  }
}

write_csv(df, "PhaseII_ET_WithNextWater_temp.csv") #save
```

Since we aren't perfect coders, and parts of the data cleaning process were created at different times, we have to fix the date string again after reading the data in R. The "Fix_ET_Date_Verification.py" file performs this quick fix, rewriting the "PhaseII_ET_WithNextWater.csv". The below code chunk performs this operation.

This outputs to "PhaseII_ET_WithNextWater.csv" when the script is done.

```
import csv
from datetime import datetime

# Open the CSV file for reading
with open(r"PhaseII_ET_WithNextWater_temp.csv", 'r') as csvfile:
    reader = csv.reader(csvfile)

    # Read the header row
    header = next(reader)

    # Change the first column label to 'Date'
    header[0] = 'Date'

    # Create a new CSV file for writing
    with open("PhaseII_ET_WithNextWater.csv", 'w', newline='') as outfile:
        writer = csv.writer(outfile)

        # Write the modified header row to the new file
        test = writer.writerow(header)

        # Process the remaining rows
        for row in reader:
            # Convert the first column to datetime object
            date_str = row[0]
            date_obj = datetime.strptime(date_str, '%Y-%m-%d')

            # Format the first column as '%m/%d/%y' with quotation marks
            row[0] = str(date_obj.strftime('%m/%d/%y'))

            # Write the modified row to the new file
            test = writer.writerow(row)
```

Now the final set of data cleaning. In the later parts of the project, we decided to include the next few days of data observations. The “Create_Simulation_Verification.py” AND “Create_Simulation_VerificationR.R” perform this task the same. These read in the “PhaseII_ET_WithNextWater.csv” and “ET_FIXED_DATE.csv” and add in new variables ‘nextET1’, ‘nextRain1’ … ‘nextET5’, and ‘nextRain5’ which just carry the ET and Rainfall values for the days following an observation up until the next observation.

This overwrites the “PhaseII_ET_WithNextWater.csv” file.

```
#library(readr)
#library(dplyr)
#library(lubridate)

#Read csv file.
df <- read_csv("PhaseII_ET_WithNextWater.csv")
```

```
## Rows: 1437 Columns: 13
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (2): Date, Irrigated.After
## dbl (10): CumulativeRainfall, CumNet, Plot.No, Mean.VWC, nextVWC_Observation...
## lgl  (1): NonZeroRain
##
## 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.
```

```
df2 <- read_csv("ET_FIXED_DATE.csv")
```

```
## Rows: 184 Columns: 15
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (1): Date
## dbl (14): TempLow, TempHigh, TempMean, RHLow, RHHigh, SolarRadiation, WindSp...
##
## 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.
```

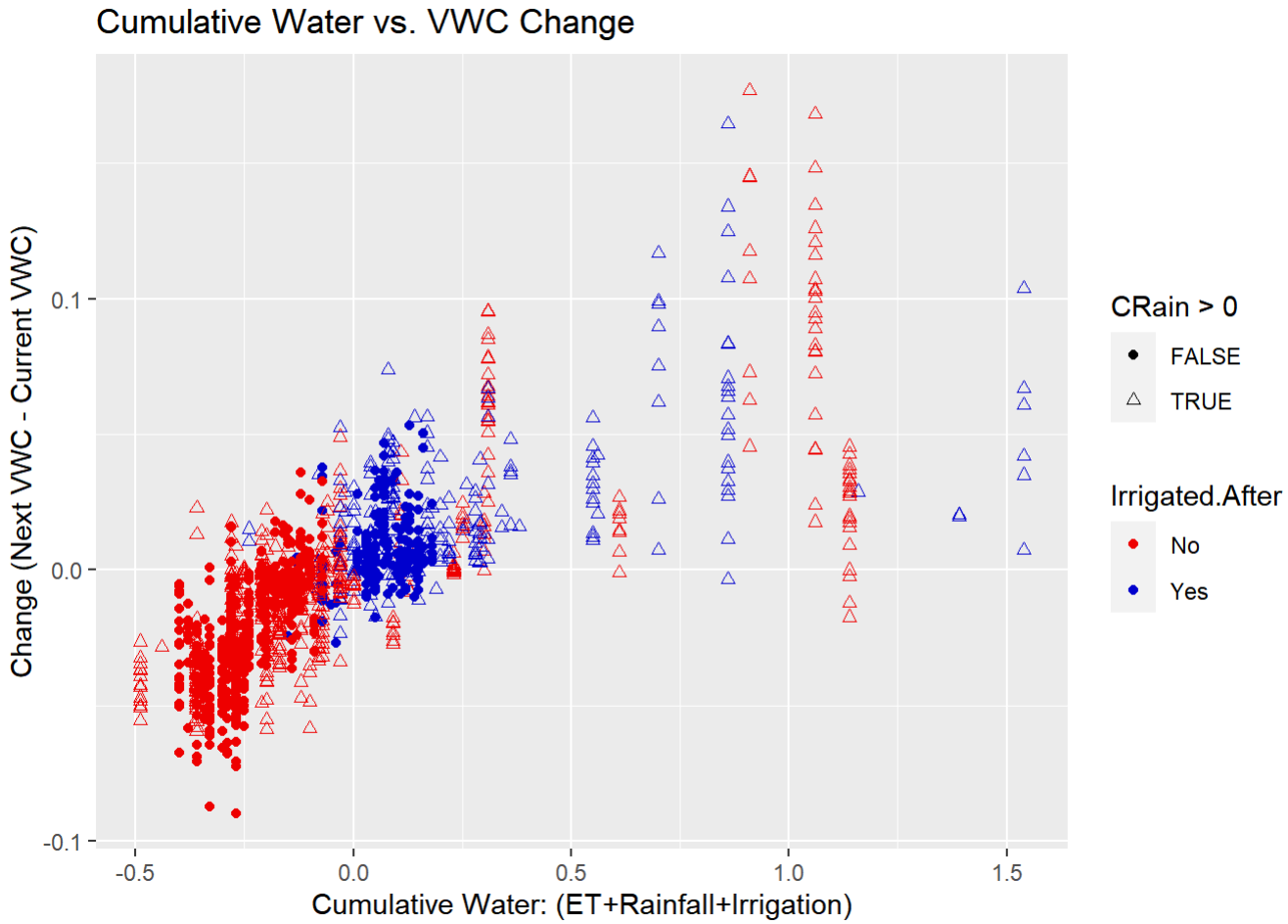
```
#Convert date to datetime objects, add new columns
df$Date <- mdy(df$Date)
df2$Date <- mdy(df2$Date)

df$nextET1 <- 0.0
df$nextRain1 <- 0.0
df$nextET2 <- 0.0
df$nextRain2 <- 0.0
df$nextET3 <- 0.0
df$nextRain3 <- 0.0
df$nextET4 <- 0.0
df$nextRain4 <- 0.0
df$nextET5 <- 0.0
df$nextRain5 <- 0.0

#Iterate through all unique plots
for (i in unique(df$`Plot.No`)){
  sample <- df[df$`Plot.No` == i,]
  dates <- unique(sample$Date) #Get all unique dates for a given plot

  #Iterate through all unique dates minus 1
  for (j in 1:(length(unique(sample$Date))-1)){
    #Find next observation, find number of days, update observation columns.
    datediff <- as.integer(abs(difftime(dates[j], dates[j+1], units = "days")))

    tempDay <- dates[j]
    rainVals <- c(0.0, 0.0, 0.0, 0.0, 0.0)
    eVals <- c(0.0, 0.0, 0.0, 0.0, 0.0)

    for (k in 1:datediff){
      tempDay <- tempDay + days(1)
      rainVal <- df2$Rainfall[df2$Date == tempDay]
      eVal <- -df2$ETRef[df2$Date == tempDay]

      rainVals[k] <- rainVal
      eVals[k] <- eVal
    }

    # Assign the new list to the DataFrame column
    thisIndex <- which(df$`Plot.No` == i & df$Date == dates[j])
    df$nextRain1[thisIndex] <- rainVals[1]
    df$nextET1[thisIndex] <- eVals[1]
    df$nextRain2[thisIndex] <- rainVals[2]
    df$nextET2[thisIndex] <- eVals[2]
    df$nextRain3[thisIndex] <- rainVals[3]
    df$nextET3[thisIndex] <- eVals[3]
    df$nextRain4[thisIndex] <- rainVals[4]
    df$nextET4[thisIndex] <- eVals[4]
    df$nextRain5[thisIndex] <- rainVals[5]
    df$nextET5[thisIndex] <- eVals[5]
  }
}
```

```
write_csv(df, "PhaseII_ET_WithNextWater.csv") #save
```

Data Exploration

Through our data exploration, we tested a few different plots. Overall, we condensed the plots down to just a single visualization between Cumulative Water Balance and the Difference between the current and next observations. This condenses all the water balance features and simplifies the relation between the current and next observations. The below chunk of code creates this plot.

```
data = read.csv("PhaseII_ET_WithNextWater.csv")
#Remove points with no next observation
data = data[!(data$nextVWC_Observation==0.0),]

p = ggplot(data,aes(x=CWB, y= nextVWC_Observation-Mean.VWC,col=Irrigated.After,shape=CRain>0.0))
+geom_point()
p + labs(x="Cumulative Water: (ET+Rainfall+Irrigation)",y="Change (Next VWC - Current VWC)",titl
e="Cumulative Water vs. VWC Change")+scale_color_manual(values=c("red2","mediumblue"))+scale_sha
pe_manual(values=c(16,2))
```



Model Selection

Initially, we fit our data models and did direct summaries and predictions to view what variables had the greatest effect on model accuracy.

As we focused on applying single day data predictions in our simulation program, we converted our model fitting process to test based on these single day steps. This provided better estimate of the true accuracy of our model and program. The below code, also found in “Simulation_Verification.Rmd” runs through the model fitting process and model comparisons.

Import Data and extra cleaning:

```
data = read.csv("PhaseII_ET_WithNextWater.csv")
data = data[!(data$nextVWC_Observation==0.0),]
data = na.omit(data)

##Analysis dataframe to not mess with original data. This is a residual data frame from an early analysis that isn't as relevent any more.
analysis = data.frame(data$Irrigated.After, data$Mean.VWC, data$nextVWC_Observation, data$daysUntilNextObservation, data$CWB, data$CRain, data$CET, data$CIrr, data$nextET1, data$nextET2, data$nextET3, data$nextET4, data$nextET5, data$nextRain1, data$nextRain2, data$nextRain3, data$nextRain4, data$nextRain5)
```

A few models and summaries for overview.

```
model = lm(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = analysis)
#model = lm(data.nextVWC_Observation ~ data.Mean.VWC * data.CRain * data.CET * data.CIrr, data = analysis)
#model = lm(data.nextVWC_Observation ~ data.Mean.VWC + data.CRain + data.CET + data.CIrr + data.Mean.VWC:data.CRain + data.Mean.VWC:data.CET, data = analysis)
#model = randomForest(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = analysis, ntree = 200, nodesize = 10)
#model = neuralnet(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data=analysis, hidden=c(9,10),linear.output=FALSE)

summary(model)
```

```
##
## Call:
## lm(formula = data.nextVWC_Observation ~ data.Mean.VWC + data.CWB,
##     data = analysis)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.09911 -0.01109 -0.00015  0.01095  0.10404
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.022262   0.002178   10.22   <2e-16 ***
## data.Mean.VWC 0.914325   0.007665  119.29   <2e-16 ***
## data.CWB      0.068516   0.001619   42.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02095 on 1404 degrees of freedom
## Multiple R-squared:  0.9167, Adjusted R-squared:  0.9166
## F-statistic:  7730 on 2 and 1404 DF,  p-value: < 2.2e-16
```

Split, train, & test model across X cross validation steps. This is not the most efficient script, but it consistently works given different models and worked for our interpretations. This first iteration has lower R2 than typical.

```
#store cross validation results
R2_fil = c()
MSE_fil = c()

numIterations = 1

#cross validation iterations
for (n in 1:numIterations) {

#Set a constant seed so each model is compared equally
set.seed(n)
split = sample.split(analysis$data.nextVWC_Observation, SplitRatio = 0.60)
train = subset(analysis, split == TRUE)
test = subset(analysis, split == FALSE)

#Fit a model
#model = lm(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = train)
model = lm(data.nextVWC_Observation ~ data.Mean.VWC * data.CRain * data.CET * data.CIrr, data =
train)
#model = lm(data.nextVWC_Observation ~ data.Mean.VWC + data.CRain + data.CET + data.CIrr + data.
Mean.VWC:data.CRain + data.Mean.VWC:data.CET, data = train)
#model = randomForest(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = train, ntree =
200, nodesize = 10)
#model = neuralnet(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data=train, hidden=c(9,1
0),linear.output=FALSE)

#summary(model)

#Actual and prediction values
actual = c()
pred = c()

#Iterate through testing data set, prepping data for a single day step and predicting using one
day step.
for (i in 1:nrow(test)) {
  row = test[i, ]

  #Future ET values
  ET_vals = c(row$data.nextET1, row$data.nextET2, row$data.nextET3, row$data.nextET4, row$data.n
extET5)
  Rain_vals = c(row$data.nextRain1, row$data.nextRain2, row$data.nextRain3, row$data.nextRain4,
row$data.nextRain5)
  numDays = row$data.daysUntilNextObservation
  startVWC = row$data.Mean.VWC
  tempVWC = startVWC
  for (j in 1:numDays) {
    tempWB = ET_vals[j] + Rain_vals[j]
    irrigate = 0.0

    #check if irrigation has occured
    if (j == 1){
      tempWB = tempWB = row$data.CIrr
```

```
      irrigate = row$data.CIrr
    }

    #Data frame just for prediction input data
    tempDF = data.frame(data.Mean.VWC = tempVWC, data.CWB = tempWB, data.daysUntilNextObservatio
n = 1, data.CIrr = irrigate, data.CRain = Rain_vals[j], data.CET= ET_vals[j])
    tempResp = predict(model, newdata = tempDF, type = 'response')
    tempVWC = tempResp
  }
  #After prediction is made, add to lists.
  actual = c(actual, row$data.nextVWC_Observation) #this was startVWC at an earlier iteration, t
hat was probably mistake.
  pred = c(pred, tempVWC)
}

R2 = 1-sum((actual-pred)^2)/sum((actual-mean(actual))^2)
R2_fil = c(R2_fil, R2)

err = actual - pred
MSE = mean((err^2))
MSE_fil = c(MSE_fil, MSE)
}

mean(MSE_fil)
```

```
## [1] 0.0005049362
```

```
mean(R2_fil)
```

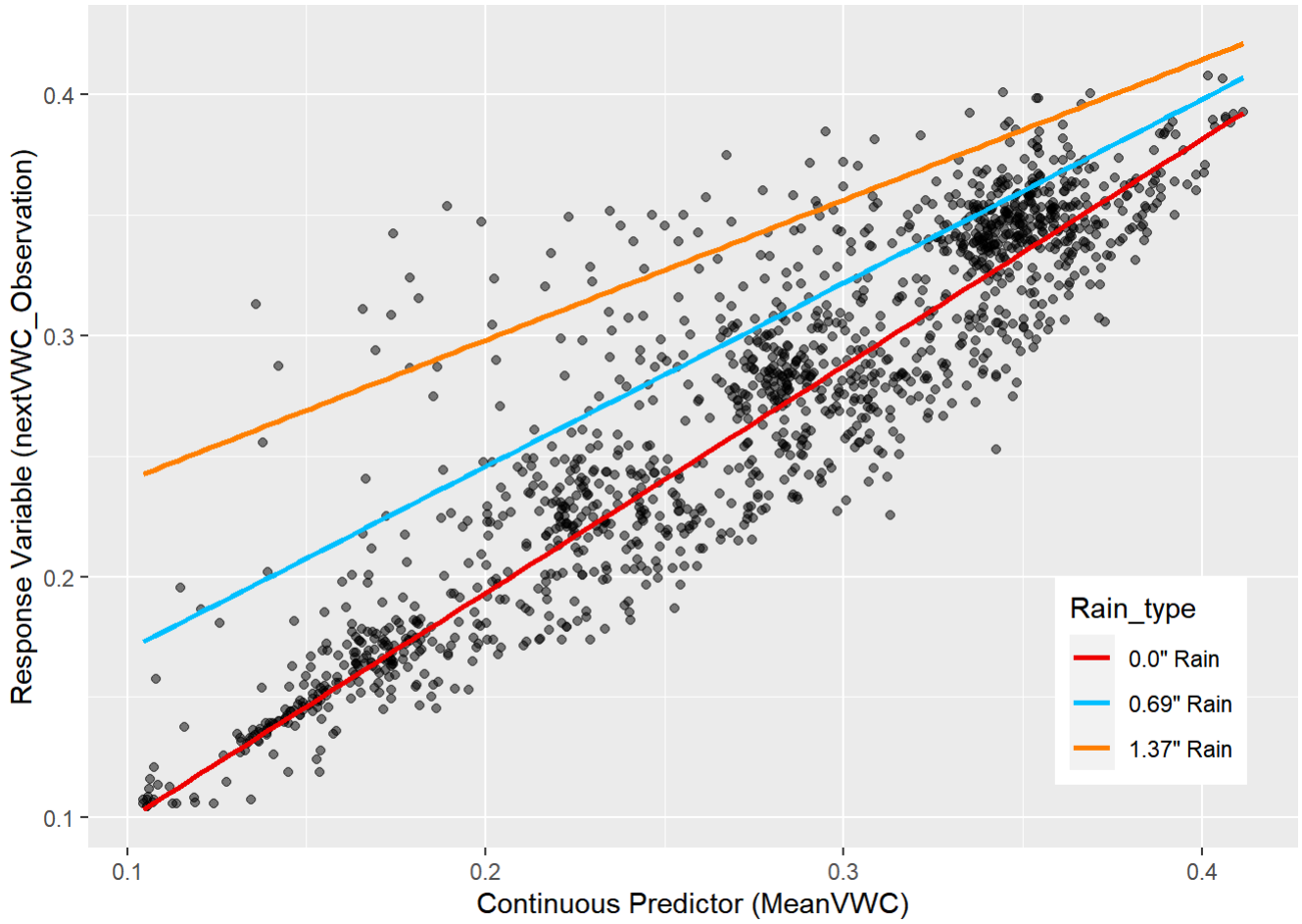
```
## [1] 0.9034373
```

Model Comparisons

Linear model

Looking at linear model results, we ended up plotting the interaction effects of Mean.VWC and Rainfall against the Next VWC Observation. The below code creates that plot. Again, the code for all these are found in “Simulation_Verification.Rmd”

```
# Load required libraries
library(ggplot2)

# Generate example data
currVWC <- analysis$data.Mean.VWC
Rain <- analysis$data.CRain
NextObs <- analysis$data.nextVWC_Observation
data <- data.frame(currVWC, Rain, NextObs)

# Fit a linear model
model <- lm(NextObs ~ currVWC * Rain, data = data)

# Choose representative values for Rain (e.g., min, mean, and max)
Rain_representative <- c(min(data$Rain), (min(data$Rain) + max(data$Rain)) / 2, max(data$Rain))

# Create a new data frame for predictions
data_pred <- expand.grid(currVWC = seq(min(data$currVWC), max(data$currVWC), length.out = 100),
                         Rain = Rain_representative)
data_pred$Y_pred <- predict(model, newdata = data_pred)

# Convert the representative Rain values to a factor for plotting
data_pred$Rain_type <- factor(data_pred$Rain,
                              labels = c("0.0\" Rain", "0.69\" Rain", "1.37\" Rain"))

# Create the line plot
ggplot() +
  geom_point(data = data, aes(x = currVWC, y = NextObs), alpha = 0.5) +
  geom_line(data = data_pred, aes(x = currVWC, y = Y_pred, color = Rain_type), size = 1) +
  labs(x = "Continuous Predictor (MeanVWC)",
       y = "Response Variable (nextVWC_Observation)") + scale_color_manual(values=c("red2","deep
skyblue","darkorange1")) +
   theme(legend.position = c(0.95, 0.1), # Position: bottom right corner
         legend.justification = c(0.95, 0.1), # Justification: bottom right corner
         legend.background = element_blank(), # Remove legend background
         legend.box.background = element_rect(color = "transparent")) # Transparent box
```



Random Forest

To tune the random forest, we tested the random forest across a variety of node and tree sizes. Due to extreme runtime of this program, the code will be left out from this file. Refer to “Simulation_Verification.Rmd”

Neural Network

To tune the neural network, we tested the neural net across a variety of neuron and hidden layer sizes. Due to extreme runtime of this program, the code will be left out from this file. Refer to “Simulation_Verification.Rmd”

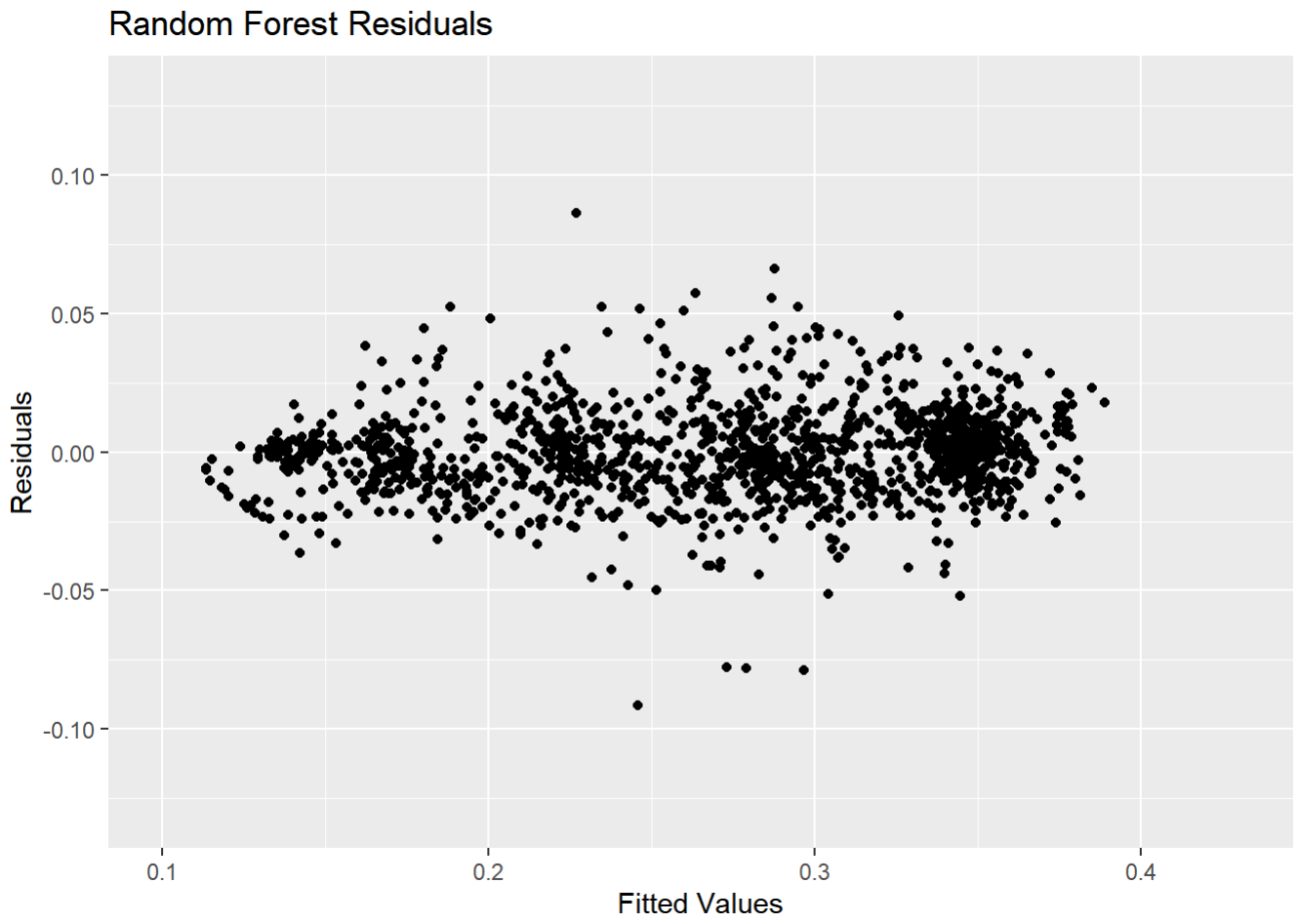
Residual plots

At the end of “Simulation_Verification.Rmd”, we created a few residual plots of the data. These can be viewed with the below chunk of code.

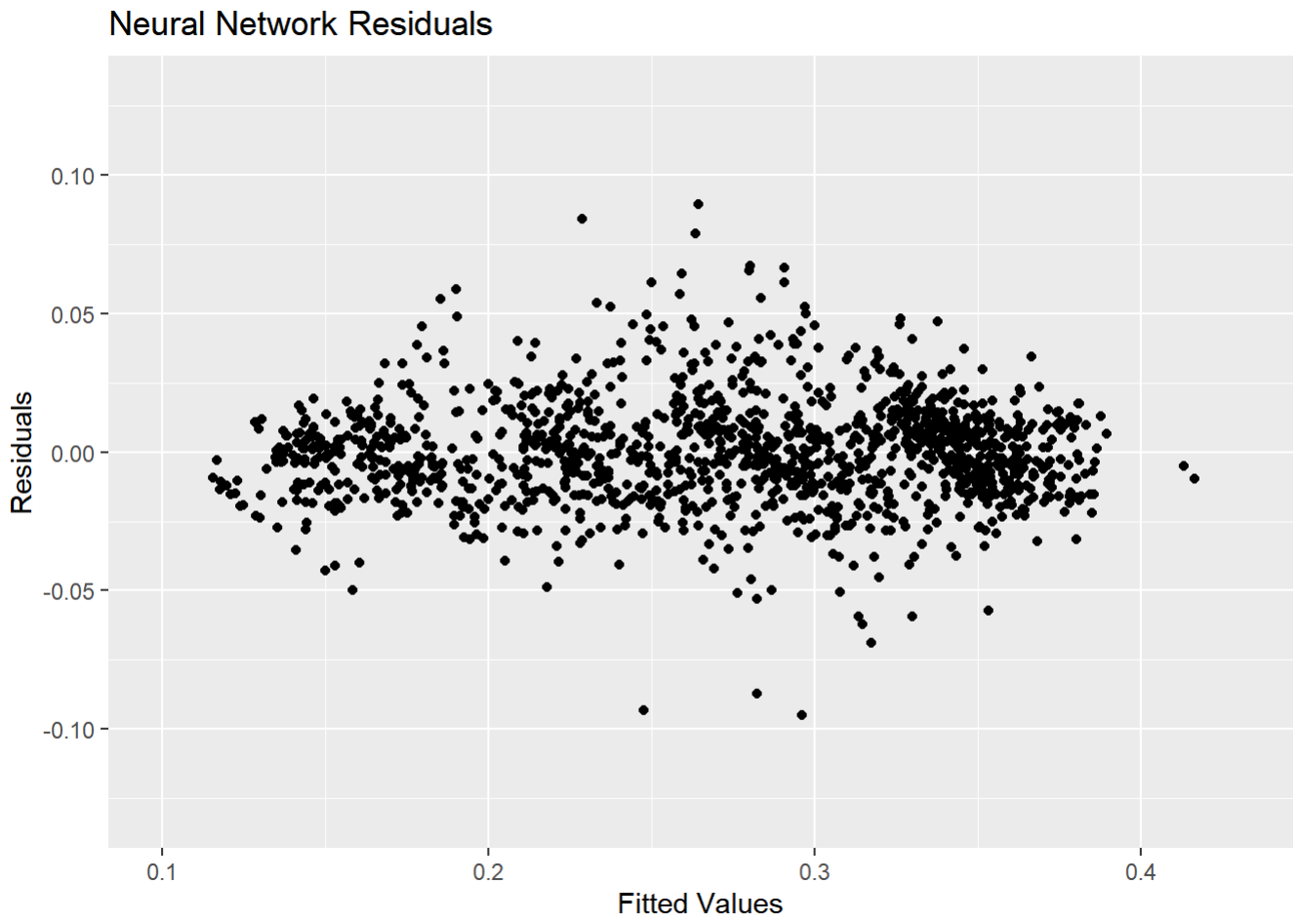
```
model = lm(data.nextVWC_Observation ~ data.Mean.VWC * data.CRain * data.CET * data.CIrr, data =
analysis)
ggplot(analysis,aes(x=predict(model),y=data.nextVWC_Observation-predict(model)))+geom_point() +l
abs(x="Fitted Values", y="Residuals",title="Linear Model Residuals")+xlim(0.1,0.43)+ylim(-0.13,
0.13)
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```


```
model = randomForest(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = analysis, ntree
= 200, nodesize = 10)
ggplot(analysis,aes(x=predict(model),y=data.nextVWC_Observation-predict(model)))+geom_point() +l
abs(x="Fitted Values", y="Residuals",title="Random Forest Residuals")+xlim(0.1,0.43)+ylim(-0.13,
0.13)
```




```
model = neuralnet(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data=analysis, hidden=c
(9,10),linear.output=FALSE)
ggplot(analysis,aes(x=predict(model,type="response",newdata=analysis),y=data.nextVWC_Observation
-predict(model,type="response",newdata=analysis)))+geom_point() +labs(x="Fitted Values", y="Resi
duals",title="Neural Network Residuals")+xlim(0.1,0.43)+ylim(-0.13,0.13)
```



Simulation Implementaion

Now to the final parts of our analysis. The simulation program can be found in “Sim_Program.Rmd”. This first trains a model of the user’s choosing. Afterwards, a user can enter data to test the simulation program. This file will pull from the example used in the report paper. This Markdown file does not support user input, so data values are hard coded for just this example.

Training the model:

```
set.seed(10)
split = sample.split(analysis$data.nextVWC_Observation, SplitRatio = 0.99)
train = subset(analysis, split == TRUE)
test = subset(analysis, split == FALSE)

#model = lm(data.nextVWC_Observation ~ data.Mean.VWC + data.CWB, data = train)
model = lm(data.nextVWC_Observation ~ data.Mean.VWC * data.CRain * data.CET * data.CIrr, data =
train)
summary(model)
```

```
##
## Call:
## lm(formula = data.nextVWC_Observation ~ data.Mean.VWC * data.CRain *
##     data.CET * data.CIrr, data = train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.093709 -0.010662 -0.000461  0.010179  0.083759
##
## Coefficients:
##                                             Estimate Std. Error t value
## (Intercept)                                 0.011563   0.007819   1.479
## data.Mean.VWC                               1.006839   0.027765  36.262
## data.CRain                                 -0.015121   0.038742  -0.390
## data.CET                                    0.073157   0.029207   2.505
## data.CIrr                                   0.004768   0.052940   0.090
## data.Mean.VWC:data.CRain                    0.017859   0.139657   0.128
## data.Mean.VWC:data.CET                      0.216102   0.102222   2.114
## data.CRain:data.CET                        -0.536917   0.142286  -3.774
## data.Mean.VWC:data.CIrr                     0.070545   0.184510   0.382
## data.CRain:data.CIrr                        0.336790   0.227401   1.481
## data.CET:data.CIrr                         -0.546420   0.238931  -2.287
## data.Mean.VWC:data.CRain:data.CET           0.914529   0.508729   1.798
## data.Mean.VWC:data.CRain:data.CIrr         -0.811340   0.797913  -1.017
## data.Mean.VWC:data.CET:data.CIrr            0.730931   0.833525   0.877
## data.CRain:data.CET:data.CIrr              -0.068182   1.033419  -0.066
## data.Mean.VWC:data.CRain:data.CET:data.CIrr  1.463939   3.575980   0.409
##                                            Pr(>|t|)
## (Intercept)                                0.139399
## data.Mean.VWC                               < 2e-16 ***
## data.CRain                                 0.696379
## data.CET                                   0.012366 *
## data.CIrr                                  0.928243
## data.Mean.VWC:data.CRain                   0.898266
## data.Mean.VWC:data.CET                     0.034691 *
## data.CRain:data.CET                        0.000168 ***
## data.Mean.VWC:data.CIrr                    0.702270
## data.CRain:data.CIrr                       0.138825
## data.CET:data.CIrr                         0.022351 *
## data.Mean.VWC:data.CRain:data.CET          0.072448 .
## data.Mean.VWC:data.CRain:data.CIrr         0.309415
## data.Mean.VWC:data.CET:data.CIrr           0.380686
## data.CRain:data.CET:data.CIrr              0.947405
## data.Mean.VWC:data.CRain:data.CET:data.CIrr 0.682324
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01837 on 1376 degrees of freedom
## Multiple R-squared:  0.9368, Adjusted R-squared:  0.9361
## F-statistic:  1360 on 15 and 1376 DF,  p-value: < 2.2e-16
```

Running the Simulation. The user can change the amount of water per irrigation manually by updating ‘irrigation_val’.

```
##User input sections
#startVWC = as.numeric(readline("Enter the current (day 0) VWC value of your plot [generally 0.1
-0.4]: "))
#numDays = as.numeric(readline("Enter how many days out to predict [generally 1-5]: "))

##Fix for not allowing user input
startVWC = 0.2272083
numDays = 3

ET_vals = c()
Rain_vals = c()
irrigation_val = 0.25

##User input for forecast
#for (i in 1:numDays) {
#  tempET = as.numeric(readline(paste("Enter the forecasted ET value for day", i-1, " [inches of
water]: ")))
#  tempRain = as.numeric(readline(paste("Enter the forecasted Precipitation value for day", i-1,
" [inches of water]: ")))
#  ET_vals = c(ET_vals, tempET)
#  Rain_vals = c(Rain_vals, tempRain)
#}

##Fix for not allowing user input
ET_vals = c(-0.19, -0.15, -0.18)
Rain_vals = c(0.0, 0.23, 0.01)

#Threshold VWC values that correspond to 0%, 15%, 20%, 25%, ... 75% PAW
VWC_thresh = c(0, 0.102635, 0.12228, 0.141925, 0.16157, 0.181215, 0.20086, 0.220505, 0.24015, 0.
259795, 0.27944, 0.299085, 0.31873, 0.338375)

VWC_tracker = matrix(nrow = 0, ncol = numDays+1)
irr_tracker = matrix(nrow = 0, ncol = numDays+1)

#Run simulation day by day and predict VWC
tempVWC = startVWC
for (i in 1:14){
  #Gather values
  VWC_Pred = c(startVWC)
  Irrigation = c()
  tempVWC = startVWC
  for (j in 1:numDays) {
    tempWB = ET_vals[j] + Rain_vals[j]
    irr_val = 0
    irr = 0
    if (tempVWC < VWC_thresh[i]){ #Decide if irrigation should happen today
      tempWB = tempWB + irrigation_val
      irr = 1
      irr_val = irrigation_val
    }
    #Set values into prediction
    tempDF = data.frame(data.Mean.VWC = tempVWC, data.CWB = tempWB, data.CRain = Rain_vals[j], d
```

```
ata.CET = ET_vals[j], data.CIrr = irr_val)
    tempResp = predict(model, newdata = tempDF, type = 'response')
    tempVWC = tempResp
    VWC_Pred = c(VWC_Pred, tempVWC) #Predict
    Irrigation = c(Irrigation, irr)
  }

  if (tempVWC < VWC_thresh[i]){ #Irrigation on final day
    irr = 1
  }
  Irrigation = c(Irrigation, irr)

  #Add to tracker matrix
  VWC_tracker = invisible(rbind(VWC_tracker, VWC_Pred))
  irr_tracker = invisible(rbind(irr_tracker, Irrigation))
}
```

View Raw data and send to CSV file.

```
df1 = as.data.frame(VWC_tracker)
row.names(df1)[row.names(df1) == "1"] = "0 (0)"
row.names(df1)[row.names(df1) == "2"] = "0.103 (15)"
row.names(df1)[row.names(df1) == "3"] = "0.122 (20)"
row.names(df1)[row.names(df1) == "4"] = "0.142 (25)"
row.names(df1)[row.names(df1) == "5"] = "0.162 (30)"
row.names(df1)[row.names(df1) == "6"] = "0.181 (35)"
row.names(df1)[row.names(df1) == "7"] = "0.201 (40)"
row.names(df1)[row.names(df1) == "8"] = "0.221 (45)"
row.names(df1)[row.names(df1) == "9"] = "0.240 (50)"
row.names(df1)[row.names(df1) == "10"] = "0.260 (55)"
row.names(df1)[row.names(df1) == "11"] = "0.279 (60)"
row.names(df1)[row.names(df1) == "12"] = "0.299 (65)"
row.names(df1)[row.names(df1) == "13"] = "0.319 (70)"
row.names(df1)[row.names(df1) == "14"] = "0.338 (75)"

colnames(df1) <- c("Day 0","Day 1","Day 2", "Day 3")

write.csv(df1, "VWC_predictions.csv", row.names=TRUE)
VWCdisplay = read.csv("VWC_predictions.csv")
VWCdisplay
```

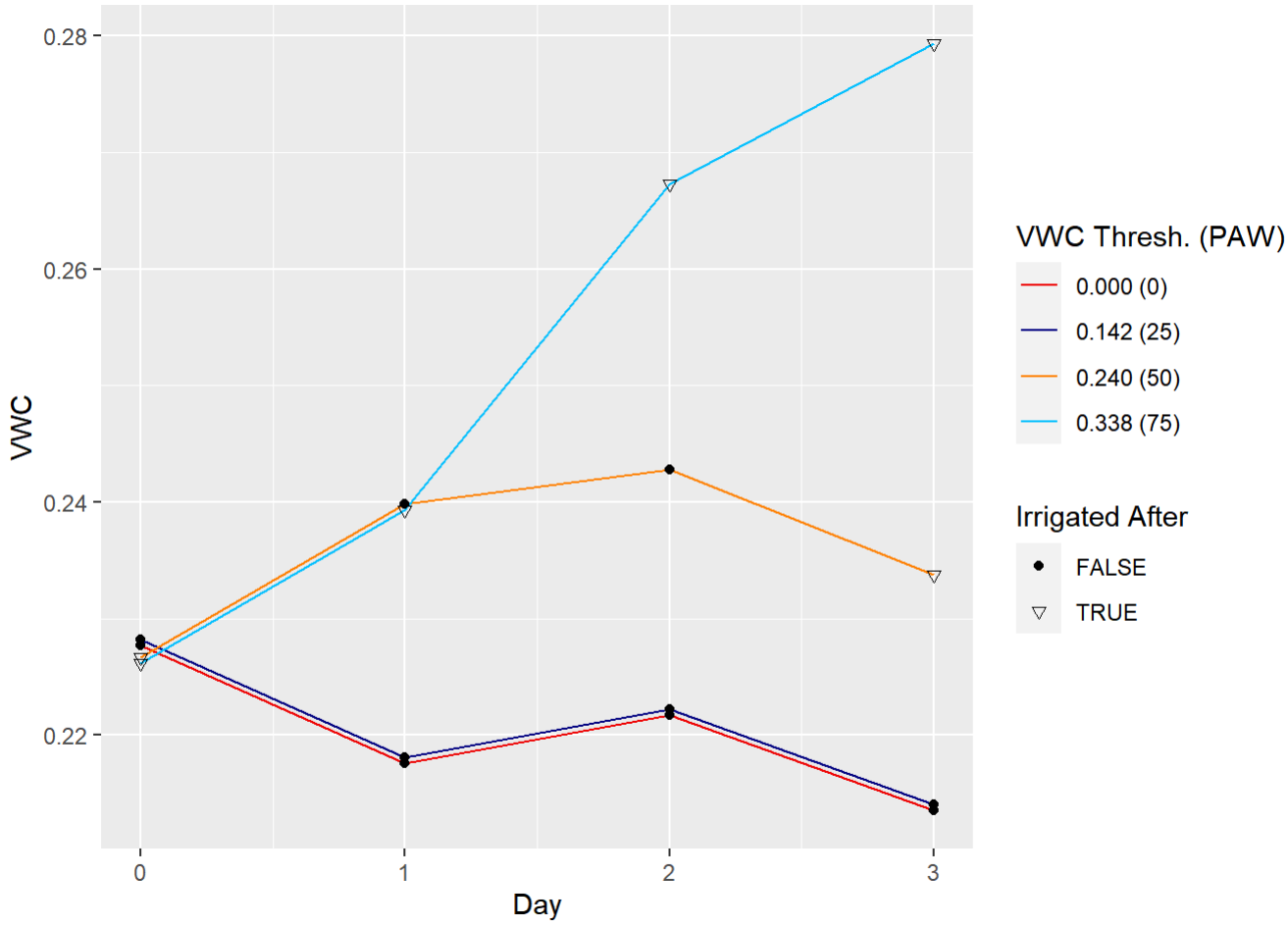
```
##              X     Day.0     Day.1     Day.2     Day.3
## 1     VWC_Pred 0.2272083 0.2170966 0.2212217 0.2130151
## 2   VWC_Pred.1 0.2272083 0.2170966 0.2212217 0.2130151
## 3   VWC_Pred.2 0.2272083 0.2170966 0.2212217 0.2130151
## 4   VWC_Pred.3 0.2272083 0.2170966 0.2212217 0.2130151
## 5   VWC_Pred.4 0.2272083 0.2170966 0.2212217 0.2130151
## 6   VWC_Pred.5 0.2272083 0.2170966 0.2212217 0.2130151
## 7   VWC_Pred.6 0.2272083 0.2170966 0.2212217 0.2130151
## 8   VWC_Pred.7 0.2272083 0.2170966 0.2478671 0.2387671
## 9   VWC_Pred.8 0.2272083 0.2403623 0.2432538 0.2343086
## 10  VWC_Pred.9 0.2272083 0.2403623 0.2682926 0.2585079
## 11 VWC_Pred.10 0.2272083 0.2403623 0.2682926 0.2803476
## 12 VWC_Pred.11 0.2272083 0.2403623 0.2682926 0.2803476
## 13 VWC_Pred.12 0.2272083 0.2403623 0.2682926 0.2803476
## 14 VWC_Pred.13 0.2272083 0.2403623 0.2682926 0.2803476
```

```
df2 = as.data.frame(irr_tracker)
colnames(df2) <- c("Day 0-1","Day 1-2","Day 2-3", "Day 3-4")
row.names(df2)[row.names(df2) == "1"] = "0 (0)"
row.names(df2)[row.names(df2) == "2"] = "0.103 (15)"
row.names(df2)[row.names(df2) == "3"] = "0.122 (20)"
row.names(df2)[row.names(df2) == "4"] = "0.142 (25)"
row.names(df2)[row.names(df2) == "5"] = "0.162 (30)"
row.names(df2)[row.names(df2) == "6"] = "0.181 (35)"
row.names(df2)[row.names(df2) == "7"] = "0.201 (40)"
row.names(df2)[row.names(df2) == "8"] = "0.221 (45)"
row.names(df2)[row.names(df2) == "9"] = "0.240 (50)"
row.names(df2)[row.names(df2) == "10"] = "0.260 (55)"
row.names(df2)[row.names(df2) == "11"] = "0.279 (60)"
row.names(df2)[row.names(df2) == "12"] = "0.299 (65)"
row.names(df2)[row.names(df2) == "13"] = "0.319 (70)"
row.names(df2)[row.names(df2) == "14"] = "0.338 (75)"


write.csv(df2, "irrigation_predictions.csv", row.names=TRUE)
irrdisplay =read.csv("irrigation_predictions.csv")
irrdisplay
```

```
##                X Day.0.1 Day.1.2 Day.2.3 Day.3.4
## 1     Irrigation       0       0       0       0
## 2   Irrigation.1       0       0       0       0
## 3   Irrigation.2       0       0       0       0
## 4   Irrigation.3       0       0       0       0
## 5   Irrigation.4       0       0       0       0
## 6   Irrigation.5       0       0       0       0
## 7   Irrigation.6       0       0       0       0
## 8   Irrigation.7       0       1       0       0
## 9   Irrigation.8       1       0       0       1
## 10  Irrigation.9       1       1       0       1
## 11 Irrigation.10       1       1       1       1
## 12 Irrigation.11       1       1       1       1
## 13 Irrigation.12       1       1       1       1
## 14 Irrigation.13       1       1       1       1
```

Create final plot. Users can change which lines are plotted by changing the if else statements below.

```
#Load only necessary points
VWC = c()
Day = c()
Threshold = c()
Irrigated = c()
threshold_vals = c("0.000 (0)", "0.103 (15)", "0.122 (20)", "0.142 (25)", "0.162 (30)", "0.181
(35)", "0.201 (40)", "0.221 (45)", "0.240 (50)", "0.260 (55)", "0.279 (60)", "0.299 (65)", "0.31
9 (70)", "0.338 (75)")

#Update irrigation True/False
for (i in 1:(numDays+1)) {
  for (j in 1:4){
    thr = 0
    offset = 0
    if (j == 1){thr = 1; offset = 0.0005}
    else if (j == 2){thr = 4; offset = 0.001}
    else if (j == 3){thr = 9; offset = -0.0005}
    else if (j == 4){thr = 14; offset = -0.001}
    Day = c(Day, (i-1))
    VWC = c(VWC, VWC_tracker[thr, i] + offset)
    Threshold = c(Threshold, threshold_vals[thr])
    Irrigated = c(Irrigated, as.logical(irr_tracker[thr, i]))
  }
}

#Plot
df = data.frame(VWC,Day,Threshold,Irrigated)
ggplot(data=df, aes(x=Day, y=VWC, shape=Irrigated))+geom_line(aes(colour=Threshold,group=interac
tion(Threshold)))+geom_point()+scale_shape_manual(values=c(16,6))+scale_color_manual(values=c("r
ed2","navy","darkorange1","deepskyblue"))+labs(shape = "Irrigated After", colour = "VWC Thresh.
(PAW)")
```


0.28
0.26
0.24
0.22
VWC
0
1
2
3
Day
VWC Thresh. (PAW)
0.000 (0)
0.142 (25)
0.240 (50)
0.338 (75)
Irrigated After
FALSE
TRUE