Data Modification Work space

> Import Data

```
crop <- read.csv("../data/Crop_recommendation.csv")
print(head(crop))
```

```
##    N  P  K temperature humidity       ph rainfall label
## 1 90 42 43    20.87974 82.00274 6.502985 202.9355  rice
## 2 85 58 41    21.77046 80.31964 7.038096 226.6555  rice
## 3 60 55 44    23.00446 82.32076 7.840207 263.9642  rice
## 4 74 35 40    26.49110 80.15836 6.980401 242.8640  rice
## 5 78 42 42    20.13017 81.60487 7.628473 262.7173  rice
## 6 69 37 42    23.05805 83.37012 7.073454 251.0550  rice
```

```
str(crop)
```

```
## 'data.frame':    2200 obs. of  8 variables:
##  $ N          : int  90 85 60 74 78 69 69 94 89 68 ...
##  $ P          : int  42 58 55 35 42 37 55 53 54 58 ...
##  $ K          : int  43 41 44 40 42 42 38 40 38 38 ...
##  $ temperature: num  20.9 21.8 23 26.5 20.1 ...
##  $ humidity   : num  82 80.3 82.3 80.2 81.6 ...
##  $ ph         : num  6.5 7.04 7.84 6.98 7.63 ...
##  $ rainfall   : num  203 227 264 243 263 ...
##  $ label      : chr  "rice" "rice" "rice" "rice" ...
```

Crops currently In the table

```
cropsLabel <- unique(crop$label)
print(cropsLabel)
```

```
##  [1] "rice"        "maize"       "chickpea"    "kidneybeans" "pigeonpeas"
##  [6] "mothbeans"   "mungbean"    "blackgram"   "lentil"      "pomegranate"
## [11] "banana"      "mango"       "grapes"      "watermelon"  "muskmelon"
## [16] "apple"       "orange"      "papaya"      "coconut"     "cotton"
## [21] "jute"        "coffee"
```

> Load Kenyan Cash Crops To be Used

Original Crop	Matched Kenyan Export Crop	Conditions Rationale
Rice	Tea	High rainfall, humid, acidic soil (pH ~5.5-6.5)
Maize	Coffee	Warm climate, moderate rainfall, slightly acidic to neutral soil
Chickpea	Avocado	Warm to hot climates, moderate rainfall, neutral to slightly acidic soil
Kidneybeans	Macadamia Nuts	Warm climates, moderate to high rainfall, slightly acidic soil
Pigeonpeas	French Beans	Warm climates, moderate rainfall, similar legume characteristics
Mothbeans	Snow Peas	Grows well in cool climates with moderate rainfall
Mungbean	Passion Fruit	Warm, humid, tropical environments, moderate to heavy rainfall
Blackgram	Mango	Warm, tropical climate, moderate rainfall, slightly acidic soil
Lentil	Pineapple	Tropical climates with good rainfall, grows well in acidic soils
Pomegranate	Flowers (Roses)	Moderate climates, well-drained soil, moderate rainfall
Banana	Cabbage	Grows in warm climates with moderate to heavy rainfall
Mango	Sugarcane	Tropical and subtropical climates, needs high rainfall and warm temperatures
Grapes	Cashew Nuts	Warm, dry conditions with moderate rainfall, grows in well-drained soil
Watermelon	Tomatoes	Warm climates, moderate water needs, similar soil preferences

Original Crop	Matched Kenyan Export Crop	Conditions Rationale
Muskmelon	Spinach	Prefers warm temperatures and moderate moisture
Apple	Carrots	Grows in cooler climates with moderate rainfall
Orange	Coconuts	Thrives in tropical climates with heavy rainfall and warm temperatures
Papaya	Sisal	Grows in semi-arid to dry conditions, requires warm temperatures
Coconut	Sesame Seeds	Thrives in warm, dry conditions with moderate rainfall
Cotton	Tobacco	Grows in warm to hot climates with moderate rainfall
Jute	Chillies	Grows in warm climates with moderate humidity and rainfall
Coffee	Pyrethrum	Grows well in high altitudes with cooler temperatures

```r
# Mapped Kenyan export crops based on environmental conditions

kenya_export_crops <- c("Tea","Coffee","Avocado","Macadamia Nuts","French Beans","Snow Peas","Pass
ion Fruit","Mango","Pineapple","Flowers (Roses)","Cabbage","Sugarcane","Cashew Nuts","Tomatoes","S
pinach","Carrots","Coconuts","Sisal","Sesame Seeds","Tobacco","Chillies","Pyrethrum")
```

```r
print(kenya_export_crops)
```

```
##  [1] "Tea"             "Coffee"          "Avocado"         "Macadamia Nuts"
##  [5] "French Beans"    "Snow Peas"       "Passion Fruit"   "Mango"
##  [9] "Pineapple"       "Flowers (Roses)" "Cabbage"         "Sugarcane"
## [13] "Cashew Nuts"     "Tomatoes"        "Spinach"         "Carrots"
## [17] "Coconuts"        "Sisal"           "Sesame Seeds"    "Tobacco"
## [21] "Chillies"        "Pyrethrum"
```

```r
# Ensure there are no duplicates in the Kenyan exports list
if (length(unique(kenya_export_crops)) == length(kenya_export_crops)) {
    print("No duplicates in the Kenyan export crops.")
} else {
```

```
    print("There are duplicates in the Kenyan export crops!")
}
```

```
## [1] "No duplicates in the Kenyan export crops."
```

```
# Print original and mapped crop labels to verify
for (i in 1:length(cropsLabel)) {
  cat("Original Crop:", cropsLabel[i], "->", "Kenyan Export Crop:", kenya_export_crops[i], "\n")
}
```

```
## Original Crop: rice -> Kenyan Export Crop: Tea
## Original Crop: maize -> Kenyan Export Crop: Coffee
## Original Crop: chickpea -> Kenyan Export Crop: Avocado
## Original Crop: kidneybeans -> Kenyan Export Crop: Macadamia Nuts
## Original Crop: pigeonpeas -> Kenyan Export Crop: French Beans
## Original Crop: mothbeans -> Kenyan Export Crop: Snow Peas
## Original Crop: mungbean -> Kenyan Export Crop: Passion Fruit
## Original Crop: blackgram -> Kenyan Export Crop: Mango
## Original Crop: lentil -> Kenyan Export Crop: Pineapple
## Original Crop: pomegranate -> Kenyan Export Crop: Flowers (Roses)
## Original Crop: banana -> Kenyan Export Crop: Cabbage
## Original Crop: mango -> Kenyan Export Crop: Sugarcane
## Original Crop: grapes -> Kenyan Export Crop: Cashew Nuts
## Original Crop: watermelon -> Kenyan Export Crop: Tomatoes
## Original Crop: muskmelon -> Kenyan Export Crop: Spinach
## Original Crop: apple -> Kenyan Export Crop: Carrots
## Original Crop: orange -> Kenyan Export Crop: Coconuts
## Original Crop: papaya -> Kenyan Export Crop: Sisal
## Original Crop: coconut -> Kenyan Export Crop: Sesame Seeds
## Original Crop: cotton -> Kenyan Export Crop: Tobacco
## Original Crop: jute -> Kenyan Export Crop: Chillies
## Original Crop: coffee -> Kenyan Export Crop: Pyrethrum
```

```
# Create a named vector for easy replacement (map original crops to Kenyan crops)
crop_mapping <- setNames(kenya_export_crops, cropsLabel)


# Replace the 'label' column in the data frame with the mapped values
crop$label <- crop_mapping[crop$label]

# Print the first few rows to verify the replacements
print(head(crop))
```

```
##    N  P  K temperature humidity       ph rainfall label
## 1 90 42 43    20.87974 82.00274 6.502985 202.9355   Tea
## 2 85 58 41    21.77046 80.31964 7.038096 226.6555   Tea
## 3 60 55 44    23.00446 82.32076 7.840207 263.9642   Tea
## 4 74 35 40    26.49110 80.15836 6.980401 242.8640   Tea
```

```
## 5 78 42 42    20.13017 81.60487 7.628473 262.7173   Tea
## 6 69 37 42    23.05805 83.37012 7.073454 251.0550   Tea
```

> Export The new modified Kenyan Cash crops Data

```
# Specify the file name
output_file <- "../data/kenyan_cash_crops_conditions.csv"

# Export the data to a CSV file
write.csv(crop, file = output_file, row.names = FALSE)

# Confirmation message
cat("Data has been successfully exported to", output_file, "\n")
```

```
## Data has been successfully exported to ../data/kenyan_cash_crops_conditions.csv
```

Data Cleaning Work space

Load Data

```
kenyancrops <- read.csv("../data/kenyan_cash_crops_conditions.csv")
```

Check for Missing values

```
missing_vals <- sapply(kenyancrops, function(x) sum(is.na(x)))
print(missing_vals)
```

```
##           N           P           K temperature    humidity          ph
##           0           0           0           0           0           0
##    rainfall       label
##           0           0
```

Check for Duplicate Values

```
duplicate_rows <- kenyancrops[duplicated(kenyancrops), ]

# Count the number of duplicate rows
num_duplicates <- nrow(duplicate_rows)

# Print the number of duplicate rows and the duplicate rows themselves
print(num_duplicates)
```

```
## [1] 0
```

```
print(duplicate_rows)
```

```
## [1] N           P           K           temperature humidity    ph
## [7] rainfall    label
## <0 rows> (or 0-length row.names)
```

Count Unique Values in a Column

```
label_counts <- table(kenyancrops$label)

#print(label_counts)

# Print each crop and its count in a well-aligned format
for (i in names(label_counts)) {
  cat(sprintf("%-20s %d\n", i, label_counts[i]))
}
```

```
## Avocado              100
## Cabbage              100
## Carrots              100
## Cashew Nuts          100
## Chillies             100
## Coconuts             100
## Coffee               100
## Flowers (Roses)      100
## French Beans         100
## Macadamia Nuts       100
## Mango                100
## Passion Fruit        100
## Pineapple            100
## Pyrethrum            100
## Sesame Seeds         100
## Sisal                100
## Snow Peas            100
## Spinach              100
## Sugarcane            100
## Tea                  100
## Tobacco              100
## Tomatoes             100
```

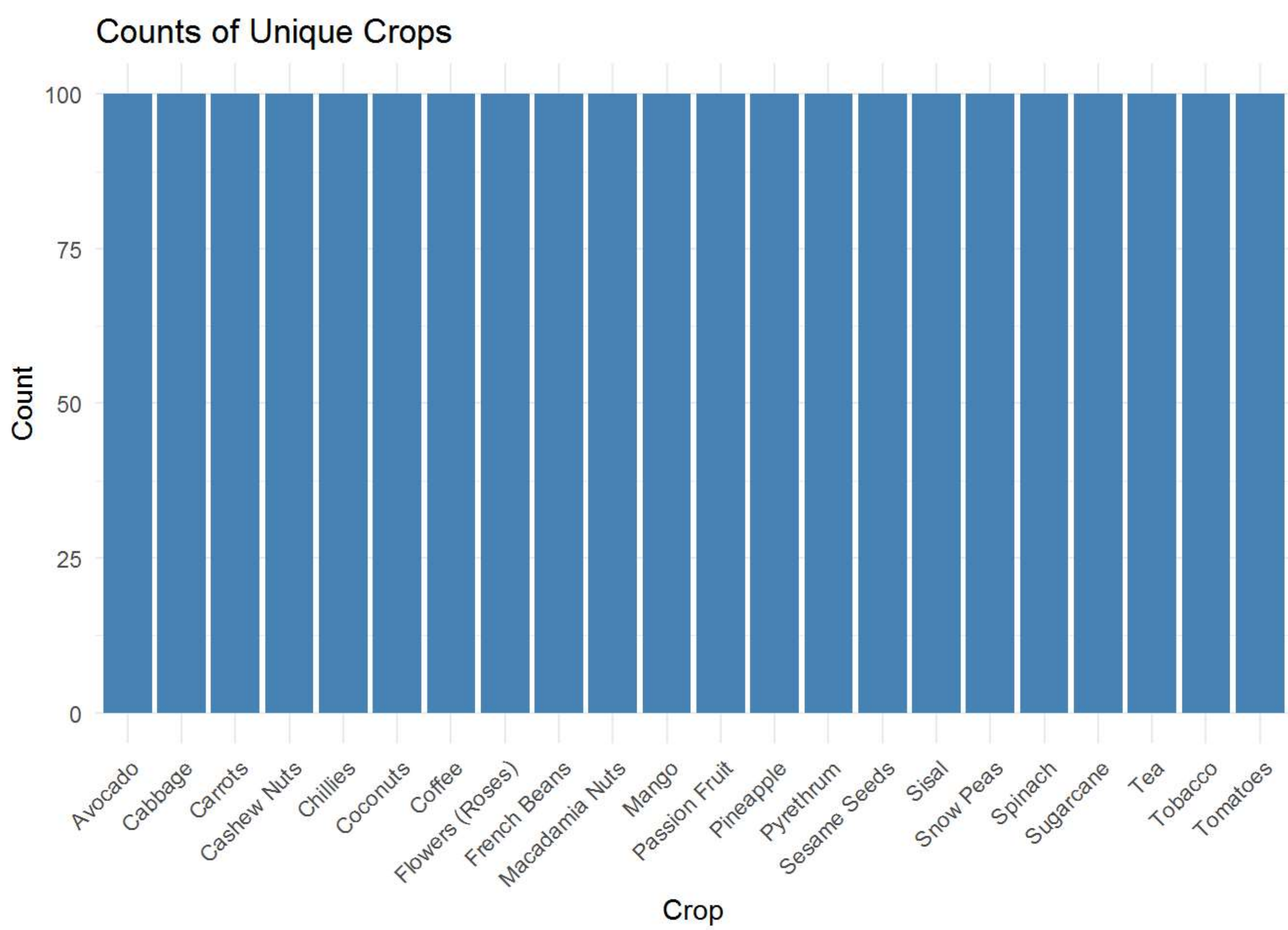
Data Visualization Work space

```
# Packages Installation and Loading

#install.packages("ggplot2")
```

```r
library(ggplot2)
```

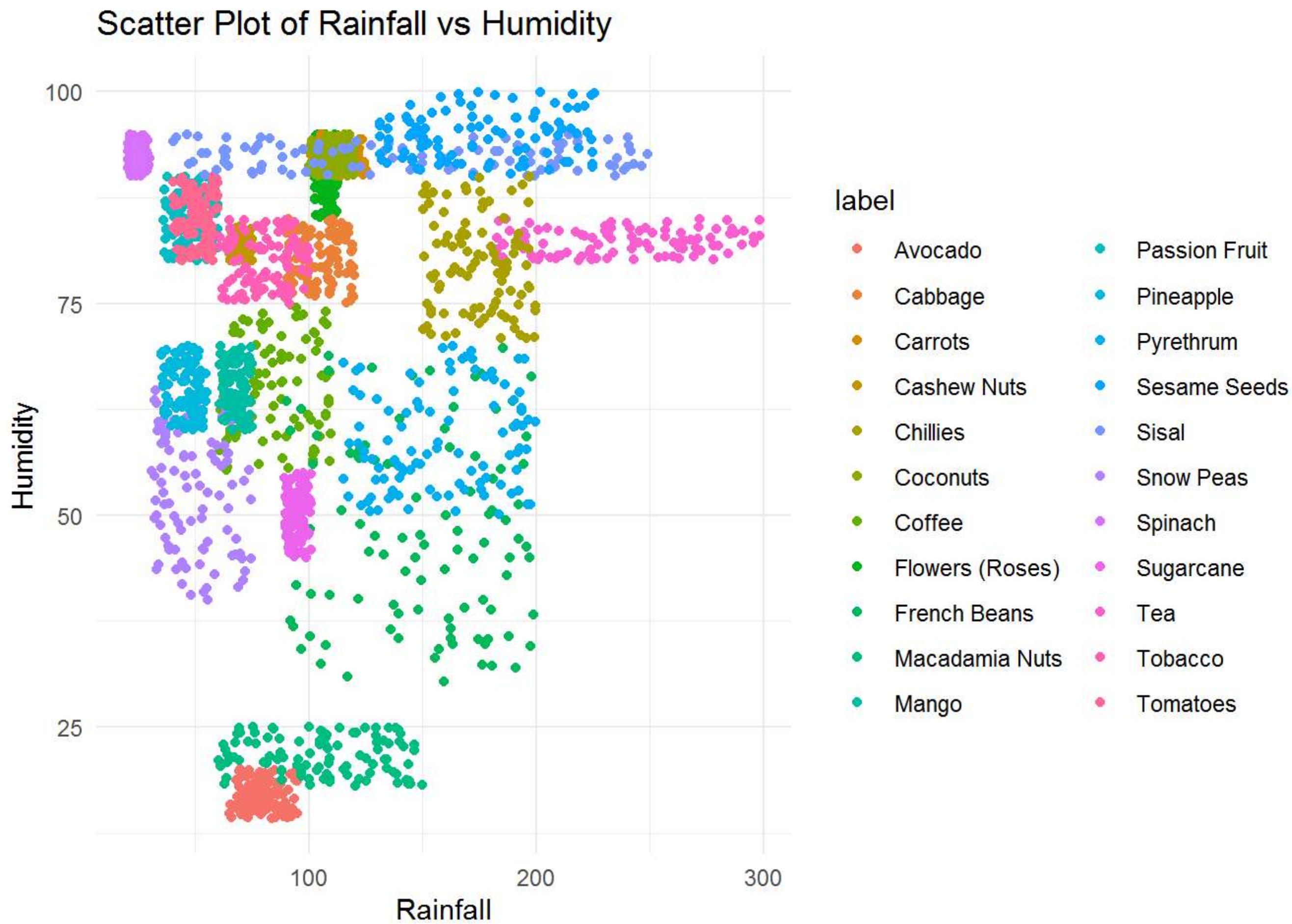
Bar Plot for Crop Counts

```r
#label_counts <- table(kenyancrops$label)  # Count unique values in 'label'

ggplot(data = as.data.frame(label_counts), aes(x = Var1, y = Freq)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Counts of Unique Crops", x = "Crop", y = "Count") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



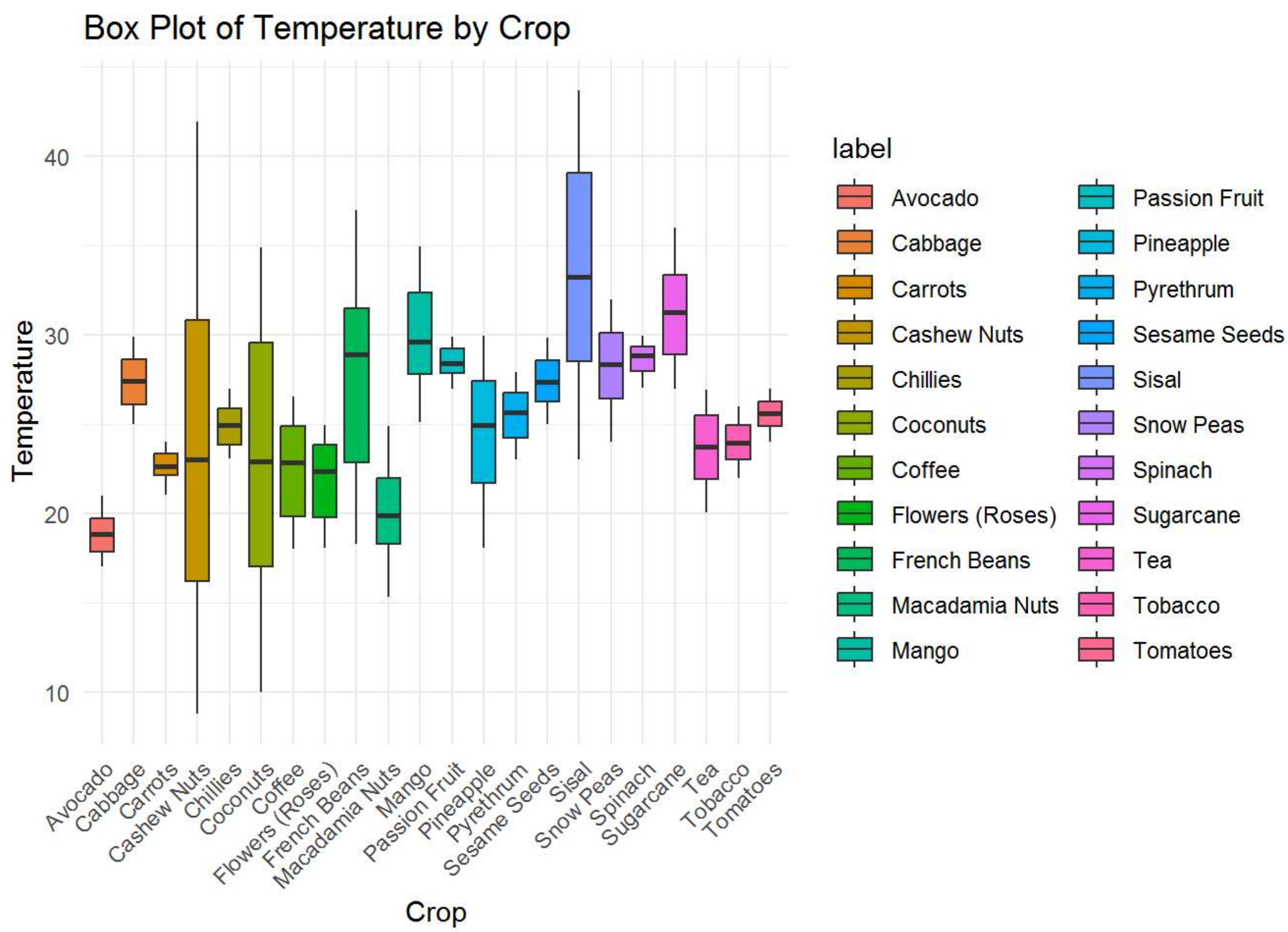
Scatter Plot for Rainfall vs. Humidity

```r
ggplot(kenyancrops, aes(x = rainfall, y = humidity, color = label)) +
  geom_point() +
  labs(title = "Scatter Plot of Rainfall vs Humidity", x = "Rainfall", y = "Humidity") +
  theme_minimal()
```



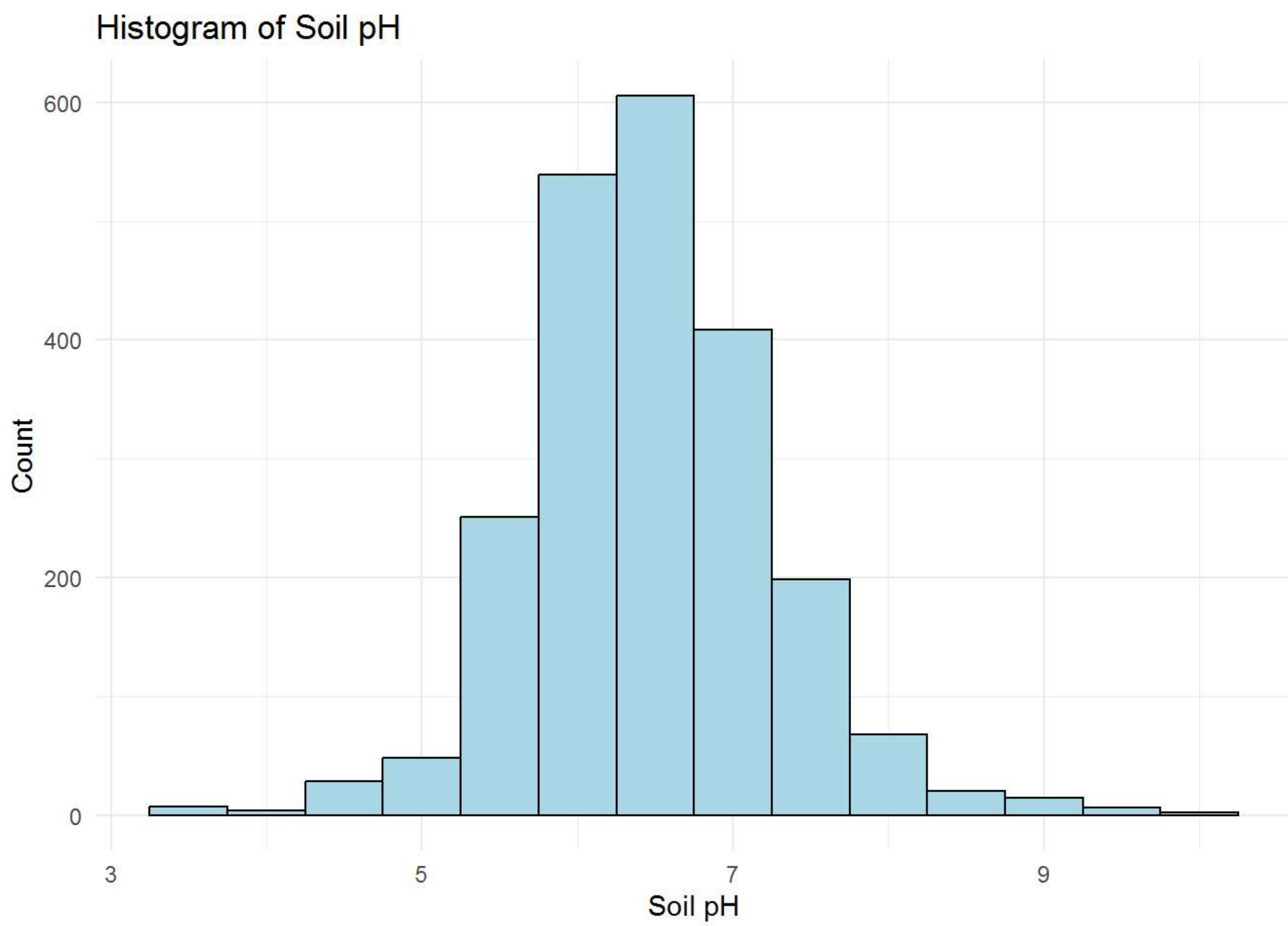
Box Plot for Temperature by Crop

```
# Create a box plot for temperature by crop
ggplot(kenyancrops, aes(x = label, y = temperature, fill = label)) +
  geom_boxplot() +
  labs(title = "Box Plot of Temperature by Crop", x = "Crop", y = "Temperature") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Histogram for Soil pH

```
# Create a histogram for soil pH
ggplot(kenyancrops, aes(x = ph)) +
  geom_histogram(binwidth = 0.5, fill = "lightblue", color = "black") +
  labs(title = "Histogram of Soil pH", x = "Soil pH", y = "Count") +
  theme_minimal()
```



MODEL WORKSPACE

Load necessary libraries

```
#install.packages(c("dplyr", "caret", "randomForest", "e1071","nnet"))


library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: lattice
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(e1071)
library(nnet)
```

> # 1. Encoding the crops

```r
# Encoding the Kenyan crops
crop_dict <- list(
    'Tea' = 1, 'Coffee' = 2, 'Avocado' = 3, 'Macadamia Nuts' = 4,
    'French Beans' = 5, 'Snow Peas' = 6, 'Passion Fruit' = 7,
    'Mango' = 8, 'Pineapple' = 9, 'Flowers (Roses)' = 10, 'Cabbage' = 11,
    'Sugarcane' = 12, 'Cashew Nuts' = 13, 'Tomatoes' = 14, 'Spinach' = 15,
    'Carrots' = 16, 'Coconuts' = 17, 'Sisal' = 18, 'Sesame Seeds' = 19,
    'Tobacco' = 20, 'Chillies' = 21, 'Pyrethrum' = 22
)

# Apply encoding using match() and convert to factor
kenyancrops$crop_num <- as.factor(match(kenyancrops$label, names(crop_dict)))

# Drop the 'label' column
kenyancrops <- kenyancrops %>% select(-label)
```

2. Train Test Split

```
set.seed(42)

# Split the data into training and testing sets (80% training, 20% testing)
trainIndex <- createDataPartition(kenyancrops$crop_num, p = 0.8, list = FALSE)
trainData <- kenyancrops[trainIndex,]
testData <- kenyancrops[-trainIndex,]

# Separate features (X) and target (Y)
x_train <- trainData %>% select(-crop_num)
y_train <- trainData$crop_num
x_test <- testData %>% select(-crop_num)
y_test <- testData$crop_num
```

3. Feature Scaling

```
# Scale the features using Min-Max Scaling
preProcess_scale <- preProcess(x_train, method = 'range')
x_train_scaled <- predict(preProcess_scale, x_train)
x_test_scaled <- predict(preProcess_scale, x_test)
```

4. Model Training

```
# Define trainControl
control <- trainControl(method = "cv", number = 5, verboseIter = FALSE)

# Initialize models training algorithms
models <- list(
    multinom = train(x_train_scaled, y_train, method = "multinom", trControl = control), # Multino
mial Logistic Regression
    rf = train(x_train_scaled, y_train, method = "rf", trControl = control), # Random Forest
    svc = train(x_train_scaled, y_train, method = "svmRadial", trControl = control), # Support Vec
tor Machine
    knn = train(x_train_scaled, y_train, method = "knn", trControl = control), #  K-Nearest Neighb
ors
    dt = train(x_train_scaled, y_train, method = "rpart", trControl = control), # Decision Tree
    gnb = train(x_train_scaled, y_train, method = "naive_bayes", trControl = control) # Gaussian N
aive Bayes
)
```

```
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1392.408808
## iter  20 value 598.356510
## iter  30 value 150.617383
## iter  40 value 55.836509
## iter  50 value 38.091154
```

```
## iter  60 value 33.438349
## iter  70 value 29.610064
## iter  80 value 28.283051
## iter  90 value 26.890468
## iter 100 value 26.337094
## final  value 26.337094
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1620.387125
## iter  20 value 1323.793859
## iter  30 value 1239.678862
## iter  40 value 1227.263072
## final  value 1227.259944
## converged
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1392.681822
## iter  20 value 599.782698
## iter  30 value 159.468580
## iter  40 value 83.815674
## iter  50 value 74.126553
## iter  60 value 70.984657
## iter  70 value 68.257749
## iter  80 value 66.503559
## iter  90 value 65.564606
## iter 100 value 64.853803
## final  value 64.853803
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1270.997584
## iter  20 value 537.301229
## iter  30 value 155.727432
## iter  40 value 51.942930
## iter  50 value 37.937947
## iter  60 value 31.757848
## iter  70 value 28.958859
## iter  80 value 28.040536
## iter  90 value 27.207469
## iter 100 value 26.309986
## final  value 26.309986
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1595.961910
## iter  20 value 1350.933947
## iter  30 value 1253.354404
## iter  40 value 1238.460747
## final  value 1238.454841
## converged
## # weights:  198 (168 variable)
```

```
## initial  value 4352.187774
## iter  10 value 1271.365506
## iter  20 value 538.654138
## iter  30 value 162.768179
## iter  40 value 82.878064
## iter  50 value 73.370040
## iter  60 value 69.494957
## iter  70 value 67.743029
## iter  80 value 66.315128
## iter  90 value 65.558255
## iter 100 value 64.855441
## final  value 64.855441
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1272.989713
## iter  20 value 526.192296
## iter  30 value 160.174664
## iter  40 value 44.444669
## iter  50 value 34.713022
## iter  60 value 27.276522
## iter  70 value 24.824351
## iter  80 value 23.317571
## iter  90 value 22.226929
## iter 100 value 21.539301
## final  value 21.539301
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1593.634817
## iter  20 value 1301.165551
## iter  30 value 1250.613661
## iter  40 value 1237.111829
## final  value 1237.107858
## converged
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1273.348060
## iter  20 value 527.715026
## iter  30 value 168.524326
## iter  40 value 78.727028
## iter  50 value 70.246736
## iter  60 value 66.969109
## iter  70 value 64.566851
## iter  80 value 63.236505
## iter  90 value 62.490037
## iter 100 value 61.929925
## final  value 61.929925
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1271.051589
```

```
## iter  20 value 570.766415
## iter  30 value 148.769994
## iter  40 value 48.544839
## iter  50 value 35.122404
## iter  60 value 30.783834
## iter  70 value 27.375370
## iter  80 value 25.351904
## iter  90 value 24.145112
## iter 100 value 23.962318
## final  value 23.962318
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1584.445402
## iter  20 value 1321.608227
## iter  30 value 1251.227296
## iter  40 value 1236.093686
## final  value 1236.090414
## converged
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1271.403532
## iter  20 value 572.320457
## iter  30 value 156.153562
## iter  40 value 78.166344
## iter  50 value 70.634138
## iter  60 value 67.424252
## iter  70 value 65.236406
## iter  80 value 63.678617
## iter  90 value 62.804415
## iter 100 value 62.178644
## final  value 62.178644
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1245.363270
## iter  20 value 647.702621
## iter  30 value 201.920854
## iter  40 value 51.156329
## iter  50 value 35.153663
## iter  60 value 29.944133
## iter  70 value 27.153899
## iter  80 value 24.563437
## iter  90 value 23.932465
## iter 100 value 23.526161
## final  value 23.526161
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1582.958661
## iter  20 value 1328.318534
## iter  30 value 1247.448714
```

```
## iter  40 value 1231.786529
## final  value 1231.782708
## converged
## # weights:  198 (168 variable)
## initial  value 4352.187774
## iter  10 value 1245.746993
## iter  20 value 649.103196
## iter  30 value 206.961866
## iter  40 value 82.152265
## iter  50 value 72.258901
## iter  60 value 69.056201
## iter  70 value 67.068855
## iter  80 value 65.702499
## iter  90 value 64.862654
## iter 100 value 64.280272
## final  value 64.280272
## stopped after 100 iterations
## # weights:  198 (168 variable)
## initial  value 5440.234718
## iter  10 value 1486.926432
## iter  20 value 784.057704
## iter  30 value 181.109043
## iter  40 value 92.753697
## iter  50 value 84.413158
## iter  60 value 80.846252
## iter  70 value 78.543801
## iter  80 value 77.291348
## iter  90 value 76.415418
## iter 100 value 75.755693
## final  value 75.755693
## stopped after 100 iterations
```

```
# Evaluate machine learning models
model_accuracies <- sapply(models, function(model) {
    if (!is.null(model)) {
        y_pred <- predict(model, x_test_scaled)
        confusionMatrix(y_pred, y_test)$overall['Accuracy']
    } else {
        NA
    }
}, USE.NAMES = TRUE)

# Print model accuracies
print("Model Accuracies:")
```

```
## [1] "Model Accuracies:"
```

```
print(model_accuracies)
```

```
## multinom.Accuracy       rf.Accuracy      svc.Accuracy      knn.Accuracy
##          0.9795455         0.9954545         0.9840909         0.9840909
##        dt.Accuracy      gnb.Accuracy
##          0.9022727         0.9954545
```

```r
# Select the best model based on accuracy
best_model_name <- names(which.max(model_accuracies))
print(paste("Best model:", best_model_name))
```

```
## [1] "Best model: rf.Accuracy"
```

5. Fit the Best (Random Forest) model

```r
# Fit the Random Forest model (as it's likely the best)
if (best_model_name == "rf") {
    rfc <- models$rf
} else {
    rfc <- randomForest(x_train_scaled, y_train)  # Fallback to RF if not best
}

# Define a predictive function using the trained model
recommendation <- function(N, P, K, temperature, humidity, ph, rainfall) {
    # Create a new data frame with the exact column names from your dataset
    features <- data.frame(
        N = N, P = P, K = K,
        temperature = temperature, humidity = humidity,
        ph = ph, rainfall = rainfall
    )

    # Ensure the input columns match the training data columns
    features <- features[, colnames(x_train), drop = FALSE]

    # Scale the input features using the same scaler used in training
    features_scaled <- predict(preProcess_scale, features)

    # Make a prediction using the trained Random Forest model
    prediction <- predict(rfc, features_scaled)
    return(prediction)
}
```

6. Predictive System Testing

```r
# sample input
N <- 20; P <- 30; K <- 40
temperature <- 40; humidity <- 20
ph <- 30; rainfall <- 50
```

```r
# Get the crop prediction
predict_crop <- recommendation(N, P, K, temperature, humidity, ph, rainfall)

# Crop dictionary for output
crop_dict_rev <- c(
    '1' = 'Tea', '2' = 'Coffee', '3' = 'Avocado', '4' = 'Macadamia Nuts',
    '5' = 'French Beans', '6' = 'Snow Peas', '7' = 'Passion Fruit',
    '8' = 'Mango', '9' = 'Pineapple', '10' = 'Flowers (Roses)',
    '11' = 'Cabbage', '12' = 'Sugarcane', '13' = 'Cashew Nuts',
    '14' = 'Tomatoes', '15' = 'Spinach', '16' = 'Carrots',
    '17' = 'Coconuts', '18' = 'Sisal', '19' = 'Sesame Seeds',
    '20' = 'Tobacco', '21' = 'Chillies', '22' = 'Pyrethrum'
)

# Print the recommended crop
if (as.character(predict_crop) %in% names(crop_dict_rev)) {
    print(paste(crop_dict_rev[[as.character(predict_crop)]], "is the best crop to be cultivate
d."))
} else {
    print("Sorry, we are unable to recommend a crop for this environment.")
}
```

```
## [1] "Macadamia Nuts is the best crop to be cultivated."
```

Save the `rf` model along with the pre processing scaler for use in the Shiny app

```r
# Save the trained Random Forest model (rfc) to an RDS file
saveRDS(rfc, file = "../models/random_forest_model.rds")

# Save the pre-processing scaler (preProcess_scale) to an RDS file
saveRDS(preProcess_scale, file = "../models/preprocess_scaler.rds")
```