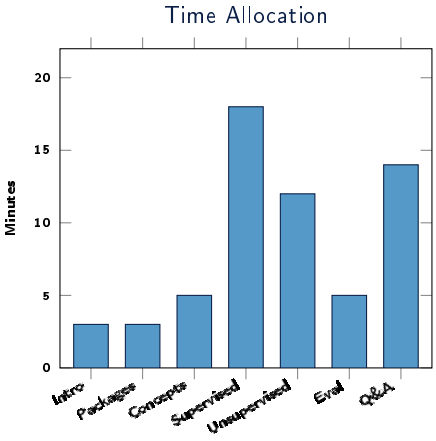
Machine Learning in R

R Workshop for Data Science

George Sichinga — Facilitator
Applied Data Science MSc Student, LUANAR
Specialises in Environmental & Climate Change Modelling
Skills: GIS, Python, R, Stata, Machine Learning

Today's Agenda

1. About R Users Malawi
2. **Key R Packages for ML**
3. What is Machine Learning?
4. Data Splitting: Train / Validation / Test
5. Regression vs Classification
6. **Supervised Learning**
   - Linear Regression
   - Logistic Regression
   - Decision Tree
   - Random Forest
   - Other important models
7. **Unsupervised Learning**
   - Clustering (K-Means, Hierarchical)
   - Dimension Reduction (PCA)
   - Association Rule Mining
8. Model Evaluation & Metrics
9. Choosing the Right Algorithm



Total: 60 minutes

Brief History

R Users Malawi was founded by **David Mwale** in 2024. David was studying for a Master's in Data Science at the University of Edinburgh when he discovered R user groups in Botswana and South Africa. After contacting the R Consortium and following their guidance, he registered the group. The **first meeting was held in late 2024**, promoted via LinkedIn, X, and WhatsApp.

Mission

Grow R adoption in Malawi alongside existing tools like Stata and SPSS, targeting universities, researchers and data professionals nationwide. We are actively looking for **co-organizers** — join us!

R Consortium RUGS

76,000+ members

across 90+ groups in 39 countries

Malawi — Growing!

LinkedIn, WhatsApp & Google Meet

+265 993 627 376 | +265 880 697 619

Key R Packages for Machine Learning

Package	What it does
caret	Unified ML training framework
tidymodels	Modern tidy ML ecosystem
randomForest	Random Forest algorithm
rpart	Decision Trees
e1071	SVM and Naive Bayes
xgboost	Gradient Boosting
arules	Association Rule Mining
factoextra	PCA / cluster visualisation
ggplot2	Visualisation
dplyr	Data wrangling

Install all at once

```r
# Run this once before the session
install.packages(c(
  "caret",          # main ML framework
  "tidymodels",   # tidy ML workflow
  "randomForest",
  "rpart",
  "e1071",
  "xgboost",
  "arules",
  "factoextra",
  "ggplot2",
  "dplyr"
))
# Load caret (used throughout today)
library(caret)
```

All packages are **completely free**. Source: cran.r-project.org

Simple Definition

Machine learning is a way of teaching a computer to **learn patterns from data** and make predictions or decisions on its own, without being told exactly what rules to follow every time.

Supervised Learning

The model learns from labelled examples. You give inputs AND correct answers.

Predict maize yield
Detect malaria risk
Loan default scoring

Unsupervised Learning

Finds hidden patterns. You provide inputs only. No labels or correct answers required.

Group districts by food security
Reduce data dimensions
Market basket analysis

Reinforcement Learning

Learns by trial and reward. Improves itself through feedback from its environment.

Game playing AI
Autonomous systems
Optimisation problems

Today: Supervised and Unsupervised Learning using the `caret` package.

Data Splitting: Train, Validation & Test Sets

Why Split Your Data?

You must never evaluate a model on the data you used to train it. That is like a student writing the exam and also marking it themselves. Splitting ensures your results are **honest and generalisable**.

Training set (60–70%) — the model learns from this.

Validation set (10–20%) — used to tune the model and pick hyper-parameters during development.

Test set (20–30%) — used *once* at the very end to report final performance. Never touch this until you are finished.

Cross-Validation

When data is small, use k-fold cross-validation instead. The data is split into k folds; the model trains on k-1 folds and validates on the remaining fold, rotating k times. `caret` handles this automatically via `trainControl`.

```r
library(caret)

# Full Malawi agricultural dataset
data <- data.frame(
  rainfall   = c(850,780,920,810,870,
                 760,900,820,740,880),
  fertilizer = c(50,40,60,45,55,
                 38,58,47,35,52),
  yield      = c(3.2,2.8,3.8,3.0,3.5,
                 2.6,3.7,3.1,2.5,3.4)
)

# Step 1: split into 70% train, 30% test
set.seed(42)
idx   <- createDataPartition(
           data$yield, p = 0.7, list = FALSE)
train <- data[idx, ]
test  <- data[-idx, ]

# Step 2: define 5-fold cross-validation
ctrl  <- trainControl(
           method = "cv",
           number = 5)
# ctrl is used when training models
# (shown in following slides)

cat("Train rows:", nrow(train), "\n")
cat("Test rows: ", nrow(test),  "\n")
```

Regression vs Classification

Regression

Used when the answer you want to predict is a **continuous number**. The model predicts values on a scale.

Malawi examples:

- How many tonnes of maize will a farmer harvest?
- What will rainfall be next month in Zomba?
- What is the expected weight of a child at age 2?

Classification

Used when the answer is a **category or group label**. The model assigns each observation to one group.

Malawi examples:

- Is this patient malaria positive or negative?
- Will this student pass or fail their exam?
- Is this district food secure, at risk, or in crisis?

Quick Rule of Thumb

If your outcome is a **number** (e.g. yield, income, temperature) use **Regression**. If your outcome is a **label or group** (e.g. yes/no, pass/fail, disease/healthy) use **Classification**.

Supervised Learning: Models Overview

Model	Task Type	Data Size	Best Strength
Linear Regression	Regression	Any	Simplest model, highly interpretable
Logistic Regression	Classification	Small-Med	Probability scores, interpretable
Decision Tree	Both	Small-Med	Easiest to explain to non-experts
Random Forest	Both	Med-Large	High accuracy, robust to noise
Naive Bayes	Classification	Small	Fast, great for categorical data
SVM	Classification	Small-Med	Small clean datasets
XGBoost	Both	Large	Competition-winning accuracy

Where to Start

Always begin with **Linear or Logistic Regression** as your baseline. These are simple, fast, and interpretable. Then try more complex models and compare their performance against the baseline.

What It Is and Why It Matters

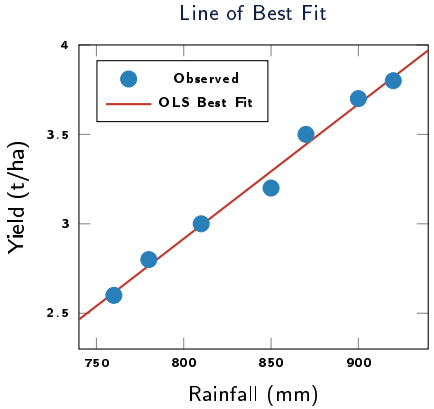
Linear Regression is the simplest machine learning model and the foundation of all other regression methods. Developed by Francis Galton in the 1880s, it fits a straight line through your data to describe the relationship between input(s) and an output. It remains one of the most widely used methods in research and data science.

Data Requirements

- Inputs (predictors): continuous numeric variables
- Output (target): one continuous numeric variable
- Requires: no missing values; roughly linear relationship; no severe outliers
- Libraries: base R `lm()` or `caret`

Prerequisites

Understand: mean, correlation, and the straight-line equation $y = mx + c$. No advanced maths needed beyond secondary school level.



OLS line:

$$\hat{y} = 0.0075x - 3.10 \quad (R^2 \approx 0.99)$$

Linear Regression: Malawi Maize Yield Example

```r
library(caret)

# Malawi districts: agricultural data
data <- data.frame(
  rainfall   = c(850,780,920,810,870,760,900),
  fertilizer = c(50,40,60,45,55,38,58),
  yield      = c(3.2,2.8,3.8,3.0,3.5,2.6,3.7)
)

# Split: 70% train, 30% test
set.seed(42)
idx   <- createDataPartition(
           data$yield, p=0.7, list=FALSE)
train <- data[idx, ]
test  <- data[-idx, ]

# Define 5-fold cross-validation control
ctrl <- trainControl(method="cv", number=5)

# Train linear regression via caret
lr_model <- train(
  yield ~ rainfall + fertilizer,
  data      = train,
  method    = "lm",
  trControl = ctrl
)
print(lr_model)       # CV results
summary(lr_model)     # Coefficients

# Predict on test set
pred <- predict(lr_model, newdata=test)
postResample(pred, test$yield)
# RMSE, Rsquared, MAE
```

Interpretation

Every extra 10mm of rainfall adds about $+0.075$ t/ha to maize yield. Every extra kg of fertilizer adds approximately $+0.04$ t/ha.

Hyperparameter Tuning

Linear regression has no classical hyperparameters. Apply regularisation variants instead:

- Ridge (`method="ridge"`): penalises large coefficients
- Lasso (`method="lasso"`): sets weak predictors to zero
- ElasticNet: combines Ridge and Lasso

Malawi Application

Agricultural officers can estimate expected district yield from rainfall forecasts and fertilizer budgets before the planting season begins.

What It Is

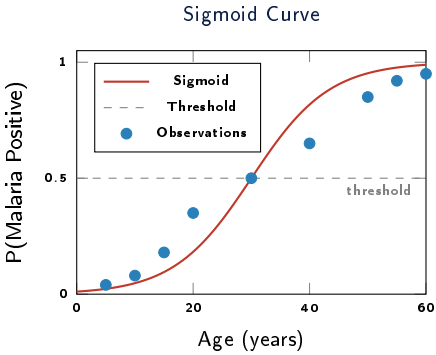
Logistic Regression is a **classification** algorithm. Despite having "regression" in the name, it predicts the **probability** that something belongs to a category. It extends linear regression by transforming the output into a value between 0 and 1 using a sigmoid (S-shaped) curve.

Data Requirements and Libraries

- **Inputs:** numeric or encoded categorical variables
- **Output:** binary label (0/1, yes/no, positive/negative)
- **Requires:** no severe class imbalance in the outcome
- **Libraries:** base R `glm()`, or `caret` with `method="glm"`

Prerequisites

Understanding of probability (values between 0 and 1) and the idea of a threshold: "if the predicted probability is above 0.5, classify as yes."



Logistic Regression: Malaria Risk Example

```r
library(caret)

health <- data.frame(
  age      = c(5,30,12,45,8,60,25,3,50,18,
               35,22,14,55,40),
  rainfall = c(850,700,900,650,880,620,720,
               870,640,800,730,810,750,610,760),
  malaria  = factor(c("Pos","Neg","Pos","Neg",
                      "Pos","Neg","Neg","Pos",
                      "Neg","Pos","Neg","Pos",
                      "Neg","Neg","Pos"))
)

# Split data
set.seed(42)
idx   <- createDataPartition(
           health$malaria, p=0.7, list=FALSE)
train <- health[idx, ]
test  <- health[-idx, ]

# Define control with ROC metric
ctrl  <- trainControl(
           method          = "cv",
           number          = 5,
           classProbs      = TRUE,
           summaryFunction = twoClassSummary)

# Train logistic regression
lg <- train(malaria ~ age + rainfall,
            data      = train,
            method    = "glm",
            family    = "binomial",
            metric    = "ROC",
            trControl = ctrl)
```

Hyperparameter Tuning

Base logistic regression has no tuning parameters. To extend it, use **regularised logistic regression**:

- `method="glmnet"`: **Ridge/Lasso penalisation**
- **Tune** `alpha` **(0=Ridge, 1=Lasso) and** `lambda` **(penalty strength) via** `tuneGrid`

Evaluation Metrics

- **Accuracy** — percentage correct
- **Sensitivity** — true positive rate
- **Specificity** — true negative rate
- **AUC-ROC** — overall discrimination ability
- **Confusion Matrix** — full breakdown of errors

Malawi Application

Health workers in high-risk districts like Chikwawa can prioritise which patients to test during peak malaria season.

What It Is

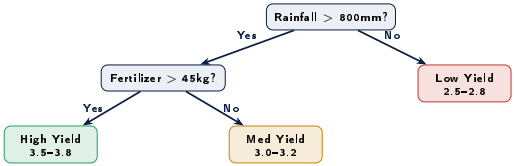
A Decision Tree makes predictions by asking a series of **yes/no questions** about the data — like a flowchart. It is the most beginner-friendly model: you can draw it on paper and explain it to anyone. It handles both regression and classification tasks and requires no data scaling.

Data Requirements and Libraries

- **Inputs:** numeric or categorical variables
- **Output:** numeric value or category label
- **Requires:** works with missing data; no scaling needed
- **Libraries:** `rpart`, or `caret` with `method="rpart"`

Prerequisites

None required. Understanding of if-then-else logic is helpful. This is an excellent starting model for beginners.



Decision tree for predicting maize yield category

Decision Tree: Code and Hyperparameter Tuning

```r
library(caret); library(rpart)

# Reuse train/test split from earlier
# Tune the complexity parameter (cp)
cpGrid <- expand.grid(cp = seq(0.001,0.05,
                               by=0.005))

dt_model <- train(
  yield ~ rainfall + fertilizer,
  data      = train,
  method    = "rpart",
  trControl = ctrl,
  tuneGrid  = cpGrid,
  metric    = "RMSE"
)

# Best cp value found by CV
print(dt_model$bestTune)

# Plot the tree
library(rpart.plot)
rpart.plot(dt_model$finalModel,
           type=2, extra=101,
           main="Maize Yield Decision Tree")

# Variable importance
varImp(dt_model)

# Evaluate on test set
pred <- predict(dt_model, test)
postResample(pred, test$yield)
```

Key Hyperparameters to Tune

- `cp` (complexity parameter) — controls tree size. Higher cp = simpler tree; lower cp = more complex. CV selects the best value automatically.
- `minsplit` — minimum rows needed to attempt a split
- `maxdepth` — maximum levels deep the tree can grow

Evaluation Metrics

Regression trees: RMSE, R^2, MAE

Classification trees: Accuracy, Kappa, Confusion Matrix, F1-Score

Overfitting Watch

A very deep tree memorises training data but fails on new data. Use cross-validation to find the right `cp` and `maxdepth`. Always compare train vs test performance.

What It Is

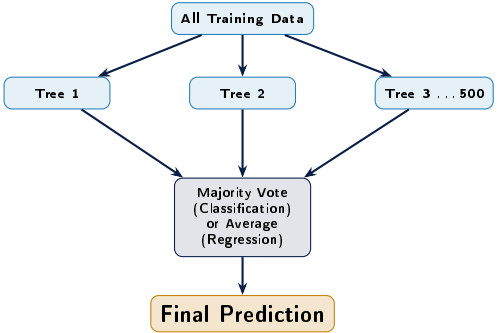
Random Forest builds **hundreds of decision trees** on random subsets of the data and combines their predictions. The idea is simple: many trees working together give a more accurate and stable result than any single tree. Developed by Leo Breiman in 2001, it is one of the most powerful algorithms in practice.

Data Requirements and Libraries

- **Inputs:** numeric or categorical variables (handles both)
- **Output:** numeric value or category label
- **Requires:** dataset of at least 100 rows for stability
- **Libraries:** `randomForest` package, or `caret` with `method="rf"`

Prerequisites

Understanding of Decision Trees first. Random Forest is simply many trees trained on random subsets of your data, voting together.



Random Forest: Food Security Classification

```r
library(caret); library(randomForest)

fs <- data.frame(
  rainfall   = c(850,600,920,500,870,450,
                 810,580,900,630,860,490),
  fertilizer = c(50,20,60,15,55,10,45,18,
                 58,22,52,12),
  status = factor(c("Secure","AtRisk",
    "Secure","AtRisk","Secure","AtRisk",
    "Secure","AtRisk","Secure","AtRisk",
    "Secure","AtRisk"))
)

set.seed(42)
idx   <- createDataPartition(
           fs$status, p=0.7, list=FALSE)
train <- fs[idx,  ]
test  <- fs[-idx, ]

# Tune: mtry = number of variables per tree
ctrl    <- trainControl(method="cv", number=5)
rfGrid  <- expand.grid(mtry = 1:2)

rf <- train(status ~ .,
            data      = train,
            method    = "rf",
            ntree     = 500,
            trControl = ctrl,
            tuneGrid  = rfGrid)

print(rf$bestTune)    # best mtry
varImpPlot(rf$finalModel) # importance

pred <- predict(rf, test)
confusionMatrix(pred, test$status)
```

Key Hyperparameters

- ntree — **number of trees (100–500)**
- mtry — **variables sampled per split. Tune via** caret. **Rule:** $\sqrt{p}$ **for classification.**
- nodesize — **minimum node size**

Evaluation Metrics

Classification: Accuracy, Kappa, Sensitivity, Specificity, F1

Regression: RMSE, R^2, MAE

Also: Variable Importance — **which features matter most.**

Malawi Application

The Ministry of Agriculture can flag at-risk districts before harvest using rainfall, fertilizer access, and soil quality data at district level.

Naive Bayes

A fast probabilistic classifier based on Bayes' Theorem. Assumes all input variables are independent of each other. Works surprisingly well for small datasets and categorical/text data.

Malawi use: Classify patient symptoms into disease categories.

Library: `e1071` or `caret` `method="nb"`

Support Vector Machine (SVM)

Finds the best boundary (hyperplane) separating two groups, maximising the margin between classes. Works well with small, clean numeric datasets. Must scale inputs first.

Malawi use: Classify satellite land-cover imagery.

Library: `e1071` or `caret` `method="svmRadial"`

XGBoost

Gradient Boosting builds trees *sequentially*, each one correcting the errors of the previous. Currently the most accurate model on tabular data.

Malawi use: Predict crop market prices using weather and economic variables.

Library: `xgboost` or `caret` `method="xgbTree"`

```r
# Quick example: all three via caret
library(caret)

# Naive Bayes
train(status~., data=train,
      method="nb", trControl=ctrl)

# SVM
train(status~., data=train,
      method="svmRadial",
      trControl=ctrl,
      preProcess=c("center","scale"))

# XGBoost
train(status~., data=train,
      method="xgbTree",
      trControl=ctrl)
```

Unsupervised Learning: Three Main Families

Key Idea

In unsupervised learning you do **not have a target or correct answer**. You give the model only the input data and ask it to find natural structures, groupings, or rules on its own. Very useful when exploring new datasets.

1. Clustering

Divides data into **groups** of similar observations.

Methods:

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN

Malawi example:

Group the 28 districts by agricultural performance indicators to target government support.

2. Dimension Reduction

Reduces many variables into a **smaller set** while keeping most of the information.

Methods:

- PCA (Principal Component Analysis)
- t-SNE
- UMAP

Malawi example:

Compress 50 survey variables into 3 main factors describing household food security.

3. Association Rule Mining

Finds **rules** describing which items tend to appear together in data.

Methods:

- Apriori Algorithm
- FP-Growth

Malawi example:

Discover which crops farmers grow together, or which health interventions are commonly used in combination.

```r
library(factoextra)

districts <- data.frame(
  name       = c("Lilongwe","Blantyre","Mzuzu",
                 "Zomba","Kasungu","Dedza",
                 "Ntchisi","Salima","Mangochi"),
  rainfall   = c(850,780,920,810,870,760,900,820,740),
  fertilizer = c(50,40,60,45,55,38,58,47,35),
  yield      = c(3.2,2.8,3.8,3.0,3.5,2.6,3.7,3.1,2.5)
)
nums <- scale(districts[, c("rainfall",
                            "fertilizer","yield")])
rownames(nums) <- districts$name

## --- K-Means Clustering ---
set.seed(42)
km <- kmeans(nums, centers=3, nstart=25)
districts$km_cluster <- km$cluster
fviz_cluster(km, data=nums,
             main="K-Means: Malawi Districts")

## --- Hierarchical Clustering ---
d  <- dist(nums, method="euclidean")
hc <- hclust(d, method="ward.D2")
plot(hc, main="Dendrogram",
     xlab="District", sub="")
districts$hc_cluster <- cutree(hc, k=3)

# See which district is in which group
districts[, c("name","km_cluster","hc_cluster")]
```

K-Means vs Hierarchical

- **K-Means:** you choose K beforehand; fast and works well on large data
- **Hierarchical:** no need to choose K first; produces a tree (dendrogram) you cut at any level; slower on large data

Hyperparameters

K-Means: **centers** (number of clusters K); use the **Elbow Method** to find the best K.

Hierarchical: **method** (ward.D2, complete, average); **k** when cutting the dendrogram.

Important: Always Scale First

Apply `scale()` before clustering. Variables measured in different units (mm vs kg) will bias the distance calculation otherwise.

Dimension Reduction: Principal Component Analysis (PCA)

```r
library(factoextra)

# Malawi survey: 6 food security indicators
survey <- data.frame(
  calorie_intake   = c(1850,1620,2100,1700,
                       2050,1580,1950,1800,1650),
  diet_diversity   = c(5,3,7,4,6,3,6,5,4),
  food_expenditure = c(45,32,60,38,55,30,52,44,35),
  stunting_rate    = c(35,48,20,42,25,50,28,38,45),
  wasting_rate     = c(8,15,4,12,5,17,6,10,14),
  market_access    = c(80,55,90,65,85,50,88,72,58)
)

# Scale before PCA (always required)
pca_result <- prcomp(survey, scale.=TRUE)
summary(pca_result)  # variance per PC

# Biplot: districts + variable directions
fviz_pca_biplot(pca_result, repel=TRUE,
  title="PCA: Malawi Food Security",
  col.var="steelblue", col.ind="darkred")

# Scree plot: how many PCs to keep?
fviz_eig(pca_result, addlabels=TRUE,
  main="Variance Explained by PCs")
```

What PCA Does

PCA combines many correlated variables into fewer Principal Components (PCs) that still capture most of the information. PC1 explains the most variance, PC2 the second most, and so on.

When to Use PCA

- You have 10+ correlated variables
- Before clustering to remove noise
- Before regression to fix collinearity
- To visualise high-dimensional data in 2D

Malawi Application

A nutrition survey with 50 questions can be reduced to 3–4 PCs representing diet quality, economic access, and child health — making analysis far more manageable and interpretable.

```r
library(arules)
library(arulesViz)

# Malawi farm household data
# Each row = one household's crop portfolio
crops <- list(
  c("Maize","Soybean","Groundnut"),
  c("Maize","Tobacco","Cassava"),
  c("Maize","Soybean","Cassava"),
  c("Maize","Groundnut","Cassava","Rice"),
  c("Soybean","Groundnut","Sunflower"),
  c("Maize","Soybean","Tobacco"),
  c("Maize","Cassava","Rice"),
  c("Groundnut","Soybean","Cassava"),
  c("Maize","Soybean","Groundnut","Rice"),
  c("Maize","Tobacco","Groundnut")
)

# Convert to transaction format
trans <- as(crops, "transactions")
summary(trans)

# Run Apriori algorithm
rules <- apriori(trans,
  parameter = list(
    supp   = 0.30,   # min 30% frequency
    conf   = 0.70,   # min 70% confidence
    minlen = 2))     # at least 2 items

# Inspect top rules sorted by lift
inspect(sort(rules, by="lift")[1:5])

# Visualise the rules
plot(rules, method="graph",
     main="Crop Association Rules")
```

Key Concepts

Association rules take the form: **If {Maize, Soybean} then {Groundnut}**

- **Support** — how often this combination appears in the data
- **Confidence** — given the left side, how often the right side also appears
- **Lift** — how much more likely the right side is given the left, compared to chance (Lift > 1 = useful rule)

Hyperparameters to Set

- **supp** (support threshold): filters out rare combinations
- **conf** (confidence threshold): filters weak rules
- Start with `supp=0.1`, `conf=0.5` and adjust

Regression Metrics

- **RMSE** (Root Mean Squared Error) — penalises large errors more. Lower is better.
- **MAE** (Mean Absolute Error) — average size of errors. Easier to interpret.
- R^2 — proportion of variance explained. Closer to 1 is better.

```
# After predictions on test set:
postResample(pred, test$yield)
# Returns RMSE, Rsquared, MAE
```

```
# Full classification evaluation
confusionMatrix(pred, test$status)
# Gives: Accuracy, Kappa, Sensitivity,
# Specificity, F1, and more

# Cross-validation summary
print(rf$results)    # all CV folds
print(rf$resample)   # per-fold results
```

The Golden Rule

Always use your **test set only once** — at the very end. Never touch it during model development. Use cross-validation on the training set to tune hyperparameters.

Classification Metrics

- **Accuracy** — % of correct predictions
- **Precision** — of all predicted positives, how many are real?
- **Recall (Sensitivity)** — of all real positives, how many did we catch?

Model Comparison Workflow

1. Fit multiple models (LR, Tree, RF...)
2. Compare cross-validation metrics
3. Pick the best model
4. Evaluate *once* on the test set
5. Report that final performance

How to Choose the Right Algorithm

1. What is my output?

A **number** (yield, income): use **Regression**
A **label** (yes/no, disease): use **Classification**
No labels at all: use **Unsupervised**

2. How much data do I have?

Under 500 rows: Naive Bayes, Logistic Regression, SVM
500–5,000 rows: Decision Tree, SVM, Random Forest
5,000+ rows: Random Forest, XGBoost

3. Must the model be explainable?

Yes (report/policy brief/government):
Use Linear Regression or Decision Tree

4. What type of data do I have?

Mostly numbers: most models work
Mostly categories or text: Naive Bayes
Mixed types: Random Forest or Decision Tree

Algorithm	Task	Start Here?
Linear Reg.	Regression	**Yes, always**
Logistic Reg.	Classification	**Yes, always**
Decision Tree	Both	Good baseline
Random Forest	Both	**Often best**
K-Means	Clustering	**Yes, explore**
PCA	Dim Reduction	When 10+ vars
Apriori	Assoc. Rules	Transaction data

Next Steps and Learning Resources

Free Online Books

- **R for Data Science** — Hadley Wickham (r4ds.had.co.nz)
- **Tidy Modeling with R** — Max Kuhn (tmwr.org)
- **Introduction to Statistical Learning (ISLR)** — James et al.
- **The caret Package Manual** — topepo.github.io/caret

Practice

- Kaggle.com — free datasets and ML competitions
- TidyTuesday — weekly real-world R datasets
- UCI Machine Learning Repository — classic benchmarks

Join R Users Malawi

- LinkedIn: **R Users Malawi**
- We need co-organizers — get in touch!
- Suggest a topic for the next event

Contact Us

+265 993 627 376
+265 880 697 619
+265 889 481 106

r-consortium.org/all-projects/rugsprogram

Supported by the R Consortium RUGS Programme

Thank You!

Questions & Discussion

"In God we trust; all others must bring data."
— W. Edwards Deming

George Sichinga — Facilitator, R Users Malawi
r-consortium.org • cran.r-project.org