Introduction to Predictive Analytics in R

Basetable Structure and Logistic Regression

Prof. Asc.Endri Raco, Ph.D.

Department of Mathematical Engineering
Polytechnic University of Tirana

November 2025

Section 1

Introduction

Welcome to Predictive Analytics in R

What You Will Learn:

- Building analytical basetables
- Logistic regression fundamentals
- Making predictions in R
- Model evaluation and interpretation
- Real-world applications in fundraising

Prerequisites: Basic R programming and data manipulation

Definition: Using historical data to predict future outcomes

Key Components:

1. **Historical Data**: Past observations with known outcomes
2. **Features**: Variables that might influence the outcome
3. **Target**: The outcome we want to predict
4. **Model**: Mathematical relationship between features and target

Goal: Learn patterns from the past to make better decisions about the future

Traditional Approach:

- Contact **ALL** donors - High mailing costs - Low response rate - Inefficient resource use - Donor fatigue

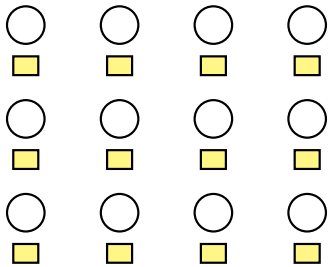
Result: Wasted resources

Predictive Approach:

- Contact donors **most likely** to donate - Lower costs - Higher response rate - Better ROI - Better donor experience

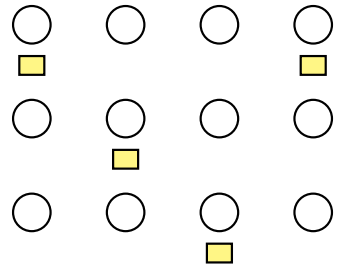
Result: Smart targeting

Address All Donors



12 mailings

Target Likely Donors



4 mailings, same response

Section 2

The Analytical Basetable

The Foundation of Predictive Modeling

Definition: A structured dataset where:

- Each **row** = one observation (e.g., one donor)
- Each **column** = one variable (feature or target)
- One column = the **target** variable (what we predict)
- Other columns = **candidate predictors** (features)

Key Principle

All information must be available **before** the prediction is made!

Candidate Predictors				**Target**
Age	Gender	Previous gifts	$\cdots$	Donate
45	F	3	$\cdots$	1
62	M	0	$\cdots$	0
38	F	5	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

Target Variable: - Binary: 0 (did not donate) or 1 (donated) - This is called a **classification problem**

```r
# Load necessary libraries
library(tidyverse)

# Read the data
basetable <- read_csv("basetable.csv")

# Examine the structure
glimpse(basetable)
```

Output preview:

```
Rows: 10,000
Columns: 8
$ age                <dbl> 45, 62, 38, ...
$ gender             <chr> "F", "M", "F", ...
$ previous_gifts     <dbl> 3, 0, 5, ...
$ Target             <dbl> 1, 0, 1, ...
```

```r
# Get population size
population_size <- nrow(basetable)

# Count targets (donations)
targets <- sum(basetable$Target)

# Calculate response rate
response_rate <- targets / population_size * 100
```

```r
# Display results
cat("Population:", population_size, "\n")
cat("Donors:", targets, "\n")
cat("Response rate:",
    round(response_rate, 2), "%\n")
```

Example Output:

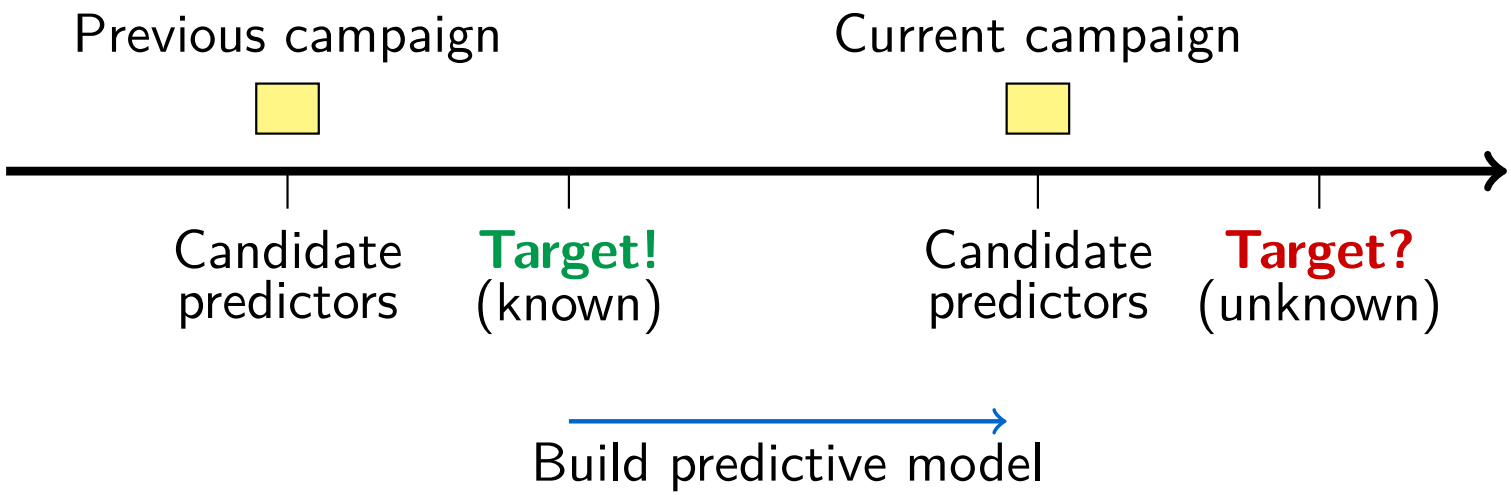
```
Population: 10000
Donors: 523
Response rate: 5.23 %
```

Key Questions:

1. How many observations do we have?
2. What is the response rate?
3. What features are available?
4. Are there missing values?
5. What are the data types?

```r
# Summary statistics
summary(basetable)

# Check for missing values
colSums(is.na(basetable))
```

```r
# Response rate (alternative calculation)
mean(basetable$Target)

# View first few rows
head(basetable)

# Check data types
str(basetable)
```


Previous campaign
Current campaign
Candidate predictors
Target! (known)
Candidate predictors
Target? (unknown)
Build predictive model

The Golden Rule of Predictive Modeling

Never use future information to predict the past!

Example - Fundraising:

- ✓ Use: Age, past donations, time since last gift
- × Don't use: Future donation amounts, next year's data

Why? When making real predictions, you won't have that information!

Section 3

Logistic Regression

The Workhorse of Binary Classification

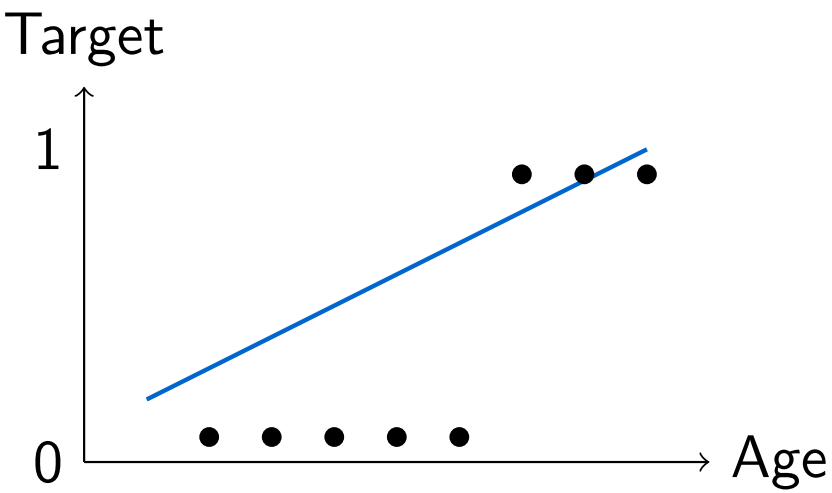
What is Logistic Regression?

- Predicts **probability** of binary outcome (0 or 1)
- Models relationship between predictors and outcome
- Output: probability between 0 and 1

Why Logistic (not Linear) Regression?

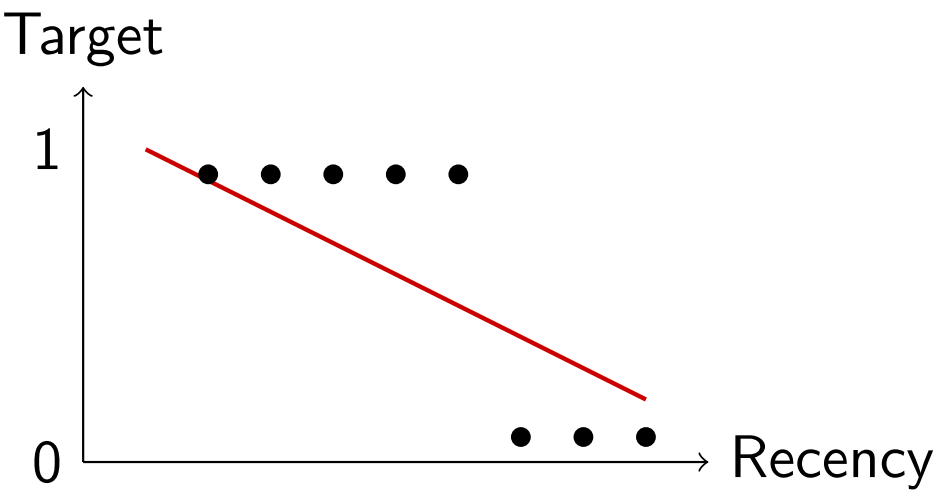
- Linear regression can predict values outside [0,1]
- Logistic regression guarantees probabilities
- Uses special transformation (logit function)

Age vs. Donation



Older people more likely to donate
Coefficient: positive

Recency vs. Donation



Recent donors more likely to donate again
Coefficient: negative

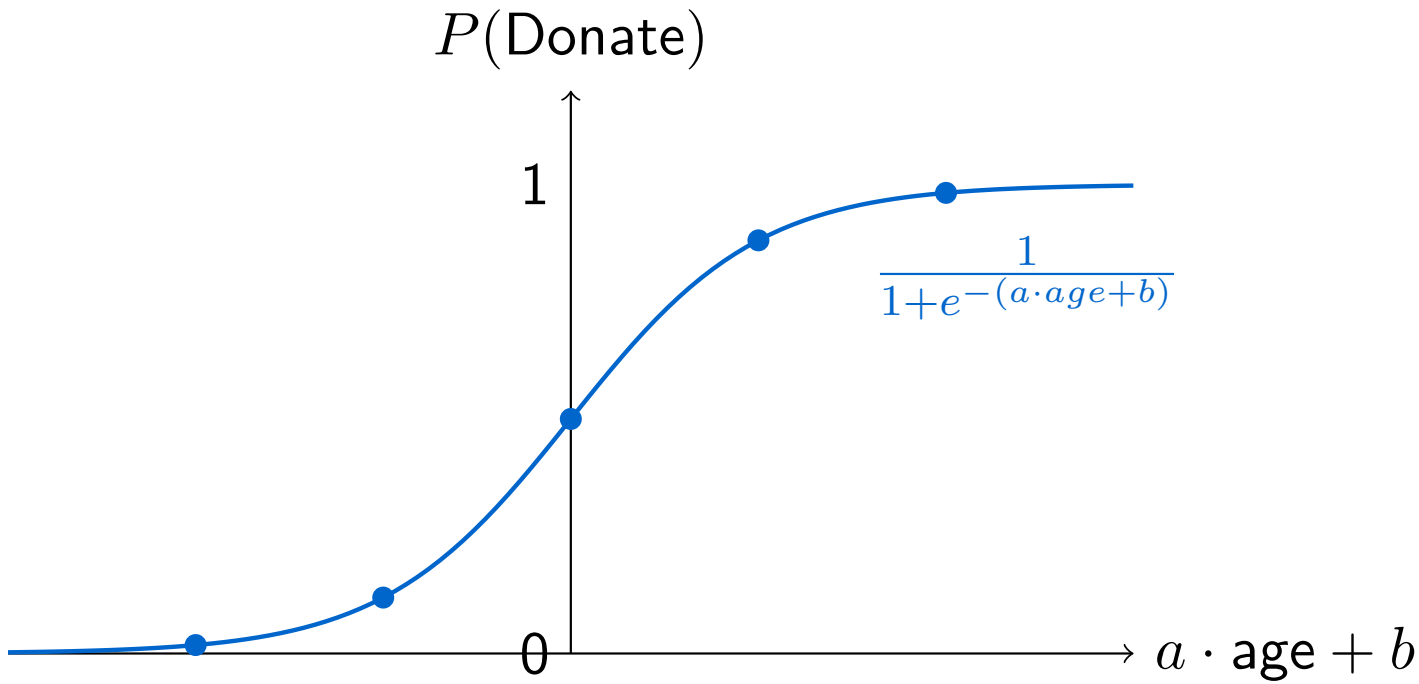
Problem: Linear combination $a \cdot x + b$ can be any real number

Solution: Apply logit function to map to [0,1]

$$P(\text{Donate} = 1) = \frac{1}{1 + e^{-(a \cdot x + b)}}$$

Properties:

- Input: any real number
- Output: probability between 0 and 1
- S-shaped (sigmoid) curve
- Smooth, differentiable


$P(\text{Donate})$
1
0
$\frac{1}{1+e^{-(a\cdot age+b)}}$
$a \cdot \text{age} + b$

```r
# Univariate logistic regression
# Predicting donation based on age

# Fit the model
logreg <- glm(Target ~ age,
              data = basetable,
              family = binomial(link = "logit"))
```

Key Components:

- `glm()`: Generalized Linear Model function
- `family = binomial`: For binary outcomes
- `link = "logit"`: Logistic transformation

```r
# View the model summary
summary(logreg)
```

Output (abbreviated):

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -4.32991    0.45123  -9.595  < 2e-16 ***
age          0.02449    0.00821   2.984  0.00284 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*'
```

```r
# Extract coefficients
coef(logreg)
```

Output:

```
(Intercept)         age
  -4.329913    0.024492
```

Interpretation:

- Intercept (b) = -4.33
- Age coefficient (a) = 0.024 (positive!)

```r
# Extract coefficients
coefs <- coef(logreg)
intercept <- coefs[1]  # b
age_coef <- coefs[2]   # a
```

```r
# Display with interpretation
cat("Intercept (b):", round(intercept, 3), "\n")
cat("Age coefficient (a):",
    round(age_coef, 4), "\n")
```

Interpretation:

- **Intercept** ($b = -4.33$): Baseline log-odds when age = 0
- **Age coefficient** ($a = 0.024$):
  - Positive $\rightarrow$ older people more likely to donate
  - Each additional year increases log-odds by 0.024

Example: Female donor, age 72, 120 days since last gift

Step 1: Calculate linear combination

$$z = 0.545 \times 1 + 0.021 \times 72 - 0.001 \times 120 - 3.39$$

$$z = 0.545 + 1.512 - 0.120 - 3.39 = -1.453$$

Step 2: Apply logistic function

$$P(\text{Donate}) = \frac{1}{1 + e^{-(-1.453)}}$$

$$= \frac{1}{1 + e^{1.453}} = 0.19$$

Result: 19% probability of donation

```r
# Create new data for prediction
new_donor <- data.frame(
  gender_F = 1,
  age = 72,
  time_since_last_gift = 120
)
```

```r
# Make prediction
# type = "response" gives probability
prediction <- predict(logreg,
                      newdata = new_donor,
                      type = "response")

cat("Predicted probability:",
    round(prediction, 3))
```

Output:

```
Predicted probability: 0.190
```

```r
# Alternative: get log-odds first
log_odds <- predict(logreg,
                    newdata = new_donor)

# Then convert to probability
prob <- 1 / (1 + exp(-log_odds))
cat("Probability:", round(prob, 3))
```

Univariate:

$$P(Y = 1) = \frac{1}{1 + e^{-(a \cdot x + b)}}$$

Multivariate:

$$P(Y = 1) = \frac{1}{1 + e^{-(a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + b)}}$$

```r
# Multiple predictors
logreg_multi <- glm(
  Target ~ age + max_gift + income_low,
  data = basetable,
  family = binomial(link = "logit")
)
```

```r
# View results
summary(logreg_multi)

# Extract coefficients
coef(logreg_multi)
```

Output:

```
 (Intercept)          age     max_gift   income_low
   -8.806435     0.024331     0.039061    -0.767938
```

```r
# Extract and display coefficients
coefs <- coef(logreg_multi)
print(round(coefs, 4))
```

Variable	Coefficient	Interpretation
age	+0.0243	Older → more likely
max_gift	+0.0391	Larger past gifts → more likely
income_low	-0.7679	Low income → less likely

Holding Other Variables Constant

Each coefficient represents the effect of that variable **while keeping all other variables fixed**

Example:

- Age coefficient (+0.024) means: for two donors with the *same* max_gift and income, the older donor has higher probability
- This is different from univariate regression!

```r
# Prepare current campaign data
current_data <- read_csv("current_campaign.csv")

# Select predictor variables
# (must match training data!)
predictors <- c("age", "max_gift",
                "income_low")

new_data <- current_data[, predictors]
```

```r
# Make predictions for all donors
predictions <- predict(logreg_multi,
                       newdata = new_data,
                       type = "response")

# Add predictions to dataset
current_data$predicted_prob <- predictions
```

```r
# View top predicted donors
current_data %>%
  arrange(desc(predicted_prob)) %>%
  head(10)
```

Output preview:

```
   age max_gift income_low predicted_prob
1   68      250          0          0.842
2   72      180          0          0.789
3   55      300          0          0.761
...
```

```r
# Set threshold (e.g., top 30%)
threshold <- quantile(predictions,
                      probs = 0.70)

cat("Threshold probability:",
    round(threshold, 3), "\n")
```

Output:

```
Threshold probability: 0.156
```

```r
# Identify high-probability donors
current_data <- current_data %>%
  mutate(should_contact =
           predicted_prob > threshold)

# How many donors to contact?
n_contact <- sum(current_data$should_contact)
cat("Contact:", n_contact, "donors\n")
```

```r
# Expected number of responses
expected <- sum(
  current_data$predicted_prob[
    current_data$should_contact
  ]
)

cat("Expected responses:",
    round(expected), "\n")

# Expected response rate
exp_rate <- expected / n_contact * 100
cat("Expected rate:",
    round(exp_rate, 1), "%\n")
```

```r
library(ggplot2)

# Create histogram
ggplot(current_data,
       aes(x = predicted_prob)) +
  geom_histogram(bins = 30,
                 fill = "steelblue",
                 color = "white")
```

```r
# Add threshold line and labels
ggplot(current_data,
       aes(x = predicted_prob)) +
  geom_histogram(bins = 30,
                 fill = "steelblue",
                 color = "white") +
  geom_vline(xintercept = threshold,
             color = "red",
             linetype = "dashed", size = 1)
```

```r
# Final plot with all elements
ggplot(current_data,
       aes(x = predicted_prob)) +
  geom_histogram(bins = 30,
                 fill = "steelblue",
                 color = "white") +
  geom_vline(xintercept = threshold,
             color = "red",
             linetype = "dashed", size = 1) +
  labs(title = "Predicted Probabilities",
       x = "Probability of Donation",
       y = "Number of Donors") +
  theme_minimal()
```

General Form:

```
glm(response ~ predictor1 + predictor2 + ...,
    data = dataset,
    family = binomial(link = "logit"))
```

Special Operators:

- `+` : Include variable
- `-` : Exclude variable
- `:` : Interaction term
- `*` : Main effects and interaction
- `.` : All variables except response

```r
# Use all variables
glm(Target ~ .,
    data = basetable,
    family = binomial)

# Interaction between age and income
glm(Target ~ age * income,
    data = basetable,
    family = binomial)

# Polynomial term
glm(Target ~ age + I(age^2),
    data = basetable,
    family = binomial)
```

```r
# 1. Load and explore data
basetable <- read_csv("donor_data.csv")
glimpse(basetable)
summary(basetable)
```

```r
# 2. Fit logistic regression
model <- glm(
  donated ~ age + income + previous_gifts,
  data = basetable,
  family = binomial
)

# 3. Check model summary
summary(model)
```

```r
# 4. Make predictions on new data
new_donors <- read_csv("new_donors.csv")

predictions <- predict(model,
                       newdata = new_donors,
                       type = "response")
```

```r
# 5. Select top 30% to contact
cutoff <- quantile(predictions, 0.70)

new_donors$contact <- predictions > cutoff

# 6. Export results
write_csv(new_donors, "targeting_list.csv")
```

1. **Basetable** is the foundation - structured data with features and target
2. **Logistic regression** predicts probabilities using:

$$P(Y = 1) = \frac{1}{1 + e^{-(a_1 x_1 + \cdots + a_n x_n + b)}}$$

3. **Coefficients** show direction and strength of relationships
4. Use `glm()` with `family = binomial` in R
5. `predict()` with `type = "response"` gives probabilities
6. Target high-probability individuals for better ROI

Data Preparation

- Clean missing values - Check for outliers - Verify timing (no future leakage!) - Scale variables if needed

Modeling

- Start simple (fewer predictors) - Check coefficient signs (do they make sense?) - Examine model diagnostics - Validate on held-out data

1. **Using future information**
   Only use data available at prediction time

2. **Ignoring imbalanced data**
   If 95% are 0's, adjust or use specialized techniques

3. **Overfitting**
   Too many predictors relative to sample size

4. **Not validating**
   Always test on new data, not training data

5. **Treating probabilities as certainties**
   A 0.8 probability means 20% chance of being wrong!

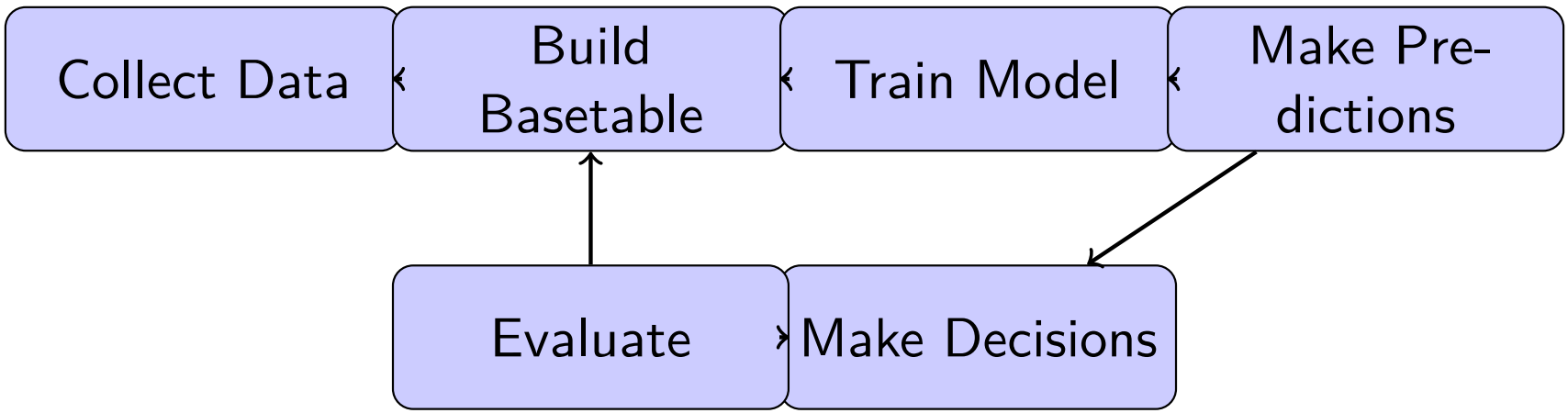
6. **Ignoring model assumptions**
   Check for multicollinearity, influential points

In Future Lectures:

1. Model evaluation metrics (accuracy, precision, recall, AUC)
2. Feature engineering and selection
3. Handling missing data
4. Cross-validation techniques
5. Advanced algorithms (Random Forests, Gradient Boosting)
6. Model interpretation and explainability

Practice Exercise:

Build a logistic regression model to predict customer churn using the provided dataset


Collect Data
Build Basetable
Train Model
Make Predictions
Evaluate
Make Decisions

Questions?

Prof. Asc. Endri Raco, Ph.D.
e.raco@fimif.edu.al

Polytechnic University of Tirana
Department of Mathematical Engineering

Next: 10 minutes break and classwork 1

Section 4

Variable Selection

Choosing the Right Predictors

The Challenge:

You have many potential predictors: - age, gender, income, location - past donation amounts (min, max, mean, median) - donation counts, recency, frequency - demographic variables

The Question: Which ones should you include in your model?

Donor Database Variables:

- `age`
- `max_gift`, `min_gift`, `mean_gift`, `median_gift`
- `income_low`, `income_medium`, `income_high`
- `country_USA`, `country_India`, `country_UK`
- `number_gift_min50`, `number_gift_min100`, `number_gift_min150`
- `gender_F`, `gender_M`
- `recency`, `frequency`, `monetary`

Total: 15+ candidate predictors!

Problem 1: Overfitting

- Model learns noise in training data
- Poor performance on new data
- Memorization instead of generalization

Problem 2: Hard to Maintain

- Collecting data for 15+ variables is expensive
- Missing data becomes more likely
- Model updates are complex

Problem 3: Hard to Interpret

- Which variables really matter?
- Multicollinearity confuses interpretation
- Business stakeholders get confused

Problem 4: Computational Cost

- Slower predictions
- More storage required
- Higher maintenance burden

Objective

Find the **smallest set** of predictors that gives the **best performance**

Balance:

- Model Performance (AUC, accuracy)
- Model Simplicity (fewer variables)
- Model Interpretability (understandable)

Section 5

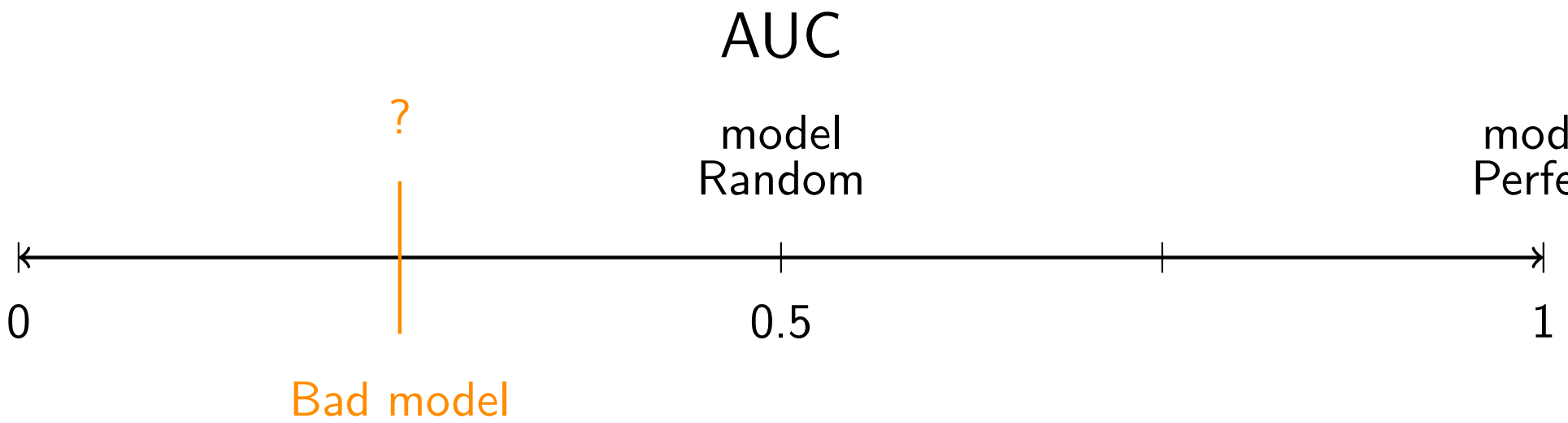
Model Evaluation: AUC

Area Under the ROC Curve

AUC = Single number that summarizes model quality

Range: 0 to 1

- **0.5** = Random guessing (coin flip)
- **1.0** = Perfect predictions
- $>$ **0.7** = Generally acceptable
- $>$ **0.8** = Good model
- $>$ **0.9** = Excellent model


AUC
?
model
Random
mod
Perfe
0
0.5
1
Bad model

```r
# Load required library
library(pROC)

# Fit a simple model
model <- glm(donated ~ age + balance,
             data = basetable,
             family = binomial)
```

```r
# Get predicted probabilities
predictions <- predict(model,
                       type = "response")

# Calculate AUC
roc_obj <- roc(basetable$donated, predictions)
auc_value <- auc(roc_obj)

cat("AUC:", round(auc_value, 3))
```

Output:

```
AUC: 0.687
```

```r
# Custom function to calculate AUC
calculate_auc <- function(variables,
                          target,
                          data) {

  # Build formula
  formula <- as.formula(
    paste(target, "~",
          paste(variables, collapse = " + "))
  )

  # Continue on next slide...
}
```

```r
calculate_auc <- function(variables,
                          target,
                          data) {
  formula <- as.formula(
    paste(target, "~",
          paste(variables, collapse = " + "))
  )

  # Fit model
  model <- glm(formula,
               data = data,
               family = binomial)

  # Continue on next slide...
}
```

```r
calculate_auc <- function(variables,
                          target,
                          data) {
  formula <- as.formula(
    paste(target, "~",
          paste(variables, collapse = " + "))
  )

  model <- glm(formula, data = data,
               family = binomial)

  # Get predictions and calculate AUC
  preds <- predict(model, type = "response")
  auc_val <- auc(roc(data[[target]], preds))

  return(as.numeric(auc_val))
}
```

```r
# Test with age and balance
auc1 <- calculate_auc(
  variables = c("age", "balance"),
  target = "donated",
  data = basetable
)

cat("AUC:", round(auc1, 3))
```

Output:

```
AUC: 0.687
```

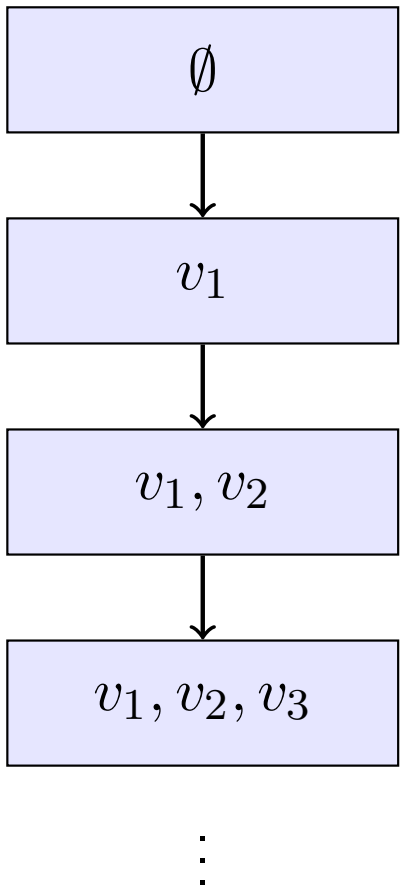
Section 6

Forward Stepwise Selection

Greedy Algorithm for Variable Selection

Idea: Start with nothing, add variables one at a time

Process: 1. Start with empty model (no predictors) 2. Try adding each candidate variable 3. Keep the one that improves AUC the most 4. Repeat until stopping criterion



Find best single variable

Add best 2nd variable

Add best 3rd variable

Algorithm Steps:

Iteration 1: - Try each variable alone - Pick the one with highest AUC

Iteration 2: - Try adding each remaining variable - Pick the combination with highest AUC

Iteration k: - Try adding each remaining variable to current set - Pick the best combination

Candidate Variables: age, balance, previous_gifts

```
# Try each variable individually
auc_age <- calculate_auc("age", "donated", data)
auc_balance <- calculate_auc("balance",
                             "donated", data)
auc_previous <- calculate_auc("previous_gifts",
                              "donated", data)

cat("Age AUC:", round(auc_age, 3), "\n")
cat("Balance AUC:", round(auc_balance, 3), "\n")
cat("Previous AUC:", round(auc_previous, 3), "\n")
```

Results:

```
Age AUC: 0.623
Balance AUC: 0.701
Previous AUC: 0.665
```

Winner: balance (AUC = 0.701)

Current Model: donated ~ balance

Current: balance

Remaining: age, previous_gifts

```
# Try adding each remaining variable
auc_bal_age <- calculate_auc(
  c("balance", "age"), "donated", data
)

auc_bal_prev <- calculate_auc(
  c("balance", "previous_gifts"),
  "donated", data
)
```

Results:

```
balance + age: 0.715
balance + previous_gifts: 0.748
```

Winner: balance + previous_gifts (AUC = 0.748)

Current Model: donated ~ balance + previous_gifts

```
find_next_best <- function(current_vars,
                           candidate_vars,
                           target,
                           data) {

  best_auc <- -1
  best_var <- NULL

  # Try each candidate variable
  for (var in candidate_vars) {
    # Continue on next slide...
  }
}
```

```r
find_next_best <- function(current_vars,
                           candidate_vars,
                           target, data) {
  best_auc <- -1
  best_var <- NULL

  for (var in candidate_vars) {
    # Test this variable
    test_vars <- c(current_vars, var)
    test_auc <- calculate_auc(test_vars,
                              target, data)

    # Continue on next slide...
  }
}
```

```r
find_next_best <- function(current_vars,
                           candidate_vars,
                           target, data) {
  best_auc <- -1
  best_var <- NULL

  for (var in candidate_vars) {
    test_vars <- c(current_vars, var)
    test_auc <- calculate_auc(test_vars,
                              target, data)

    # Is this the best so far?
    if (test_auc > best_auc) {
      best_auc <- test_auc
      best_var <- var
    }
  }

  return(best_var)
}
```

```r
# Current variables
current <- c("balance")

# Candidate variables
candidates <- c("age", "previous_gifts",
                "income_low")

# Find next best
next_var <- find_next_best(
  current_vars = current,
  candidate_vars = candidates,
  target = "donated",
  data = basetable
)

cat("Next best variable:", next_var)
```

```r
# Setup
candidate_vars <- c("age", "balance",
                    "previous_gifts", "income_low",
                    "max_gift", "recency")

current_vars <- c()
target <- "donated"
max_vars <- 5
```

```r
# Main loop
n_iterations <- min(max_vars,
                    length(candidate_vars))

for (i in 1:n_iterations) {

  # Find next best variable
  next_var <- find_next_best(
    current_vars = current_vars,
    candidate_vars = candidate_vars,
    target = target,
    data = basetable
  )

  # Continue on next slide...
}
```

```r
for (i in 1:n_iterations) {

  next_var <- find_next_best(
    current_vars = current_vars,
    candidate_vars = candidate_vars,
    target = target,
    data = basetable
  )

  # Add to current variables
  current_vars <- c(current_vars, next_var)

  # Remove from candidates
  candidate_vars <- setdiff(candidate_vars,
                            next_var)

  # Print progress
  cat("Step", i, ":", next_var, "\n")
}
```

Output:

```
Step 1 : balance
Step 2 : previous_gifts
Step 3 : recency
Step 4 : age
Step 5 : max_gift
```

Final Selected Variables:

```
print(current_vars)
```

```
[1] "balance" "previous_gifts" "recency"
[4] "age" "max_gift"
```

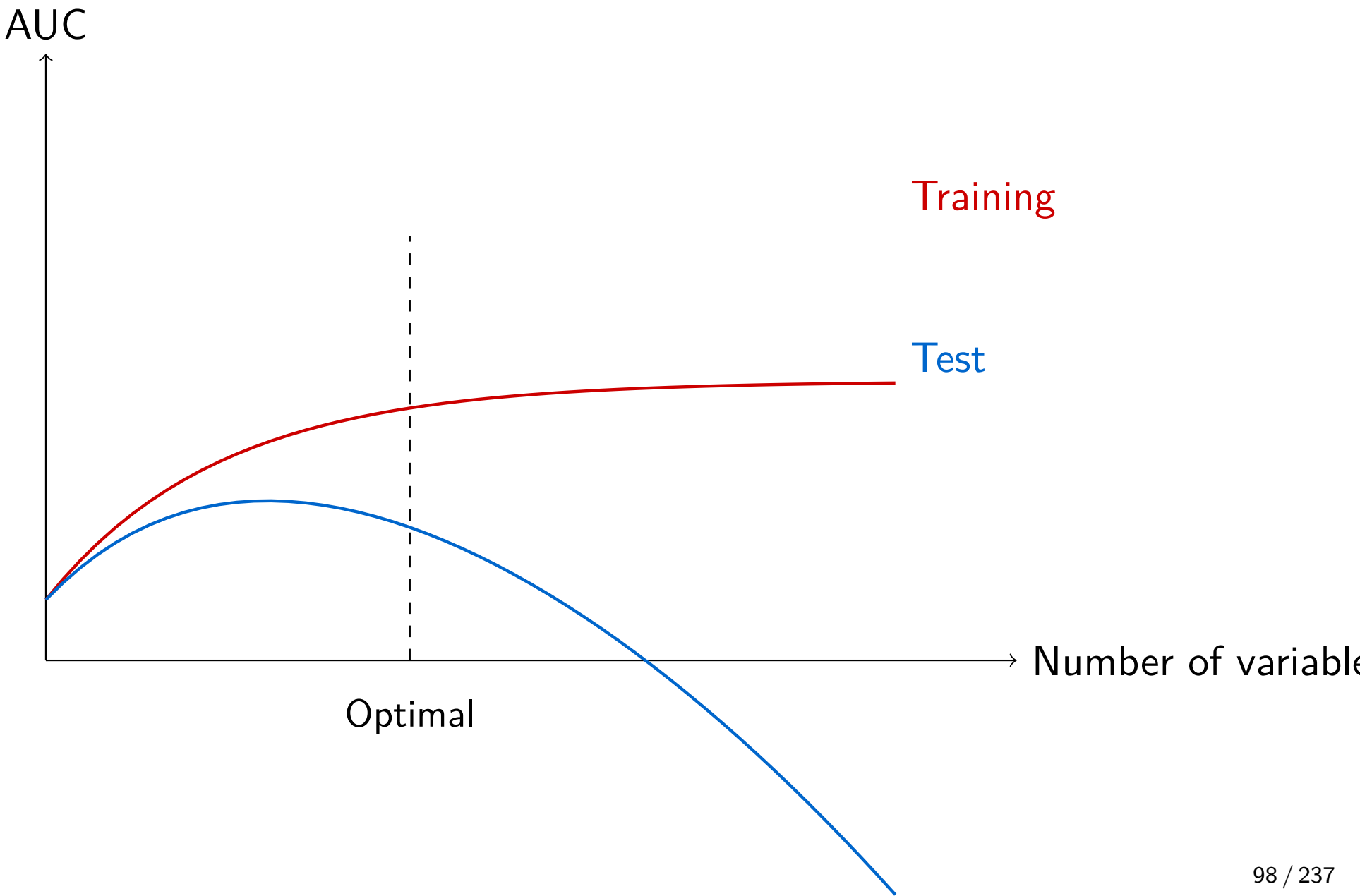
Section 7

Deciding on Number of Variables

More variables $\neq$ Better model

Training Data: - AUC always increases (or stays same) - Model learns specific patterns

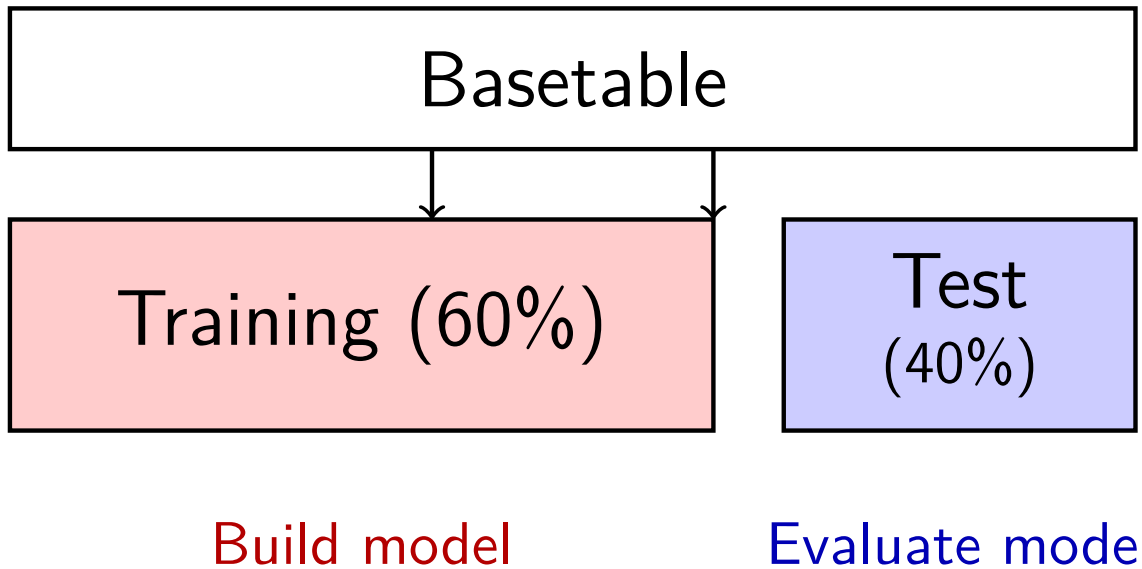
New Data: - AUC may decrease with too many variables - Model learned noise, not signal


AUC
Training
Test
Number of variable
Optimal

Key Idea

Evaluate model performance on **unseen data**

Procedure: 1. Split data into training (60%) and test (40%) 2. Build models using training data only 3. Evaluate models on test data 4. Choose model with best test performance


Basetable
Training (60%)
Test
(40%)
Build model
Evaluate model

```r
# Load library
library(caret)

# Set seed for reproducibility
set.seed(123)

# Create partition (60% training)
train_index <- createDataPartition(
  basetable$donated,
  p = 0.6,
  list = FALSE
)
```

```r
# Split the data
train_data <- basetable[train_index, ]
test_data <- basetable[-train_index, ]

# Check sizes
cat("Training size:", nrow(train_data), "\n")
cat("Test size:", nrow(test_data), "\n")
```

Output:

```
Training size: 6000
Test size: 4000
```

```r
# Check if split is balanced
train_rate <- mean(train_data$donated)
test_rate <- mean(test_data$donated)

cat("Training donation rate:",
    round(train_rate * 100, 2), "%\n")
cat("Test donation rate:",
    round(test_rate * 100, 2), "%\n")
```

Output:

```
Training donation rate: 12.3 %
Test donation rate: 12.5 %
```

Good! Rates are similar

```r
# Run forward selection on TRAINING data
candidate_vars <- c("age", "balance",
                    "previous_gifts", "income_low",
                    "max_gift", "recency")

current_vars <- c()
train_aucs <- c()
test_aucs <- c()
```

```r
# For each step in forward selection
for (i in 1:length(candidate_vars)) {

  # Find next best on TRAINING data
  next_var <- find_next_best(
    current_vars = current_vars,
    candidate_vars = candidate_vars,
    target = "donated",
    data = train_data  # Use training!
  )

  current_vars <- c(current_vars, next_var)
  candidate_vars <- setdiff(candidate_vars,
                            next_var)

  # Continue on next slide...
}
```

```r
for (i in 1:length(candidate_vars)) {
  # ... previous code ...

  current_vars <- c(current_vars, next_var)
  candidate_vars <- setdiff(candidate_vars,
                            next_var)

  # Calculate AUC on TRAINING data
  train_auc <- calculate_auc(current_vars,
                             "donated",
                             train_data)

  # Calculate AUC on TEST data
  test_auc <- calculate_auc(current_vars,
                            "donated",
                            test_data)

  train_aucs <- c(train_aucs, train_auc)
  test_aucs <- c(test_aucs, test_auc)
}
```

```r
# Create dataframe for plotting
results <- data.frame(
  num_vars = 1:length(train_aucs),
  train_auc = train_aucs,
  test_auc = test_aucs
)

# View results
print(results)
```

```r
library(ggplot2)

# Create plot
ggplot(results, aes(x = num_vars)) +
  geom_line(aes(y = train_auc,
                color = "Training"),
            size = 1.2) +
  geom_line(aes(y = test_auc,
                color = "Test"),
            size = 1.2) +
  geom_point(aes(y = train_auc,
                 color = "Training"),
             size = 3) +
  geom_point(aes(y = test_auc,
                 color = "Test"),
             size = 3)
```

```r
# Add labels and theme
ggplot(results, aes(x = num_vars)) +
  # ... geom_line and geom_point ...
  scale_color_manual(
    values = c("Training" = "red",
               "Test" = "blue")
  ) +
  labs(
    title = "Model Performance vs Complexity",
    x = "Number of Variables",
    y = "AUC",
    color = "Dataset"
  ) +
  theme_minimal() +
  theme(legend.position = "top")
```

Example Output:

```
  num_vars train_auc test_auc
1        1     0.701    0.695
2        2     0.748    0.732
3        3     0.772    0.751
4        4     0.783    0.748
5        5     0.795    0.742
6        6     0.802    0.738
```

Optimal: 3 variables (highest test AUC)

Criteria for Selection:

1. **Highest test AUC** Ensures generalization
2. **Simplicity** Fewer variables = easier to maintain
3. **Stability** Small gap between train and test AUC
4. **Practical constraints** Data availability, cost

```r
# Find index with highest test AUC
optimal_idx <- which.max(test_aucs)

cat("Optimal number of variables:",
    optimal_idx, "\n")
cat("Test AUC:",
    round(test_aucs[optimal_idx], 3), "\n")
```

```r
# Get the optimal variable set
# (Re-run forward selection to track vars)
current_vars <- c()
candidate_vars_reset <- c("age", "balance",
                          "previous_gifts",
                          "income_low",
                          "max_gift", "recency")

for (i in 1:optimal_idx) {
  next_var <- find_next_best(
    current_vars, candidate_vars_reset,
    "donated", train_data
  )
  current_vars <- c(current_vars, next_var)
  candidate_vars_reset <-
    setdiff(candidate_vars_reset, next_var)
}

cat("Selected variables:\n")
print(current_vars)
```

```r
# Build final model with optimal variables
final_model <- glm(
  donated ~ balance + previous_gifts + recency,
  data = train_data,
  family = binomial
)

# Evaluate on test set
test_preds <- predict(final_model,
                      newdata = test_data,
                      type = "response")

final_auc <- auc(roc(test_data$donated,
                     test_preds))

cat("Final model AUC:", round(final_auc, 3))
```

1. **Split data** into training and test sets
2. **Run forward selection** on training data
3. **Evaluate each model** on test data
4. **Plot** training vs test AUC
5. **Choose** model with best test AUC
6. **Build final model** with selected variables

Variable Selection

- Start simple, add complexity carefully - Use forward stepwise for systematic selection - Always validate on holdout data

Overfitting

- More variables can hurt performance - Train/test split reveals true performance - Choose model at peak of test AUC curve

DO:

- Split data before any modeling
- Evaluate on untouched test data
- Consider practical constraints
- Document your selection process

DON'T:

- Use test data for variable selection
- Overfit to training data
- Ignore the train/test gap
- Choose models based only on training AUC

Forward Stepwise (what we learned) - Start empty, add best variables

Backward Stepwise - Start with all, remove worst variables

Lasso Regularization - Automatic variable selection via penalties

Information Criteria (AIC, BIC) - Balance fit and complexity

We'll explore these in advanced courses!

Mistake 1: Using all variables without selection

- Results in overfitting
- Hard to interpret and maintain

Mistake 2: Selecting variables without train/test split

- Cannot detect overfitting
- Overconfident in model performance

Mistake 3: Choosing model with highest training AUC

- Ignores generalization
- Poor real-world performance

1. **Start simple** Begin with 2-3 most important variables
2. **Add gradually** Monitor test performance after each addition
3. **Stop early** When test AUC stops improving
4. **Consider business** Easy-to-collect variables preferred
5. **Document everything** Explain why each variable was selected

In Next Lecture:

- Cross-validation (better than single train/test)
- Advanced evaluation metrics
- Feature engineering
- Handling categorical variables
- Regularization techniques

Practice:

- Apply forward selection to bank marketing data
- Compare different numbers of variables
- Visualize train vs test performance

Questions?

Review:

- Variable selection reduces overfitting
- Forward stepwise is systematic and interpretable
- Always validate on test data
- Choose models that generalize well

Next: 10 minutes break and Classwork 2 - Variable Selection Exercise #
Chapter 3: Model Evaluation with Business Metrics

Cumulative Gains and Lift Curves

The Challenge:

- AUC is great for data scientists
- But business stakeholders ask: "How much money will we make?"
- Need evaluation metrics that speak the language of business

Today's Goal: Learn to evaluate models using business-friendly metrics

Problem with AUC:

- **Complex**: What does 0.75 actually mean?
- **Abstract**: Doesn't translate to dollars
- **Single number**: Hides important details

What Business Needs:

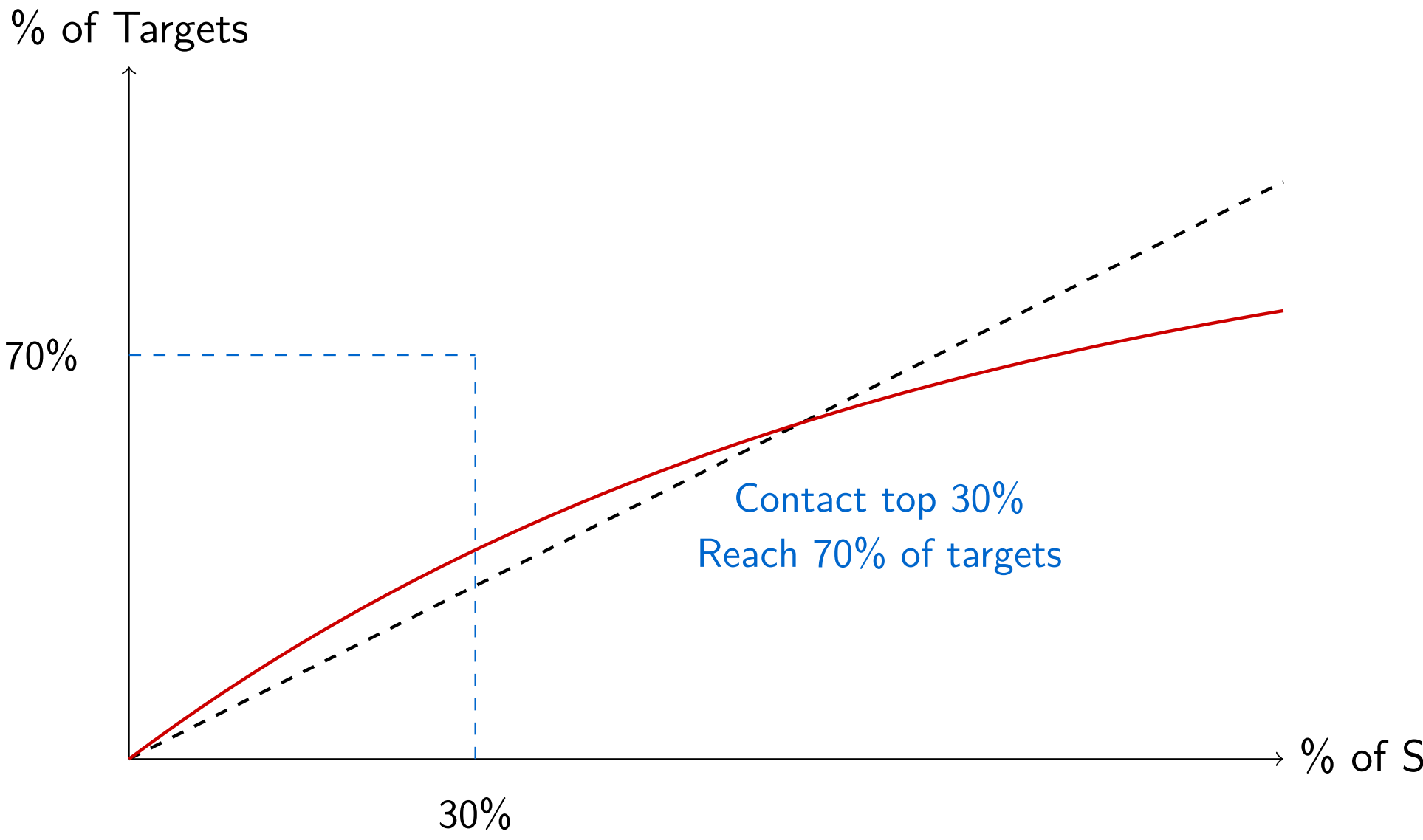
- How many customers should we contact?
- What's our expected profit?
- How much better is this than random?

A Picture Worth a Thousand Predictions

Key Question: If I contact the top X% of customers, what % of all potential donors will I reach?

Why It's Useful:

- Visual and intuitive
- Directly answers business questions
- Easy to explain to non-technical stakeholders
- Shows model performance at different thresholds


% of Targets
70%
Contact top 30%
Reach 70% of targets
% of S
30%

X-Axis: Percentage of Sample Contacted

- 0% = Contact nobody
- 50% = Contact half the population
- 100% = Contact everyone

Y-Axis: Percentage of Targets Reached

- 0% = Found no donors
- 50% = Found half of all potential donors
- 100% = Found all potential donors

Step 1: Sort by Predicted Probability

```r
# You have predictions from your model
predictions <- predict(model,
                       newdata = test_data,
                       type = "response")

# Add to dataframe
test_data$pred_prob <- predictions
test_data$actual <- test_data$donated
```

Step 2: Sort from Highest to Lowest Probability

```r
# Sort by predicted probability (descending)
sorted_data <- test_data %>%
  arrange(desc(pred_prob))

# Now the "most likely donors" are at the top
head(sorted_data[, c("pred_prob", "actual")])
```

Why? We want to contact the most promising prospects first!

Step 3: Calculate Cumulative Metrics

```
# Calculate cumulative values
sorted_data <- sorted_data %>%
  mutate(
    # What % of sample have we contacted?
    perc_sample = row_number() / n(),

    # How many targets found so far?
    cum_targets = cumsum(actual),

    # What % of all targets is that?
    perc_targets = cum_targets / sum(actual)
  )
```

Now we have all the points for our curve!

Rank	Predicted	Actual	Cum. %
1	0.95	1	10%
2	0.92	1	20%
3	0.88	1	30%
4	0.85	0	30%
5	0.82	1	40%
...	...	...	...
10	0.50	0	50%

Interpretation: By contacting top 5 people (50% of sample), we reached 40% of all donors.

```r
# Load required library
library(tidyverse)

# Function to calculate gains curve data
calculate_gains <- function(actual, predicted) {

  # Create dataframe
  df <- data.frame(
    actual = actual,
    predicted = predicted
  ) %>%
    arrange(desc(predicted))

  # Continue on next slide...
}
```

```r
calculate_gains <- function(actual, predicted) {

  df <- data.frame(
    actual = actual,
    predicted = predicted
  ) %>%
    arrange(desc(predicted))

  # Calculate cumulative metrics
  df <- df %>%
    mutate(
      perc_sample = row_number() / n(),
      cum_targets = cumsum(actual),
      perc_targets = cum_targets / sum(actual)
    )

  return(df)
}
```

```r
# Use the function
gains_data <- calculate_gains(
  actual = test_data$donated,
  predicted = test_preds
)

# Plot the curve
ggplot(gains_data, aes(x = perc_sample,
                       y = perc_targets)) +
  geom_line(color = "red", size = 1.2) +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed") +
  labs(title = "Cumulative Gains Curve",
       x = "Percentage of Sample",
       y = "Percentage of Targets") +
  theme_minimal()
```

Perfect Model:

- Steep initial rise - Reaches 100% quickly - All targets at the top

Random Model:

- Diagonal line - Proportional gains - No targeting ability

Your Model:

- Between the two - Closer to perfect = better - Steeper = more valuable

Key Insight:

The area between your curve and the diagonal shows the model's value!

At 30% of sample, we reach 70% of targets

Business Translation:

- Contact: 30,000 out of 100,000 customers
- Reach: 70 out of 100 potential donors
- Benefit: 70% results with 30% effort
- Savings: Avoid contacting 70,000 people

```r
# Calculate gains for both models
gains_model1 <- calculate_gains(actual, pred1)
gains_model2 <- calculate_gains(actual, pred2)

# Add model identifier
gains_model1$model <- "Model 1"
gains_model2$model <- "Model 2"

# Combine
gains_combined <- rbind(gains_model1,
                        gains_model2)
```

```r
ggplot(gains_combined,
       aes(x = perc_sample,
           y = perc_targets,
           color = model)) +
  geom_line(size = 1.2) +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "black") +
  scale_color_manual(
    values = c("Model 1" = "blue",
               "Model 2" = "red")
  ) +
  labs(title = "Model Comparison",
       x = "% of Sample",
       y = "% of Targets") +
  theme_minimal()
```

The model with the curve furthest from the diagonal

- Model 1 (blue) is higher $\rightarrow$ Better targeting
- Model 2 (red) is lower $\rightarrow$ Weaker targeting
- Both beat random (dashed line) $\rightarrow$ Both useful

Rule: Higher curve = Better model

Section 8

The Lift Curve

How Much Better Than Random?

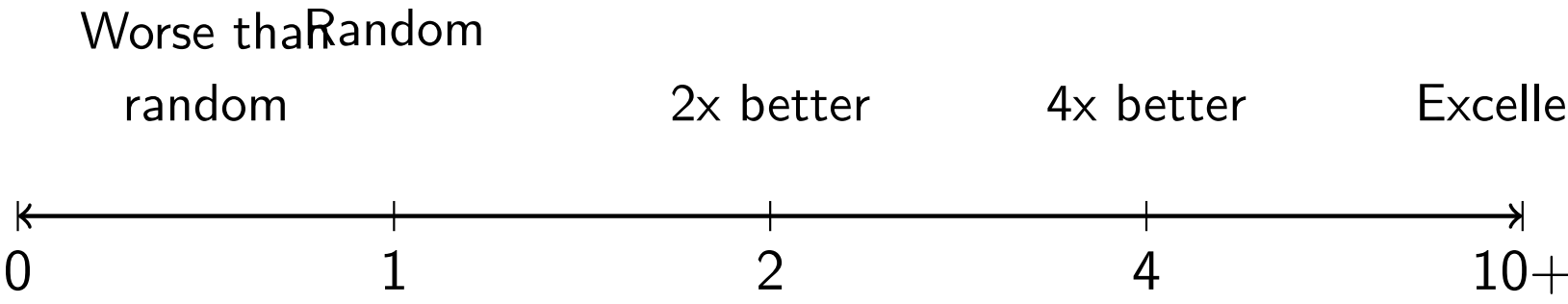
Lift = Factor of improvement over random selection

Formula:

$$\text{Lift} = \frac{\text{Percentage of targets in selection}}{\text{Overall percentage of targets}}$$

Example: If 25% of selected customers donate, but only 5% donate overall:

$$\text{Lift} = \frac{25\%}{5\%} = 5$$


Worse than random
Random
2x better
4x better
Excellent!
0
1
2
4
10+

Similar to gains curve, but:

- **Y-axis**: Lift (not cumulative percentage)
- **Shape**: Decreasing (starts high, drops)
- **Interpretation**: How much better at each point

Key Features:

- Maximum lift at the beginning
- Gradually decreases
- Converges to 1 at 100%

As you contact more people, average quality drops

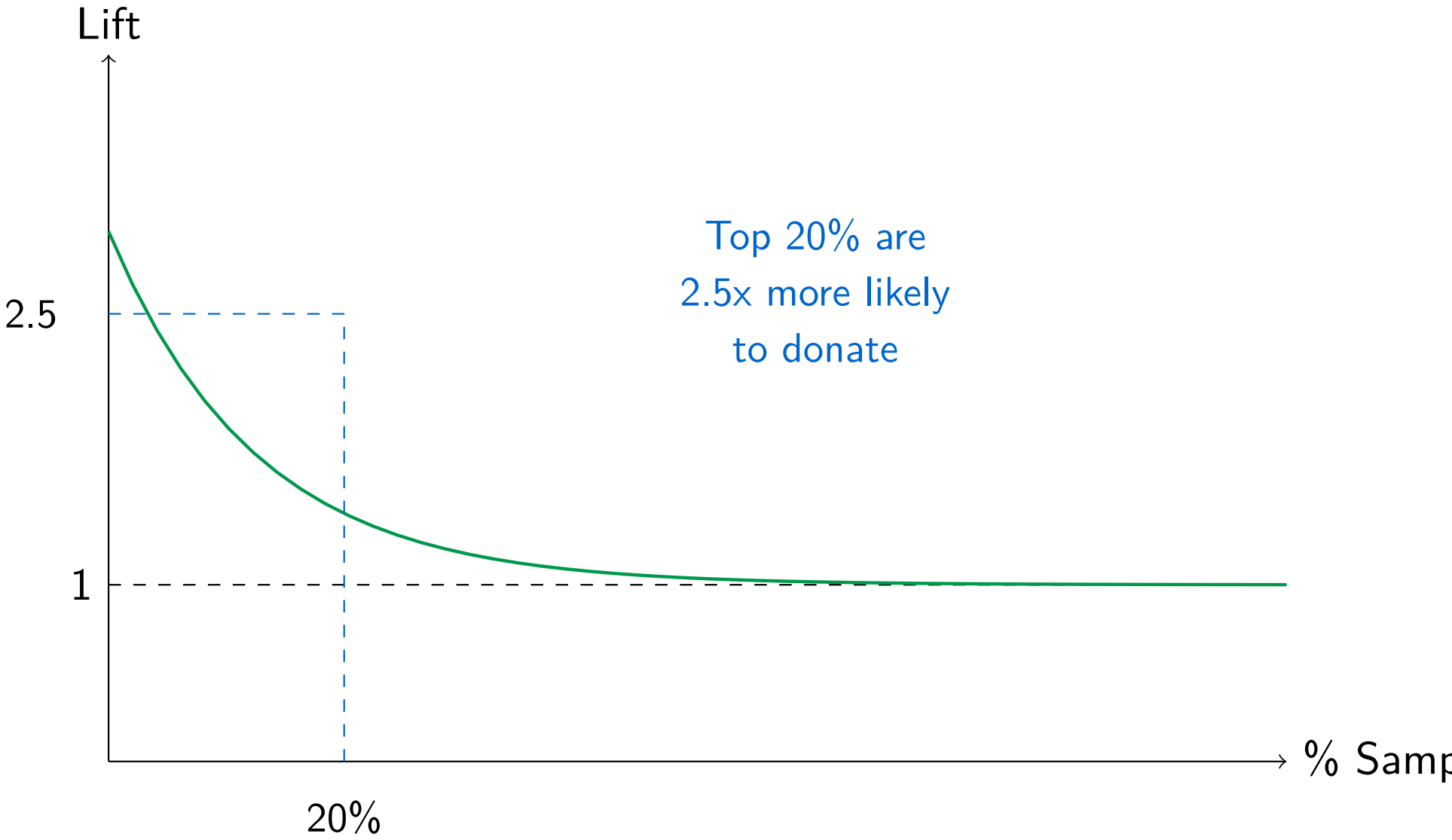
At 10% of sample: - Only contacting the BEST prospects - Very high concentration of donors - Lift might be 5x or more

At 50% of sample: - Contacting good and mediocre prospects - Lower concentration - Lift might be 2x

```r
calculate_lift <- function(actual, predicted) {

  # Overall target rate (baseline)
  baseline_rate <- mean(actual)

  # Sort by prediction
  df <- data.frame(
    actual = actual,
    predicted = predicted
  ) %>%
    arrange(desc(predicted))

  # Continue on next slide...
}
```

```r
calculate_lift <- function(actual, predicted) {

  baseline_rate <- mean(actual)

  df <- data.frame(
    actual = actual,
    predicted = predicted
  ) %>%
    arrange(desc(predicted))

  # Calculate lift at each point
  df <- df %>%
    mutate(
      perc_sample = row_number() / n(),
      cum_targets = cumsum(actual),
      cum_sample = row_number(),

      # Lift = (targets found) / (expected if random)
      lift = (cum_targets / cum_sample) /
             baseline_rate
    )
```

```r
# Calculate lift
lift_data <- calculate_lift(
  actual = test_data$donated,
  predicted = test_preds
)

# Plot
ggplot(lift_data,
       aes(x = perc_sample, y = lift)) +
  geom_line(color = "darkgreen",
            size = 1.2) +
  geom_hline(yintercept = 1,
             linetype = "dashed",
             color = "black") +
  labs(title = "Lift Curve",
       x = "Percentage of Sample",
       y = "Lift") +
  theme_minimal()
```


Lift
2.5
1
Top 20% are
2.5x more likely
to donate
20%
% Samp

Scenario: Donation campaign

- Overall donation rate: 5%
- Budget allows contacting 20% of database

Without Model (Random): - Contact 20,000 people - Expect: 20,000 $\times$ 5% = 1,000 donors

With Model (Lift = 3): - Contact 20,000 people (top 20%) - Expect: 1,000 $\times$ 3 = 3,000 donors

```r
# Calculate for both models
lift_model1 <- calculate_lift(actual, pred1)
lift_model2 <- calculate_lift(actual, pred2)

# Add identifiers
lift_model1$model <- "Model 1"
lift_model2$model <- "Model 2"

# Combine
lift_combined <- rbind(lift_model1,
                       lift_model2)

# Plot
ggplot(lift_combined,
       aes(x = perc_sample, y = lift,
           color = model)) +
  geom_line(size = 1.2) +
  geom_hline(yintercept = 1,
             linetype = "dashed") +
  theme_minimal()
```

Section 9

Business Decision Making

Translating Models into Dollars

Key Questions:

1. How many people should we contact?
2. What's the expected profit?
3. Is the campaign worth running?

Answer: Use the lift curve to estimate ROI!

Components:

- **Revenue per donor**: \$50
- **Cost per contact**: \$2
- **Population size**: 100,000
- **Baseline donation rate**: 5%

Formula:

$$\text{Profit} = (\text{Revenue} \times \text{Donors}) - (\text{Cost} \times \text{Contacts})$$

```r
# Define campaign parameters
calc_profit <- function(perc_contacted,
                        lift,
                        pop_size = 100000,
                        baseline_rate = 0.05,
                        revenue_per_donor = 50,
                        cost_per_contact = 2) {

  # How many people contacted?
  n_contacted <- pop_size * perc_contacted

  # Continue on next slide...
}
```

```r
calc_profit <- function(perc_contacted, lift,
                        pop_size = 100000,
                        baseline_rate = 0.05,
                        revenue_per_donor = 50,
                        cost_per_contact = 2) {

  n_contacted <- pop_size * perc_contacted

  # Expected donation rate with model
  expected_rate <- baseline_rate * lift

  # Expected number of donors
  n_donors <- n_contacted * expected_rate

  # Continue on next slide...
}
```

```r
calc_profit <- function(perc_contacted, lift,
                        pop_size = 100000,
                        baseline_rate = 0.05,
                        revenue_per_donor = 50,
                        cost_per_contact = 2) {

  n_contacted <- pop_size * perc_contacted
  expected_rate <- baseline_rate * lift
  n_donors <- n_contacted * expected_rate

  # Calculate profit
  revenue <- n_donors * revenue_per_donor
  cost <- n_contacted * cost_per_contact
  profit <- revenue - cost

  return(profit)
}
```

```r
# Scenario 1: Contact top 20% with lift of 2.5
profit_1 <- calc_profit(
  perc_contacted = 0.20,
  lift = 2.5
)

cat("Profit (20%, lift 2.5):",
    scales::dollar(profit_1), "\n")

# Scenario 2: Contact everyone (no targeting)
profit_2 <- calc_profit(
  perc_contacted = 1.0,
  lift = 1.0
)

cat("Profit (100%, no model):",
    scales::dollar(profit_2), "\n")
```

```
Profit (20%, lift 2.5): $85,000
Profit (100%, no model): $50,000
```

Insight:

- Targeted approach: $85,000 profit
- Contact everyone: $50,000 profit
- **Benefit of targeting: $35,000 more profit**
- **With 80% fewer contacts!**

```r
# Calculate profit at different percentages
contact_levels <- seq(0.05, 1.0, by = 0.05)

profit_by_level <- data.frame(
  perc_contacted = contact_levels,
  lift = NA,
  profit = NA
)

# For each contact level...
for (i in 1:nrow(profit_by_level)) {
  perc <- profit_by_level$perc_contacted[i]

  # Continue on next slide...
}
```

```r
for (i in 1:nrow(profit_by_level)) {
  perc <- profit_by_level$perc_contacted[i]

  # Get lift at this percentage from lift curve
  lift_at_perc <- lift_data %>%
    filter(perc_sample <= perc) %>%
    slice_tail(n = 1) %>%
    pull(lift)

  # Calculate profit
  profit_at_perc <- calc_profit(
    perc_contacted = perc,
    lift = lift_at_perc
  )

  # Store results
  profit_by_level$lift[i] <- lift_at_perc
  profit_by_level$profit[i] <- profit_at_perc
}
```

```r
# Find optimal point
optimal <- profit_by_level %>%
  filter(profit == max(profit))

# Plot
ggplot(profit_by_level,
       aes(x = perc_contacted, y = profit)) +
  geom_line(color = "darkgreen",
            size = 1.2) +
  geom_point(data = optimal,
             color = "red", size = 4) +
  geom_hline(yintercept = 0,
             linetype = "dashed") +
  labs(title = "Profit by Contact Percentage",
       x = "% of Database Contacted",
       y = "Expected Profit ($)") +
  scale_y_continuous(labels = scales::dollar) +
  theme_minimal()
```

Key Features:

- Peak: Optimal contact percentage
- Before peak: Too few contacts (missed opportunities)
- After peak: Too many contacts (wasted money)
- Negative region: Campaign loses money

Business Recommendation: Contact the percentage at the peak!

```r
# Business goal: Reach 80% of potential donors
target_coverage <- 0.80

# Find required contact percentage
required_contact <- gains_data %>%
  filter(perc_targets >= target_coverage) %>%
  slice_head(n = 1) %>%
  pull(perc_sample)

cat("To reach", target_coverage * 100,
    "% of donors,\n")
cat("Contact", round(required_contact * 100, 1),
    "% of database\n")
```

Output:

```
To reach 80% of donors,
Contact 35.2% of database
```

Comparing Strategies

Strategy	Contact %	Donors	Profit
Random	100%	5,000	$50,000
Model (20%)	20%	2,500	$85,000
Model (35%)	35%	3,800	$120,000
Model (Optimal)	30%	3,500	$130,000

Result: Same money, 2.6x more profit!

```r
# Generate business report
create_report <- function(lift_data,
                          profit_data) {

  optimal_perc <- profit_data %>%
    filter(profit == max(profit)) %>%
    pull(perc_contacted)

  max_profit <- max(profit_data$profit)

  cat("=== CAMPAIGN RECOMMENDATION ===\n\n")
  cat("Optimal Strategy:\n")
  cat("- Contact", round(optimal_perc*100, 1),
      "% of database\n")
  cat("- Expected profit:",
      scales::dollar(max_profit), "\n\n")

  # Continue on next slide...
}
```

```r
create_report <- function(lift_data,
                          profit_data) {
  # ... previous code ...

  # Compare to random
  random_profit <- calc_profit(1.0, 1.0)
  improvement <- max_profit - random_profit

  cat("Compared to random selection:\n")
  cat("- Additional profit:",
      scales::dollar(improvement), "\n")
  cat("- Efficiency gain:",
      round((1 - optimal_perc) * 100, 1),
      "% fewer contacts\n")
}

# Generate the report
create_report(lift_data, profit_by_level)
```

1. **Cumulative Gains**: Shows % of targets reached Answers: "Should I contact top 30%?"
2. **Lift**: Shows how much better than random Answers: "Is this model worth using?"
3. **Profit Analysis**: Translates to dollars Answers: "What's the expected ROI?"
4. **Optimization**: Find best contact percentage Answers: "What's my optimal strategy?"

Metric	Best For
AUC	Model development Comparing algorithms
Cumulative Gains	Determining contact strategy Estimating coverage
Lift	Explaining to stakeholders Justifying model use
Profit Analysis	Budget decisions ROI calculations

For Data Scientists

- Always create gains and lift curves - Calculate profit for realistic scenarios - Document assumptions clearly

For Business Presentations

- Lead with lift curve (easiest to understand) - Show profit analysis second - Keep gains curve for detailed questions - Use real dollar amounts

Mistake 1: Using lift at 100% of sample

- Lift always equals 1 at 100%
- Look at initial lift instead

Mistake 2: Ignoring campaign costs

- Profit must account for costs
- Different costs change optimal strategy

Mistake 3: Not validating on test data

- Always use holdout set
- Training data gives overly optimistic curves

```r
# Divide predictions into 10 groups (deciles)
decile_analysis <- lift_data %>%
  mutate(
    decile = ntile(desc(predicted), 10)
  ) %>%
  group_by(decile) %>%
  summarise(
    n = n(),
    n_targets = sum(actual),
    rate = mean(actual),
    lift = rate / mean(lift_data$actual)
  ) %>%
  arrange(decile)

print(decile_analysis)
```

1. Build and validate model
2. Generate predictions on test set
3. Calculate cumulative gains curve
4. Calculate lift curve
5. Estimate profit at different contact levels
6. Find optimal contact percentage
7. Create executive summary
8. Present to stakeholders

In Next Lecture:

- Advanced model comparison techniques
- Cost-sensitive learning
- Threshold optimization
- Deployment considerations

Practice:

- Create gains and lift curves for your models
- Calculate expected profits
- Present findings to a partner

Questions?

Review:

- Gains curve: % of targets reached
- Lift curve: Factor better than random
- Profit analysis: Expected ROI
- Optimization: Find best strategy

Next: Classwork 3 - Model Evaluation Exercise # Introduction

Understanding What Drives Your Model

The Final Step in Model Validation:

After building a model and evaluating its performance, we need to understand:

- Are the variables behaving as expected?
- Do the relationships make business sense?
- Can we explain the model to stakeholders?

1. Build model ✓
2. Evaluate using AUC ✓
3. Evaluate using gains/lift curves ✓
4. Verify variables are interpretable ← We are here

Why This Matters:

- Models with counterintuitive patterns may have data quality issues
- Stakeholders need to trust the model
- Regulatory requirements may demand explainability

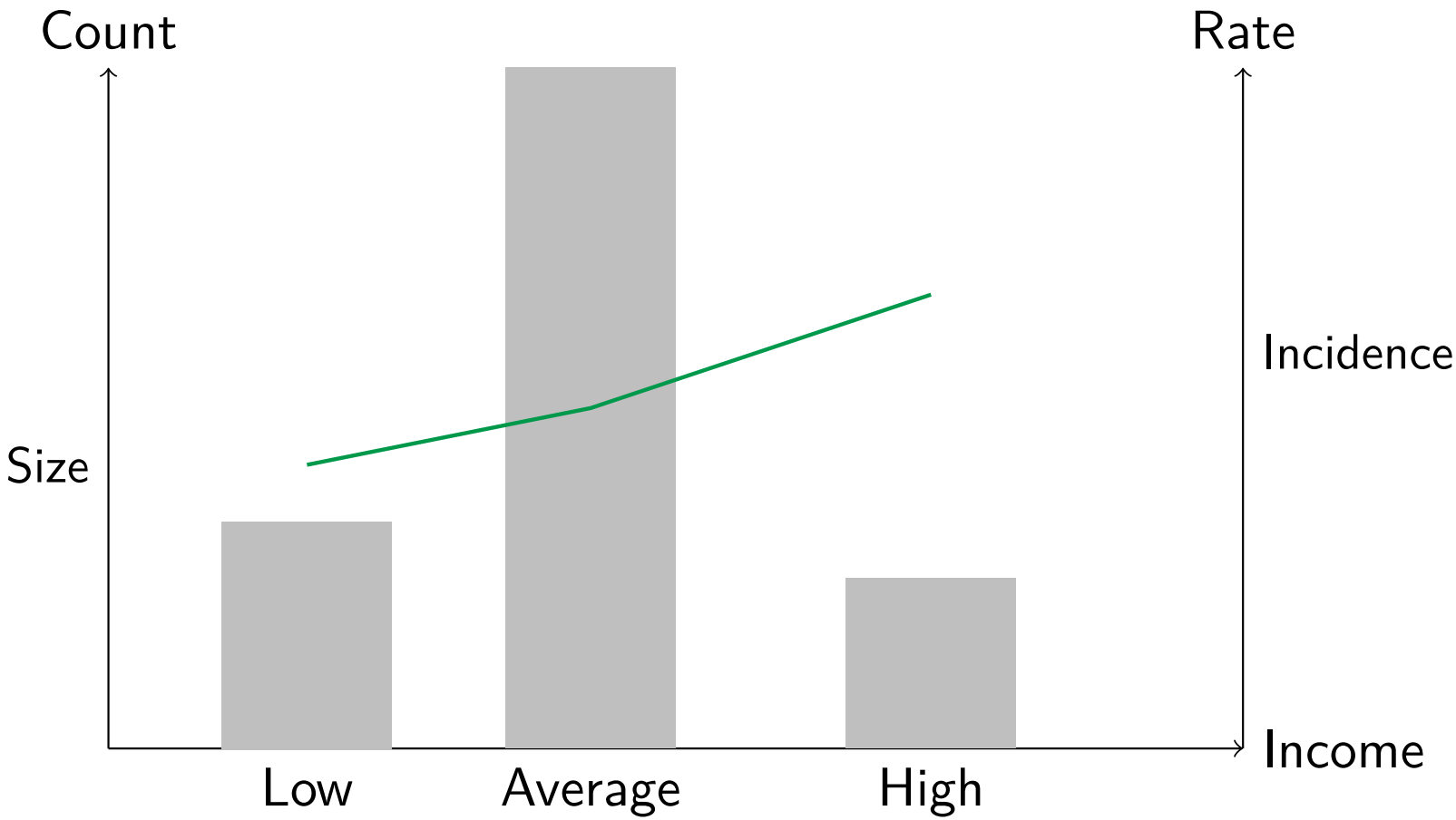
Visual tools that show the relationship between each predictor variable and the target

They Answer:

- How does age affect donation probability?
- Is income positively or negatively related to the target?
- Are there unexpected patterns in the data?

Components:

- **Bar chart**: Sample size in each category
- **Line plot**: Target incidence rate
- **Combined view**: Both on same graph



Interpretation: Average income group has most people, but donation rate increases with income.

Model Validation

Verify that relationships make sense

Business Communication

Explain model behavior to non-technical stakeholders

Data Quality Check

Identify unexpected patterns that may indicate errors

Feature Engineering

Discover non-linear patterns suggesting transformations

Section 10

Section 1: Categorical Variables

For categorical variables (like gender, country, income bracket):

- Each category gets a bar showing sample size
- Line shows target incidence rate
- No discretization needed!

Example Variables:

- Gender (M/F)
- Income level (Low/Average/High)
- Country
- Education level

```r
# Sample data structure
gender_summary <- data.frame(
  Gender = c("F", "M"),
  Size = c(30000, 45000),
  Incidence = c(0.053, 0.046)
)

print(gender_summary)
```

```
  Gender  Size Incidence
1      F 30000     0.053
2      M 45000     0.046
```

Interpretation:

- More males in dataset (45,000 vs 30,000)
- But females have slightly higher donation rate (5.3% vs 4.6%)

```r
income_summary <- data.frame(
  Income = c("Low", "Average", "High"),
  Size = c(20850, 62950, 16200),
  Incidence = c(0.0431, 0.0492, 0.0615)
)

print(income_summary)
```

```
   Income  Size Incidence
1     Low 20850    0.0431
2 Average 62950    0.0492
3    High 16200    0.0615
```

Key Insights:

- Most people in "Average'' income (62,950)
- Donation rate increases with income
- Clear positive relationship

Section 11

Section 2: Continuous Variables

Problem: Variables like age or income can have hundreds of unique values

Example:

- Age ranges from 18 to 95
- Cannot create 77 separate bars!
- Graph would be unreadable

Solution: **Discretization**

Convert continuous variable into categories (bins)

Converting continuous values into discrete bins

Before Discretization:

Age: 23, 24, 25, 26, 27, 28, 29, 30, 31, 32...

After Discretization (5 bins):

- Bin 1: [18, 30]
- Bin 2: (30, 40]
- Bin 3: (40, 50]
- Bin 4: (50, 60]
- Bin 5: (60, 110]

Equal-Width Bins

Using `cut()`

- Divide range into equal intervals - May have unequal sample sizes - Good for interpretation

Example: Age - [18-30] - [31-43] - [44-56] - [57-69] - [70+]

Equal-Frequency Bins

Using `cut()` with quantiles

- Each bin has similar number of observations - Better for analysis - Bins may have different widths

Example: Age - [18-35] - [36-42] - [43-51] - [52-63] - [64+]

```r
# Load library
library(tidyverse)

# Create sample data
set.seed(123)
ba

## Discretization in R: Basic Example

\small
```

```r
# Load library
library(tidyverse)

# Create sample data
set.seed(123)
basetable <- data.frame(
  age = sample(18:95, 1000, replace = TRUE),
  donated = rbinom(1000, 1, 0.05)
)
```

```
[18,33]  (33,47]  (47,62]  (62,78]  (78,95]
    201      199      200      200      200
```

Observations:

- Each bin has approximately 200 observations
- Equal-frequency binning achieved
- Age ranges differ per bin

Decision Rule:

If a variable has more than 5-10 unique values, discretize it

```
# Function to check if discretization needed
should_discretize <- function(variable, threshold = 5) {
  n_unique <- length(unique(variable))
  return(n_unique > threshold)
}

# Test it
should_discretize(basetable$age, threshold = 5)
# TRUE - many unique values

should_discretize(basetable$gender, threshold = 5)
# FALSE - only 2 categories
```

```r
# List of model variables
model_vars <- c("age", "income", "gender",
                "mean_gift", "max_gift")

# Check which need discretization
for (var in model_vars) {
  needs_disc <- should_discretize(basetable[[var]],
                                  threshold = 5)

  cat(var, "needs discretization:", needs_disc, "\n")
}
```

```
age needs discretization: TRUE
income needs discretization: FALSE
gender needs discretization: FALSE
mean_gift needs discretization: TRUE
max_gift needs discretization: TRUE
```

```r
# Variables to discretize
continuous_vars <- c("age", "mean_gift", "max_gift")

# Number of bins
n_bins <- 5

# Discretize all at once
for (var in continuous_vars) {
  # Create new variable name
  new_var <- paste0(var, "_disc")

  # Discretize using quantiles
  basetable[[new_var]] <- cut(
    basetable[[var]],
    breaks = quantile(basetable[[var]],
                      probs = seq(0, 1, 1/n_bins),
                      na.rm = TRUE),
    include.lowest = TRUE
  )
}
```

```r
# Problem: Quantile breaks create messy labels
table(basetable$age_disc)
# [18,33.4] (33.4,47.8] (47.8,62.2] ...

# Solution: Define clean breaks manually
age_breaks <- c(18, 30, 40, 50, 60, 110)

basetable$age_disc_clean <- cut(
  basetable$age,
  breaks = age_breaks,
  include.lowest = TRUE,
  right = TRUE
)

table(basetable$age_disc_clean)
```

```
[18,30] (30,40] (40,50] (50,60] (60,110]
    247     312     198     143      100
```

Trade-offs:

Bins	Advantages	Disadvantages
3	Simple, clear	May miss patterns
5	Good balance	Standard choice
10	Detailed patterns	May be too granular

General Guidelines:

- **Small datasets** ($<$ 1,000): Use 3-4 bins
- **Medium datasets** (1,000-10,000): Use 5 bins
- **Large datasets** ($>$ 10,000): Can use 7-10 bins

Section 12

Section 3: Creating PIG Tables

Core data structure for creating graphs

Three Columns:

1. **Variable value** (or bin)
2. **Size**: Number of observations in each category
3. **Incidence**: Target rate in each category

```r
# Example structure
pig_table <- data.frame(
  category = c("Low", "Medium", "High"),
  Size = c(1000, 2500, 500),
  Incidence = c(0.03, 0.05, 0.08)
)
```

```r
# Function to create predictor insight graph table
create_pig_table <- function(data, target, variable) {

  # Group by the variable
  pig_table <- data %>%
    group_by(!!sym(variable)) %>%
    summarise(
      Size = n(),
      Incidence = mean(!!sym(target), na.rm = TRUE),
      .groups = 'drop'
    )

  # Rename first column for consistency
  names(pig_table)[1] <- "Category"

  return(pig_table)
}
```

Key Functions:

- group_by(): Group data by variable

```r
# Create PIG table
income_pig <- create_pig_table(
  data = basetable,
  target = "donated",
  variable = "income"
)

print(income_pig)
```

```
  Category  Size Incidence
1      Low 20850   0.04310
2  Average 62950   0.04920
3     High 16200   0.06150
```

Interpretation Ready:

- Formatted for plotting
- Clear column names
- Calculated statistics

```r
# List of variables to analyze
variables <- c("income", "gender",
               "age_disc", "mean_gift_disc")

# Create empty list to store tables
pig_tables <- list()

# Loop through variables
for (var in variables) {
  pig_tables[[var]] <- create_pig_table(
    data = basetable,
    target = "donated",
    variable = var
  )
}

# Access specific table
print(pig_tables[["income"]])
```

```r
# Look at all tables
for (var_name in names(pig_tables)) {
  cat("\n=== PIG Table for", var_name, "===\n")
  print(pig_tables[[var_name]])
}
```

```
=== PIG Table for income ===
  Category  Size Incidence
1      Low 20850   0.04310
2  Average 62950   0.04920
3     High 16200   0.06150

=== PIG Table for gender ===
  Category  Size Incidence
1        F 30000   0.05300
2        M 45000   0.04600
```

```r
# Check for missing values in variables
check_missing <- function(data, variables) {
  for (var in variables) {
    n_missing <- sum(is.na(data[[var]]))
    pct_missing <- round(n_missing / nrow(data) * 100, 2)

    if (n_missing > 0) {
      cat(var, ":", n_missing, "missing (",
          pct_missing, "%)\n")
    }
  }
}

# Use it
check_missing(basetable, model_vars)
```

Best Practice: Handle missing values before creating PIG tables

Section 13

Section 4: Plotting PIG Graphs

Dual-axis visualization

Left Y-Axis: Sample Size (bars)

Right Y-Axis: Target Incidence (line)

X-Axis: Variable categories

Why This Works:

- See both distribution AND relationship
- Bars show data quality (sample sizes)
- Line shows predictive pattern

```r
library(ggplot2)

# Simple line plot of incidence
ggplot(income_pig,
       aes(x = Category, y = Incidence, group = 1)) +
  geom_line(color = "darkgreen", size = 1.5) +
  geom_point(color = "darkgreen", size = 3) +
  labs(title = "Donation Rate by Income",
       x = "Income Level",
       y = "Donation Incidence") +
  theme_minimal() +
  ylim(0, max(income_pig$Incidence) * 1.1)
```

Purpose: Start simple, verify the pattern

```r
plot_pig <- function(pig_table, var_name) {

  # Calculate scaling factor
  scale_factor <- max(pig_table$Size) /
                  max(pig_table$Incidence)

  # Create plot
  p <- ggplot(pig_table, aes(x = Category)) +
    geom_col(aes(y = Size),
             fill = "lightgray",
             alpha = 0.7,
             width = 0.6) +
    geom_line(aes(y = Incidence * scale_factor,
                  group = 1),
              color = "darkgreen",
              size = 1.5) +
    geom_point(aes(y = Incidence * scale_factor),
               color = "darkgreen",
               size = 3)

  # Continued on next slide
```

```r
plot_pig <- function(pig_table, var_name) {
  # ... previous code ...

  p <- p +
    scale_y_continuous(
      name = "Size",
      labels = scales::comma,
      sec.axis = sec_axis(
        ~./scale_factor,
        name = "Incidence",
        labels = scales::percent
      )
    ) +
    labs(title = paste("Predictor Insight Graph:",
                       var_name),
         x = var_name) +
    theme_minimal(base_size = 12) +
    theme(axis.title.y.right =
            element_text(color = "darkgreen"),
          axis.text.y.right =
```

```r
# Create plot for income
income_plot <- plot_pig(
  pig_table = income_pig,
  var_name = "Income Level"
)

print(income_plot)

# Save the plot
ggsave("income_pig.pdf",
       income_plot,
       width = 8,
       height = 6)
```

```r
# Create plots for all variables
library(gridExtra)

plot_list <- list()

for (var_name in names(pig_tables)) {
  plot_list[[var_name]] <- plot_pig(
    pig_table = pig_tables[[var_name]],
    var_name = var_name
  )
}

# Arrange in grid
grid.arrange(
  plot_list[[1]], plot_list[[2]],
  plot_list[[3]], plot_list[[4]],
  ncol = 2
)
```

Questions to Ask:

1. **Direction:** Does incidence increase or decrease?
2. **Magnitude:** How big is the difference?
3. **Pattern:** Linear, curved, or irregular?
4. **Sample sizes:** Are some groups too small?
5. **Business sense:** Does this make intuitive sense?

Good Patterns:

+ Smooth monotonic trend
+ Clear relationship
+ Adequate sample sizes
+ Matches expectations

Example: Income leads to higher donation rate

Problematic Patterns:

- Erratic ups and downs
- No clear relationship
- Tiny sample in some bins
- Contradicts domain knowledge

Example: Age shows random fluctuations

Days Since First Donation

Bin	Size	Incidence
[0, 1500]	16500	0.021
(1500, 2000]	10500	0.033
(2000, 2500]	30500	0.048
(2500, 2700]	28000	0.061
(2700, 3000]	14500	0.066

Interpretation:

- Clear increasing trend
- More recent donors more likely to donate again
- Makes business sense
- Reasonable sample sizes

Hypothetical: Number of Website Visits

Bin	Size	Incidence
[0, 5]	25000	0.05
(5, 10]	8000	0.03
(10, 20]	1500	0.12
(20, 50]	300	0.02
(50+]	50	0.40

Problems:

- Erratic pattern (no clear trend)
- Very small samples in last bins
- Extreme values may be data errors
- Suggests variable needs investigation

Action Steps:

1. **Investigate data quality** - Check for outliers or errors - Verify data collection process
2. **Try different binning** - Fewer bins may smooth pattern - Custom breaks for problem areas
3. **Consider transformations** - Log transform for skewed distributions - Cap extreme values
4. **Remove or combine categories** - Merge bins with small samples - Consider excluding the variable

```r
# Save all PIG tables to Excel
library(writexl)

write_xlsx(pig_tables, "pig_tables.xlsx")

# Or save individual tables
for (var_name in names(pig_tables)) {
  filename <- paste0("pig_", var_name, ".csv")
  write.csv(pig_tables[[var_name]],
            filename,
            row.names = FALSE)
}

# Save all plots
for (var_name in names(plot_list)) {
  filename <- paste0("pig_", var_name, ".pdf")
  ggsave(filename,
         plot_list[[var_name]],
         width = 8,
         height = 6)
}
```

Data Preparation

- Discretize continuous variables (5 bins default) - Handle missing values before creating tables - Check for outliers and data quality

Visualization

- Always show both size and incidence - Use consistent color scheme - Label axes clearly - Include variable name in title

Interpretation

- Look for monotonic trends - Verify adequate sample sizes - Check business logic - Document unexpected patterns

When to Use PIG:

1. **Before modeling**: Understand variable relationships
2. **During feature selection**: Identify informative variables
3. **After modeling**: Validate model makes sense
4. **For communication**: Explain model to stakeholders

PIG graphs complement, not replace:

- Model performance metrics (AUC)
- Business metrics (lift, profit)
- Statistical tests

```r
# Variables in bank marketing model
bank_variables <- c("age_disc", "balance_disc",
                    "duration_disc", "campaign",
                    "education", "marital")

# Create all PIG tables
bank_pig_tables <- lapply(bank_variables, function(var) {
  create_pig_table(
    data = bank_data,
    target = "subscribed",
    variable = var
  )
})

names(bank_pig_tables) <- bank_variables

# Create all plots
bank_plots <- lapply(bank_variables, function(var) {
  plot_pig(bank_pig_tables[[var]], var)
})
```

Key Messages from PIG Graphs:

- "Higher income customers are 40% more likely to donate"
- "Recent donors ($<$ 6 months) show 3x higher response rate"
- "Age has a positive but modest effect"
- "Gender shows minimal difference"

Use PIG graphs in presentations:

- Clear, visual, easy to understand
- Shows both pattern and data distribution
- Builds trust in model

1. **Purpose**: Validate variable relationships make sense
2. **Discretization**: Convert continuous to categorical
3. **PIG Tables**: Size and incidence by category
4. **Visualization**: Dual-axis plots
5. **Interpretation**: Look for clear patterns
6. **Communication**: Present to stakeholders

Technical

- Discretize continuous variables into 3-7 bins - Create tables with Size and Incidence - Use dual-axis plots for visualization

Analytical

- Look for monotonic relationships - Check sample sizes are adequate - Verify patterns match domain knowledge

Communication

- PIG graphs are stakeholder-friendly - They build trust in models - Essential for model documentation

What You've Learned

1. Construct the basetable
2. Build predictive models using logistic regression
3. Perform forward variable selection
4. Evaluate with AUC and ROC curves
5. Assess business value with gains/lift curves
6. Validate interpretability with PIG graphs

Technical Excellence

Building accurate models requires rigorous methodology

Business Value

Models must deliver measurable ROI

Interpretability

Stakeholders must trust and understand the model

All three are essential for success!

Questions?

Prof. Asc. Endri Raco, PhD

Polytechnic University of Tirana

endri.raco@upt.al

Ready for Classwork 4!

```r
# Load library
library(tidyverse)

# Create sample data
set.seed(123)
basetable <- data.frame(
  age = sample(18:95, 1000, replace = TRUE),
  donated = rbinom(1000, 1, 0.05)
)

# Method 1: Equal-frequency bins (quantile-based)
basetable$age_disc <- cut(
  basetable$age,
  breaks = quantile(basetable$age,
                    probs = seq(0, 1, 0.2)),
  include.lowest = TRUE
)

# Check the bins
table(basetable$age_disc)
```

```
[18,33]  (33,47]  (47,62]  (62,78]  (78,95]
    201      199      200      200      200
```

Observations:

- Each bin has approximately 200 observations
- Equal-frequency binning achieved
- Age ranges differ per bin

Decision Rule:

If a variable has more than 5-10 unique values, discretize it

```r
# Function to check if discretization needed
should_discretize <- function(variable, threshold = 5) {
  n_unique <- length(unique(variable))
  return(n_unique > threshold)
}

# Test it
should_discretize(basetable$age, threshold = 5)
# TRUE - many unique values

should_discretize(basetable$gender, threshold = 5)
# FALSE - only 2 categories
```

```r
# List of model variables
model_vars <- c("age", "income", "gender",
                "mean_gift", "max_gift")

# Check which need discretization
for (var in model_vars) {
  needs_disc <- should_discretize(basetable[[var]],
                                  threshold = 5)

  cat(var, "needs discretization:", needs_disc, "\n")
}
```

```
age needs discretization: TRUE
income needs discretization: FALSE
gender needs discretization: FALSE
mean_gift needs discretization: TRUE
max_gift needs discretization: TRUE
```

```r
# Variables to discretize
continuous_vars <- c("age", "mean_gift", "max_gift")

# Number of bins
n_bins <- 5

# Discretize all at once
for (var in continuous_vars) {
  # Create new variable name
  new_var <- paste0(var, "_disc")

  # Discretize using quantiles
  basetable[[new_var]] <- cut(
    basetable[[var]],
    breaks = quantile(basetable[[var]],
                      probs = seq(0, 1, 1/n_bins),
                      na.rm = TRUE),
    include.lowest = TRUE
  )
}
```

```r
# Problem: Quantile breaks create messy labels
table(basetable$age_disc)
# [18,33.4] (33.4,47.8] (47.8,62.2] ...

# Solution: Define clean breaks manually
age_breaks <- c(18, 30, 40, 50, 60, 110)

basetable$age_disc_clean <- cut(
  basetable$age,
  breaks = age_breaks,
  include.lowest = TRUE,
  right = TRUE
)

table(basetable$age_disc_clean)
```

```
[18,30] (30,40] (40,50] (50,60] (60,110]
    247     312     198     143      100
```

Trade-offs:

Bins	Advantages	Disadvantages
3	Simple, clear	May miss patterns
5	Good balance	Standard choice
10	Detailed patterns	May be too granular

General Guidelines:

- **Small datasets** ($< 1{,}000$): Use 3-4 bins
- **Medium datasets** (1,000-10,000): Use 5 bins
- **Large datasets** ($> 10{,}000$): Can use 7-10 bins