The Basetable Timeline

Intermediate Predictive Analytics

Constructing Temporal Structures for Predictive Modeling

Prof. Asc.Endri Raco, Ph.D.

Department of Mathematical Engineering
Polytechnic University of Tirana

November 2025

Section 1

Introduction

Foundations of Predictive Analytics I

- Build predictive models
- Evaluate predictive models
- Present predictive models to business stakeholders

Foundations of Predictive Analytics II

- **Construct the basetable**

By the end of this lecture, you will be able to:

1. Define and construct a basetable for predictive modeling
2. Understand the temporal structure of prediction problems
3. Implement timeline-compliant data partitioning
4. Define population eligibility criteria
5. Create binary and continuous target variables
6. Apply set operations for population filtering

Section 2

The Basetable

Definition

A basetable is a **structured data matrix** where:

- Each **row** represents an observation unit (customer, donor, patient)
- Each **column** represents a variable (predictor or target)

Population

Population

The set of **observation units** eligible for analysis.

Candidate predictors

Population	Age	Gender	Previous gifts	
	25	F	12	
	60	M	5	
	45	F	9	

Candidate Predictors

Historical features calculated from data available **before** the observation point.

	Candidate predictors			Target
	Age	Gender	Previous gifts	Donate
Population	25	F	12	0
	60	M	5	1
	45	F	9	0

Target Variable

The outcome variable measured **after** the observation point that we aim to predict.

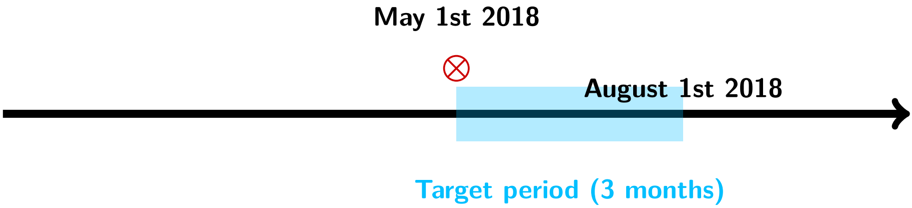
Section 3

The Timeline

Temporal Structure

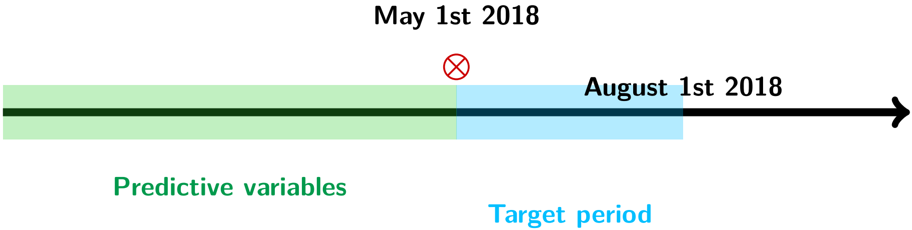
Predictive modeling requires a clear **temporal separation** between:

- **Past**: Data used to calculate predictors
- **Future**: Outcomes to be predicted



Key Dates

- **Observation date**: Reference point (e.g., mailing date)
- **Target period**: Window for measuring outcomes



Critical Principle

No data leakage: Predictors must be calculated using **only** information available before the observation date.

Why is Timeline Important?

1. **Prevents data leakage**: Ensures predictors don't contain future information
2. **Mimics deployment**: Replicates real-world prediction scenarios
3. **Valid evaluation**: Enables honest assessment of model performance
4. **Temporal validity**: Accounts for time-dependent patterns

Real-world Example

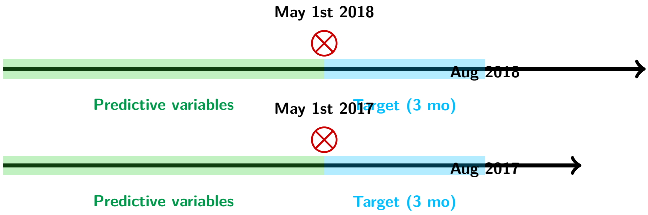
To predict donations in May-July 2018 using a mailing sent May 1st, we can only use donor characteristics and behavior from before May 1st.

Section 4

Reconstructing History

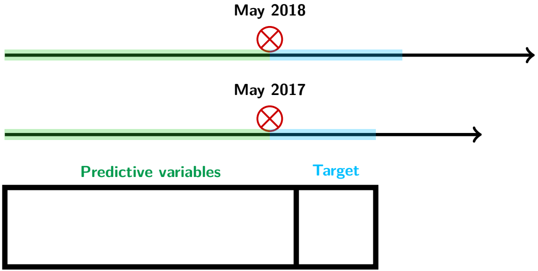
Training Data Construction

To build robust models, we need **multiple observation points** from historical data.



Multiple Snapshots

By shifting the observation date backward, we create additional training samples while maintaining timeline integrity.



Result

Each historical observation point creates a **row** in the basetable, increasing sample size for model training.

```r
# Load and prepare donation data
library(tidyverse)
library(lubridate)

gifts <- read_csv("gifts.csv") %>%
  mutate(date = as.Date(date))

# Define timeline boundaries
start_target <- as.Date("2018-05-01")  # Observation date
end_target <- as.Date("2018-08-01")    # End of target period

# Partition data by timeline
gifts_target <- gifts %>%
  filter(date >= start_target & date < end_target)

gifts_pred_variables <- gifts %>%
  filter(date < start_target)  # Only historical data
```

```r
head(gifts, 5)
```

```
## # A tibble: 5 x 3
##      id date       amount
##   <int> <date>      <dbl>
## 1     1 2015-10-16     75
## 2     1 2014-02-11    111
## 3     1 2012-03-28     93
## 4     2 2013-12-13    113
## 5     2 2012-01-10     93
```

Data Structure

Each row represents a **donation transaction** with donor ID, date, and amount.

Section 5

The Population

What is the Population?

The population is the set of **observation units** (individuals, customers, entities) who are:

1. **Eligible** for the intervention or prediction
2. **Available** in the data at the observation date
3. **Relevant** to the business problem

Example: Donor Prediction

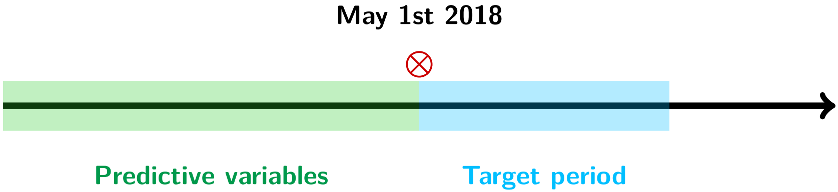
Population = donors who:

- Have a valid mailing address
- Have not opted out of communications
- Have donated at least once before the observation date

	Candidate predictors			Target
	Age	Gender	Prev. gifts	Donate
Population	25	F	12	0
	60	M	5	1
	45	F	9	0

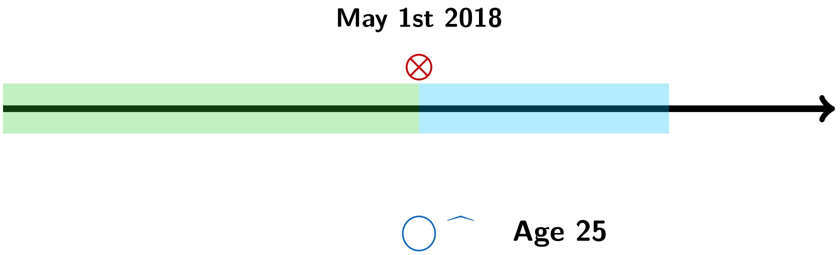
Eligibility Criteria

- Address available
- Privacy settings allow contact
- Active in the system



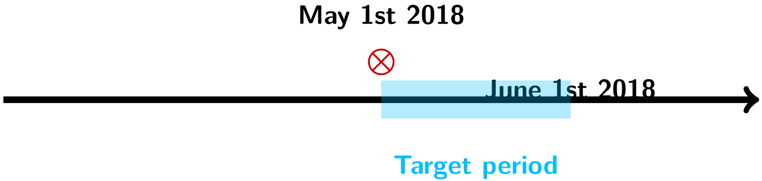
Temporal Consistency

Age must be calculated as of the **observation date** (May 1st, 2018), not current age.



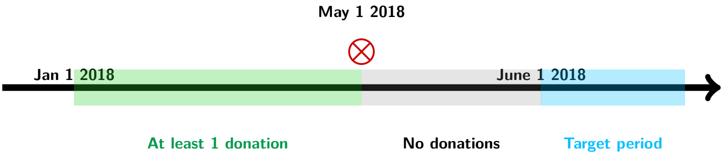
Implementation

```
age_at_observation = year(observation_date) -
year(birth_date)
```



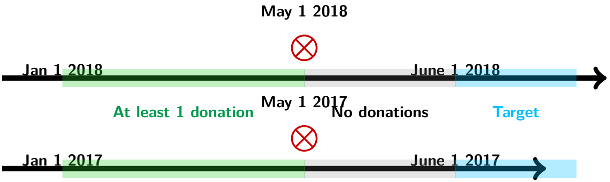
Population Filtering by Donation History

To ensure population has potential to donate, we often require **at least one prior donation**.



Inclusion Criterion

Include donors with ≥ 1 donation between Jan 1st and May 1st, but **exclude** those who donated between May 1st and June 1st.



Multiple Time Points

Apply the same logic to create populations for 2017, 2016, etc., building a larger training dataset.

```r
# Identify donors to INCLUDE (donated in 2016)
donations_2016 <- gifts %>%
  filter(year(date) == 2016)

donors_include <- unique(donations_2016$id)

# Identify donors to EXCLUDE (donated Jan-Apr 2017)
donations_2017_early <- gifts %>%
  filter(year(date) == 2017, month(date) < 5)

donors_exclude <- unique(donations_2017_early$id)

# Population = Include \ Exclude
population <- setdiff(donors_include, donors_exclude)
```

```r
# Create example donor sets
set.seed(456)
donors_include <- c(1002, 3043, 4934, 5012, 7834, 2451, 3047)
donors_exclude <- c(2451, 3047, 4474)

# Apply set difference
population <- setdiff(donors_include, donors_exclude)
population
```

```
## [1] 1002 3043 4934 5012 7834
```

Set Difference Operation

`setdiff(A, B)` returns elements in set A that are **not** in set B.

Section 6

The Target Variable

What is a Target Variable?

The target (dependent variable, outcome, label) is the **quantity we aim to predict**, measured during the target period.

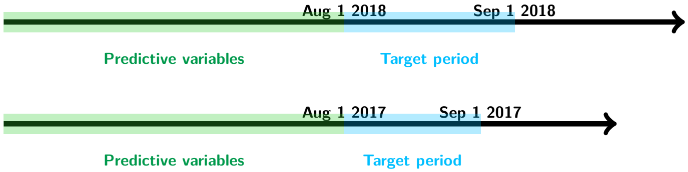
Types of Targets

- **Binary**: Did event occur? (Yes/No, 1/0)
  - Example: Donated (1) or not (0)
- **Continuous**: What magnitude? (Real number)
  - Example: Total donation amount ($)
- **Categorical**: Which category?
  - Example: Customer segment (A, B, C)

Target Period Selection

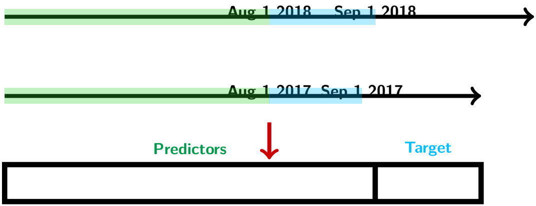
The target period should:

- Be **actionable** (e.g., campaign duration)
- Match **business cycle** (quarterly, monthly)
- Provide sufficient **signal** (not too short/long)



Consistent Target Definition

The **same target definition** must be applied across all historical observation points for valid model training.



Basetable Construction

Each timeline generates one row in the basetable with historical predictors and future target.

```r
# Load target period outcomes (e.g., list of unsubscribers)
unsubscribe_2017 <- c(90112, 65537, 24577, 8196, 73737)

# Create basetable with donor IDs from population
basetable <- tibble(donor_id = population)

# Define binary target: 1 if unsubscribed, 0 otherwise
basetable <- basetable %>%
  mutate(target = if_else(donor_id %in% unsubscribe_2017,
                          1, 0))
```

Binary Encoding

1 = event occurred (positive class), 0 = event did not occur (negative class)

```r
# Example binary target creation
unsubscribe_2017 <- c(65537, 65540)
basetable <- tibble(
  donor_id = c(65537, 65538, 65539, 65540, 65541)
)

basetable <- basetable %>%
  mutate(target = if_else(donor_id %in% unsubscribe_2017, 1, 0

basetable
```

```
## # A tibble: 5 x 2
##   donor_id target
##      <dbl>  <dbl>
## 1    65537      1
## 2    65538      0
## 3    65539      0
## 4    65540      1
## 5    65541      0
```

```r
# Define target period
start_target <- as.Date("2017-01-01")
end_target <- as.Date("2018-01-01")

# Select donations in target period
gifts_target <- gifts %>%
  filter(date >= start_target & date < end_target)

# Aggregate: sum donations by donor
gifts_target_byid <- gifts_target %>%
  group_by(id) %>%
  summarize(total_amount = sum(amount), .groups = "drop")
```

```r
# Define target based on threshold (e.g., donated >$500)
high_value_donors <- gifts_target_byid %>%
  filter(total_amount > 500) %>%
  pull(id)

# Add binary target to basetable
basetable <- basetable %>%
  mutate(target = if_else(donor_id %in% high_value_donors,
                          1, 0))
```

Aggregation Strategy

For continuous outcomes, we often **aggregate** transactions (sum, mean, count) within the target period, then potentially **threshold** to create binary targets.

Section 7

Summary

Core Concepts

1. **Basetable**: Structured matrix with observations (rows) and variables (columns)
2. **Timeline**: Temporal separation between predictor calculation and target measurement
3. **Population**: Eligible observation units defined by business rules
4. **Target**: Outcome variable measured in the target period
5. **Historical reconstruction**: Multiple observation points create training samples

Golden Rules

- **No data leakage**: Predictors use only pre-observation data
- **Consistent definitions**: Same target/population logic across time
- **Timeline integrity**: Maintain temporal ordering in all operations
- **Eligibility criteria**: Population must be actionable

1. Define **business problem** and target outcome
2. Establish **observation dates** and target periods
3. Specify **population eligibility** criteria
4. Partition **data by timeline** (predictors vs. target)
5. Calculate **features** from historical data
6. Define and measure **target variable**
7. Construct **final basetable**
8. Validate **temporal integrity**

Coming Up

In the next lecture, we will cover:

- **Feature engineering**: Creating predictive variables from raw data
- **Aggregation techniques**: RFM (Recency, Frequency, Monetary) features
- **Handling missing data**: Imputation strategies
- **Feature selection**: Identifying the most predictive variables

Questions?

Section 8

Appendix

Recommended Reading

- Verbiest, N. et al. (2018). "Building Maintainable Credit Scoring Models Using Time-Consistent Strategies"
- Provost, F., & Fawcett, T. (2013). *Data Science for Business*. O'Reilly Media.
- Kuhn, M., & Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press.

R Packages

- `tidyverse`: Data manipulation and visualization
- `lubridate`: Date-time handling
- `recipes`: Feature engineering framework

Exercise: Construct a Basetable

Given a dataset of customer transactions:

1. Define an observation date (e.g., 2019-06-01)
2. Create a 3-month target period
3. Filter population: customers with ≥ 2 purchases before observation
4. Calculate predictor: total spending before observation date
5. Define binary target: purchased during target period (1/0)
6. Construct final basetable

Deliverable

A basetable with columns: `customer_id`, `total_spending`, `target`

Section 9

The Feature Engineering Mindset

Raw Data Says: "Donor 123 gave €50 last month"

Good Features Ask:

1. **Recency:** How long ago? (Yesterday? Last year?)
2. **Frequency:** Is this typical? (First time? Monthly ritual?)
3. **Monetary:** Generous or modest? (More than usual? Less?)
4. **Trend:** What's the direction? (Increasing? Decreasing?)
5. **Context:** What else matters? (Season? Life event?)

Real Example: Two donors, both gave €100 this year

- **Donor A:** €10 $\times$ 10 times $\rightarrow$ Consistent supporter
- **Donor B:** €100 $\times$ 1 time $\rightarrow$ One-time gift ?

Same total, different stories! Features capture these nuances.

Section 10

Part 1: Multi-Window Aggregation

The Problem: Behavior changes at different speeds

```r
# Three windows, three perspectives
library(lubridate)
reference_date <- as.Date("2024-01-01")

# Recent behavior (3 months)
window_3m <- gifts %>%
  filter(date >= reference_date - months(3),
         date < reference_date)

# Medium-term pattern (12 months)
window_12m <- gifts %>%
  filter(date >= reference_date - months(12),
         date < reference_date)

# Long-term history (24 months)
window_24m <- gifts %>%
  filter(date >= reference_date - months(24)
```

Too Short (1 month):

- Captures noise, not signal
- Sensitive to one-off events
- Example: "Donor gave last week because of emergency appeal"

Too Long (5 years):

- Ancient history dominates
- Misses recent changes
- Example: "Used to give a lot, but stopped 2 years ago"

Just Right (3-12 months):

- Balances recency and stability
- Captures true behavior patterns
- Aligns with business planning cycles

Rule of Thumb: Match your window to your prediction horizon
Predicting next month? Use 3-6 month features
Predicting next year? Use 12-24 month features

```r
# Aggregate each window separately
agg_3m <- window_3m %>%
  group_by(donor_id) %>%
  summarise(
    donations_3m = sum(amount),     # Total given
    count_3m = n(),                 # How many times
    avg_3m = mean(amount)           # Average gift
  )

agg_12m <- window_12m %>%
  group_by(donor_id) %>%
  summarise(
    donations_12m = sum(amount),
    count_12m = n(),
    avg_12m = mean(amount)
  )

# Combine into basetable
```

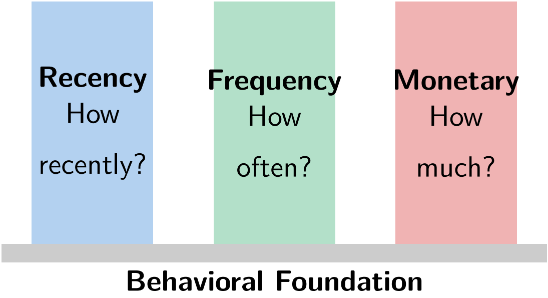
Donor	3m Total	12m Total	Interpretation
101	€150	€500	Slowing down (30% of annual in recent quarter)
102	€0	€400	Went quiet recently! **(!)**
103	€300	€350	Accelerating! (86% of annual in last 3m)

The magic: Comparing windows reveals **momentum**

Section 11

Part 2: RFM - The Holy Trinity

RFM: The three pillars of behavioral prediction



Why RFM works: These three capture fundamentally different aspects of engagement

Concept: Time since last donation predicts next donation

```r
# Calculate days since last gift
recency <- gifts %>%
  filter(date < reference_date) %>%
  group_by(donor_id) %>%
  summarise(last_gift_date = max(date)) %>%
  mutate(
    days_since = as.numeric(reference_date - last_gift_date),

    # Create meaningful categories
    recency_segment = case_when(
      days_since <= 30  ~ "Active",      # Gave last month
      days_since <= 90  ~ "Warm",        # Gave this quarter
      days_since <= 365 ~ "Cooling",     # Gave this year
      TRUE ~ "Cold"                      # Over a year ago
    )
  )
```

Concept: Past frequency predicts future frequency

```
# How often do they give?
frequency <- gifts %>%
  filter(date >= reference_date - months(12),
         date < reference_date) %>%
  group_by(donor_id) %>%
  summarise(
    gift_count = n(),
    unique_months = n_distinct(floor_date(date, "month")),

    # Calculate regularity
    regularity = unique_months / 12  # Score 0-1
  ) %>%
  mutate(
    frequency_segment = case_when(
      gift_count >= 12 ~ "Monthly",      # Every month
      gift_count >= 4  ~ "Regular",      # Quarterly
      gift_count >= 2  ~ "Occasional"    # Semi-annual
```

```r
# How much do they give?
monetary <- gifts %>%
  filter(date >= reference_date - months(12),
         date < reference_date) %>%
  group_by(donor_id) %>%
  summarise(
    total_value = sum(amount),
    avg_gift = mean(amount),
    max_gift = max(amount),

    # Variability matters too!
    cv = sd(amount) / mean(amount)  # Coefficient of variatio
  ) %>%
  mutate(
    value_tier = case_when(
      total_value >= 1000 ~ "Major",      # €1000+
      total_value >= 500  ~ "Premium",    # €500-1000
      total_value >= 100  ~ "Standard",   # €100-500
```

```r
# Join all three components
rfm <- basetable %>%
  left_join(recency %>% select(donor_id, days_since),
            by = "donor_id") %>%
  left_join(frequency %>% select(donor_id, gift_count),
            by = "donor_id") %>%
  left_join(monetary %>% select(donor_id, total_value),
            by = "donor_id") %>%
  mutate(
    # Score each dimension 1-5 (5 = best)
    R_score = ntile(-days_since, 5),   # Negative: recent = h
    F_score = ntile(gift_count, 5),
    M_score = ntile(total_value, 5),

    # Combined RFM code (e.g., "555" = best)
    RFM_segment = paste0(R_score, F_score, M_score)
  )
```

Segment	R	F	M	Label	Strategy
555	5	5	5	Champions	Cultivate & thank
511	5	1	1	New Enthusiasts	Nurture relationship
155	1	5	5	At Risk	Win-back campaign **(!)**
111	1	1	1	Lost	Don't waste resources

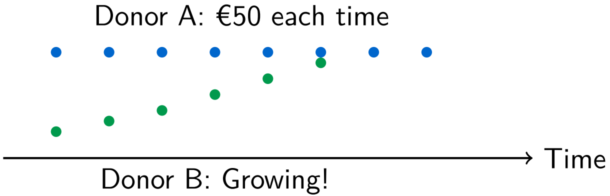
Marketing insight: Segment 155 (At Risk) $\rightarrow$ immediate intervention!

Cost savings: Don't mail Segment 111 $\rightarrow$ save 40% of mailing costs

Section 12

Part 3: The Power of Trends

Problem with snapshots: They miss the movie



Both gave €300 total → but very different futures!

Solution: Calculate **change rates** and **trends**

```r
# Step 1: Define comparison periods
recent_3m <- gifts %>%
  filter(date >= reference_date - months(3),
         date < reference_date)

previous_3m <- gifts %>%
  filter(date >= reference_date - months(6),
         date < reference_date - months(3))

# Step 2: Aggregate each period
recent_agg <- recent_3m %>%
  group_by(donor_id) %>%
  summarise(recent_total = sum(amount))

previous_agg <- previous_3m %>%
  group_by(donor_id) %>%
  summarise(previous_total = sum(amount))
```

Donor	Previous	Recent	Change	Signal
Alice	€100	€200	+100%	$\nearrow$ Accelerating
Bob	€200	€150	-25%	$\searrow$ Declining
Carol	€0	€50	New!	$\star$ Emerging

Actionable insights:

- **Alice:** Ready for upgrade ask (€300?)
- **Bob:** Investigate decline (contact them!)
- **Carol:** Welcome series (nurture new behavior)

```r
# Create interpretable trend categories
trends <- trends %>%
  mutate(
    trend_category = case_when(
      previous_total == 0 & recent_total > 0 ~ "New Active",
      percent_change > 0.25 ~ "Strong Growth",
      percent_change > 0 ~ "Modest Growth",
      percent_change > -0.25 ~ "Slight Decline",
      percent_change > -0.5 ~ "Moderate Decline",
      TRUE ~ "Sharp Decline"
    ),

    # Binary flag for action
    needs_attention = percent_change < -0.25
  )
```

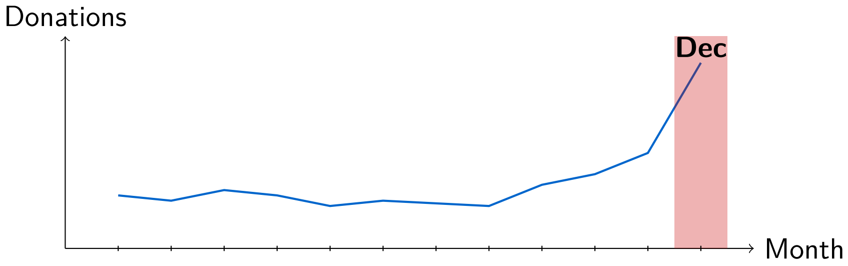
Why categories? Easier for business users to understand and act on

Pro tip: Create `needs_attention` flags → automatic alerts to fundraising

Section 13

Part 4: Seasonality Matters

Real phenomenon: Donations spike in December (end-of-year tax planning)



Problem: December total isn't comparable to July total!

Solution: Calculate **seasonal indices**

```r
# Extract historical seasonal patterns (exclude recent year)
seasonal_history <- gifts %>%
  filter(date < reference_date - years(1),
         date >= reference_date - years(3)) %>%
  mutate(month = month(date)) %>%
  group_by(donor_id, month) %>%
  summarise(avg_monthly = mean(amount), .groups = "drop")

# Calculate index: ratio to annual average
seasonal_indices <- seasonal_history %>%
  group_by(donor_id) %>%
  mutate(
    annual_avg = mean(avg_monthly),
    seasonal_index = avg_monthly / annual_avg
  )

# Extract current month's index
current_month <- month(reference_date)
```

```r
# Add seasonal adjustment to features
basetable <- basetable %>%
  left_join(donor_seasonality, by = "donor_id") %>%
  mutate(
    # If missing seasonality data, assume neutral (1.0)
    seasonal_index = replace_na(seasonal_index, 1.0),

    # Adjust recent donations for fair comparison
    donations_3m_adjusted = donations_3m / seasonal_index,

    # Compare adjusted values to annual average
    performance_vs_seasonal = donations_3m_adjusted /
      (donations_12m / 4)  # Quarterly average
  )
```

Business value: "Bob gave €300 in July (low season, index=0.8) → adjusted = €375 → Actually performing well!"

Section 14

Part 5: Feature Interactions

Concept: Features are more powerful combined than alone

Example: Recency × Frequency interaction

```r
# Create interaction terms
basetable <- basetable %>%
  mutate(
    # Recency-Frequency: Recent × Frequent = highly engaged
    RF_interaction = (1 / (days_since + 1)) * gift_count,

    # Frequency-Monetary: High frequency + High value = premi
    FM_interaction = gift_count * avg_gift,

    # Trend-Level: Growing + Large = invest more attention
    trend_strength = abs(percent_change) * total_value
  )
```

Donor	Days Since	Count	RF Score	Interpretation
A	10	12	1.09	Recent & frequent = **Best!** $\star$
B	10	2	0.18	Recent but infrequent
C	365	12	0.03	Frequent but not recent **(!)**
D	365	2	0.005	Neither recent nor frequent

Key insight: Donor C looks good on frequency alone, but RF interaction reveals the problem!

Model benefit: Interaction terms help models learn these nuances automatically

```r
# Create ratio-based features
basetable <- basetable %>%
  mutate(
    # Evolution: is recent behavior above or below average?
    ratio_3m_to_12m = donations_3m / (donations_12m + 0.01),

    # Concentration: does one big gift dominate?
    max_to_total_ratio = max_gift / (total_value + 0.01),

    # Consistency: how variable are gift sizes?
    consistency_score = 1 - (cv_donation / 2),  # Scaled 0-1

    # Lifetime value rate
    lifetime_intensity = total_value /
      as.numeric(reference_date - member_since) * 365
  )
```

Why ratios? They're **scale-invariant** → work for small and large donors

Section 15

Part 6: Handling Missing Values

Type 1: "No Data" → Donor joined after the window started

Type 2: "No Activity" → Donor didn't give during the window

```
# Smart imputation strategy
basetable <- basetable %>%
  mutate(
    # Flag the reason for missingness
    is_new_donor = as.numeric(reference_date - member_since) <

    # Different imputation by reason
    donations_12m = case_when(
      is_new_donor & is.na(donations_12m) ~ NA_real_,  # Keep
      is.na(donations_12m) ~ 0,                         # Zero
      TRUE ~ donations_12m
    ),

    # For ratios, handle zero denominators
    ratio_3m_to_12m = case_when(
```

```r
# Create "missingness flags" as features
basetable <- basetable %>%
  mutate(
    # Flag no recent activity
    flag_inactive_3m = as.integer(donations_3m == 0),
    flag_inactive_12m = as.integer(donations_12m == 0),

    # Flag new donor status
    flag_new_donor = as.integer(is_new_donor),

    # Flag data quality issues
    flag_incomplete_history = as.integer(
      as.numeric(reference_date - member_since) < 365 &
      !is_new_donor
    )
  )
```

Why flags? They're features themselves! "No activity" is predictive.

Section 16

Part 7: Feature Binning

Why bin? Sometimes categories capture non-linear relationships better

```r
# Create bins using quantiles (equal population)
basetable <- basetable %>%
  mutate(
    # Donation frequency bins
    freq_bin = cut(
      gift_count,
      breaks = quantile(gift_count, probs = seq(0, 1, 0.25),
                        na.rm = TRUE),
      labels = c("Q1-Low", "Q2", "Q3", "Q4-High"),
      include.lowest = TRUE
    ),

    # Recency bins (business-defined)
    recency_bin = cut(
      days_since,
      breaks = c(0, 30, 90, 180, 365, Inf),
      labels = c("0-30d", "31-90d", "3-6m", "6-12m", "12m+"),
```

Quantile bins: Equal population in each bin

- Pro: Handles outliers well
- Con: Bin boundaries change over time

Fixed bins: Domain-knowledge boundaries

- Pro: Stable, interpretable
- Con: May have very unequal populations

Example decision: Use fixed bins for **recency** (business naturally thinks in months), quantiles for **monetary value** (wide range)

Section 17

Part 8: Putting It All Together

```r
# Final feature engineering pipeline
create_features <- function(gifts, basetable, reference_date)

  # 1. RFM features
  rfm_features <- calculate_rfm(gifts, reference_date)

  # 2. Trend features
  trend_features <- calculate_trends(gifts, reference_date)

  # 3. Seasonal adjustments
  seasonal_features <- calculate_seasonality(gifts, reference_

  # 4. Interaction terms
  basetable <- basetable %>%
    left_join(rfm_features, by = "donor_id") %>%
    left_join(trend_features, by = "donor_id") %>%
    left_join(seasonal_features, by = "donor_id") %>%
    mutate(
```

Before moving to modeling, verify:

- ☐ All features use **only past data** (before reference date)
- ☐ Missing values handled **appropriately** (not arbitrarily)
- ☐ Outliers **capped or winsorized** (if needed)
- ☐ Categorical variables **encoded** (if using tree models)
- ☐ Feature names are **clear and documented**
- ☐ Temporal **stability checked** (do features exist at all time points?)

Red flag: Feature has 50% missing values $\rightarrow$ investigate before using!

Green light: Feature has clear business meaning and predictive logic

Section 18

Part 9: Feature Selection

Problem: 100+ features → some are redundant or noisy

```r
# Method 1: Correlation filtering
library(caret)

# Remove highly correlated features
feature_matrix <- basetable %>%
  select(where(is.numeric)) %>%
  select(-donor_id)

cor_matrix <- cor(feature_matrix, use = "complete.obs")
high_cor <- findCorrelation(cor_matrix, cutoff = 0.90)

features_to_drop <- names(feature_matrix)[high_cor]
```

Why? If two features are 95% correlated, we only need one!

Example: `donations_12m` and `avg_gift * count_12m` → redundant

```r
# Method 2: Random Forest importance
library(randomForest)

# Fit initial model
rf_model <- randomForest(
  target ~ .,
  data = basetable %>% select(-donor_id),
  importance = TRUE,
  ntree = 100
)

# Extract importance scores
importance_df <- importance(rf_model) %>%
  as.data.frame() %>%
  rownames_to_column("feature") %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(20)  # Keep top 20
```

Rank	Feature	Importance	Interpretation
1	days_since	245	Recency dominates!
2	donations_12m	189	Total value matters
3	RF_interaction	156	Interaction helps
4	trend_category	134	Momentum signal
5	gift_count	98	Frequency counts
6	ratio_3m_to_12m	87	Recent behavior
7	seasonal_index	72	Context matters
8	max_gift	65	Capacity indicator
9	cv_donation	54	Consistency signal
10	lifetime_days	48	Tenure relevant

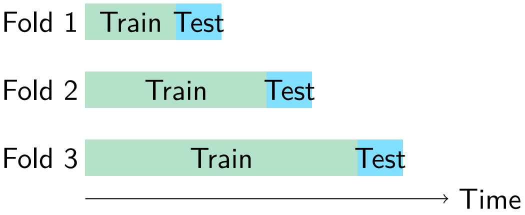
Surprise: Demographics (age, gender) ranked 25+!

Section 19

Part 10: Validation Strategy

Problem: Random CV violates timeline!

Solution: Walk forward through time



Key: Test set always **after** training set → realistic evaluation

```r
# Create temporal splits
walk_forward_splits <- function(data, n_splits = 3) {
  n_obs <- nrow(data)
  fold_size <- floor(n_obs / (n_splits + 1))

  splits <- list()

  for(i in 1:n_splits) {
    train_idx <- 1:(fold_size * i)
    test_idx <- (fold_size * i + 1):(fold_size * (i + 1))

    splits[[i]] <- list(
      train = data[train_idx, ],
      test = data[test_idx, ]
    )
  }

  return(splits)
```

```r
# Train and evaluate on each fold
library(pROC)

cv_results <- map_df(1:length(splits), function(i) {
  # Train model
  model <- glm(
    target ~ days_since + donations_12m + RF_score,
    data = splits[[i]]$train,
    family = "binomial"
  )

  # Predict on test set
  predictions <- predict(model, splits[[i]]$test, type = "resp

  # Calculate AUC
  auc_value <- auc(roc(splits[[i]]$test$target, predictions))

  tibble(
```

Section 20

Part 11: Feature Documentation

Problem: 6 months later, “What does `var_x47` mean?”

Solution: Document everything!

```r
# Create feature catalog
feature_catalog <- tibble(
  feature_name = c(
    "donations_12m",
    "RF_interaction",
    "ratio_3m_to_12m"
  ),
  description = c(
    "Total donation value in 12 months before reference date",
    "Interaction: recency × frequency for engagement score",
    "Proportion of annual donations made in recent quarter"
  ),
  calculation = c(
    "sum(amount) WHERE date IN [ref-12m, ref)",
    "(1 / (days_since + 1)) * gift_count",
```

Feature	Type	Window	Business Meaning
days_since	Numeric	Point-in-time	Days since last donation (recency)
donations_12m	Numeric	12 months	Total annual contribution
RF_score	Numeric	Derived	Combined engagement metric
trend_category	Categorical	3m vs 3m	Direction of behavior change

Pro tip: Export as CSV, share with business stakeholders

Bonus: Helps detect errors (e.g., "Wait, this calculation doesn't match reality!")

Section 21

Part 12: Production Considerations

Development: Code runs once on historical data

Production: Code runs repeatedly on new data

```
# Parameterized feature engineering
engineer_features <- function(reference_date,
                              gifts_data,
                              basetable_data) {

  # Use parameters, not hardcoded dates!
  window_3m_start <- reference_date - months(3)
  window_12m_start <- reference_date - months(12)

  # Filter data
  recent_gifts <- gifts_data %>%
    filter(date >= window_3m_start, date < reference_date)

  # Calculate features
  features <- calculate_all_features(
```

```r
# Unit tests catch bugs early
library(testthat)

test_that("Features respect timeline", {
  # Create test data
  test_gifts <- tibble(
    donor_id = 1,
    date = as.Date(c("2023-06-01", "2024-01-15")),
    amount = c(100, 50)
  )

  ref_date <- as.Date("2024-01-01")

  # Run feature engineering
  features <- engineer_features(ref_date, test_gifts, basetabl

  # Assert: Only June gift should count
  expect_equal(features$donations_12m[features$donor_id == 1],
```

Track feature drift:

```r
# Compare distributions over time
monitor_features <- function(new_data, baseline_data) {

  features_to_monitor <- c("donations_12m", "days_since", "RF_

  drift_report <- map_df(features_to_monitor, function(feat) {
    # KS test for distribution change
    ks_result <- ks.test(
      baseline_data[[feat]],
      new_data[[feat]]
    )

    tibble(
      feature = feat,
      ks_statistic = ks_result$statistic,
      p_value = ks_result$p.value,
      drift_detected = ks_result$p.value < 0.05
```

Section 22

Summary: Feature Engineering Principles

1. **Timeline Compliance:** Never use future data
2. **Multiple Windows:** Short-term and long-term perspectives
3. **RFM Always:** Recency, Frequency, Monetary are foundational
4. **Capture Trends:** Change matters more than level
5. **Context Matters:** Seasonality and life stage
6. **Interactions:** $1 + 1$ can equal 3
7. **Handle Missing:** Distinguish "no data" from "no activity"
8. **Document Everything:** Future-you will thank present-you
9. **Validate Temporally:** Walk forward, don't shuffle
10. **Monitor Production:** Features drift, models decay

Next Steps:

1. **Feature Selection:** Keep top 20-30 features
2. **Model Training:** Logistic regression → Random Forest → Gradient Boosting
3. **Hyperparameter Tuning:** Grid search with CV
4. **Model Evaluation:** AUC, calibration, business metrics
5. **Deployment:** API for scoring new donors
6. **Monitoring:** Track performance decay

Remember:

Better features > Fancier models

Spend 80% of time on feature engineering, 20% on model selection!

Organization: International humanitarian NGO

Challenge: Retain monthly donors (50% churned within 1 year)

Solution: Built features tracking:

- RFM scores
- Donation trends (3m vs 12m)
- Seasonal patterns
- Email engagement $\times$ donation frequency

Results:

- **AUC:** 0.58 $\rightarrow$ 0.74 (massive improvement!)
- **Business impact:** Identified 8% of donors representing 40% of churn risk
- **Intervention:** Personalized outreach $\rightarrow$ 15% churn reduction
- **ROI:** €450K saved in first year

Key insight: Trend features (growth/decline) were most predictive!

Books:

- *Feature Engineering for Machine Learning* by Zheng & Casari
- *Feature Engineering and Selection* by Kuhn & Johnson

Online:

- Kaggle: "Feature Engineering" courses
- Towards Data Science: Time series feature engineering

R Packages:

- `recipes`: Feature engineering pipeline
- `timetk`: Time series features
- `caret`: Feature selection

Key Principle: Domain knowledge + creativity + validation = great features

Dataset: Provided donor transaction data

Task: Create these features:

1. RFM scores (R, F, M separate)
2. Trend: 3-month change rate
3. Ratio: Recent/historical comparison
4. Interaction: RF combined score
5. Flag: New donor indicator

Deliverable: Documented feature catalog

Evaluation: Do features predict donation in next month?

Hint: Start simple, validate early, iterate!

Key Takeaways:

Features are **stories** about donor behavior

Timeline compliance is non-negotiable

RFM + **Trends** + **Context** = powerful predictions

Document and **validate** everything

Production requires robust pipelines

Section 23

Part 1: Creating Dummy Variables

Machine learning models need numbers, not categories!

Consider this donor dataset:

donor_id	gender	country	segment
5	F	India	Gold
3	M	USA	Silver
2	M	India	Bronze
8	F	UK	Silver
1	F	USA	Bronze

Question: How do we include gender, country, and segment in a logistic regression model?

Answer: Convert categories to binary dummy variables (one-hot encoding)

Transform each category into a binary indicator:

donor_id	gender	gender_F	gender_M
5	F	1	0
3	M	0	1
2	M	0	1
8	F	1	0
1	F	1	0

Interpretation: Each dummy variable answers a yes/no question.
`gender_F = 1` means "Is this donor female? Yes."

Issue: If we know `gender_F`, we automatically know `gender_M`!

When `gender_F = 0`, then `gender_M` must equal 1. This creates **perfect multicollinearity**, which breaks regression models.

Mathematical problem: $\text{gender\_F} + \text{gender\_M} = 1$ (always)

This linear dependence means the design matrix is not full rank, causing estimation failure.

Solution: Drop one category to serve as the reference level.

Keep only k-1 dummy variables for a categorical variable with k categories:

donor_id	gender	gender_F
5	F	1
3	M	0
2	M	0
8	F	1
1	F	1

Interpretation: `gender_F = 0` implicitly means Male (reference category). The coefficient on `gender_F` represents the difference between Female and Male donors.

For a variable with 3 categories, create 2 dummy variables:

donor_id	country	country_USA	country_India
5	India	0	1
3	USA	1	0
2	India	0	1
8	UK	0	0
1	USA	1	0

Reference category: UK (when both dummies equal 0).

Model interpretation: Coefficients measure effects relative to UK.

```r
# Method 1: Using fastDummies package
library(fastDummies)

# Create dummies, drop first category
basetable <- dummy_cols(
  basetable,
  select_columns = "segment",
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)

# Method 2: Using model.matrix (automatic)
dummies <- model.matrix(~ segment - 1, data = basetable)

# Method 3: Manual approach for single variable
basetable$segment_Gold <- as.integer(basetable$segment == "Gol
basetable$segment_Silver <- as.integer(basetable$segment == "S
# Bronze becomes reference (both dummies = 0)
```

```r
# Original data
head(basetable[, c("donor_id", "segment")])
#   donor_id segment
# 1    32770    Gold
# 2    32776  Silver
# 3    32777  Bronze
# 4    65552  Bronze

# Create dummy variables
basetable <- dummy_cols(
  basetable,
  select_columns = "segment",
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)

# Result: Bronze is reference (omitted)
head(basetable[, c("donor_id", "segment_Gold", "segment_Silver
```

Section 24

Part 2: Handling Missing Values

Models cannot handle NA values! Most algorithms will fail or silently drop observations.

```
# Example: Missing age values
basetable[c("donor_id", "age")]
#   donor_id age
# 1        5  NA
# 2        3  25
# 3        2  36
# 4        8  40
# 5        1  26
```

Critical questions:

1. Why is the value missing? (Random? Systematic?)
2. How many values are missing? (1%? 50%?)
3. What's the best replacement strategy?

When to use: Continuous variables, few missing values, roughly normal distribution.

```r
# Calculate mean age (excluding NA)
mean_age <- mean(basetable$age, na.rm = TRUE)  # 31.75

# Replace missing values
basetable$age[is.na(basetable$age)] <- mean_age

# Result
#   donor_id age
# 1        5  31.75  # Was NA
# 2        3  25.00
# 3        2  36.00
# 4        8  40.00
# 5        1  26.00
```

Advantage: Simple, maintains overall mean.

When to use: Skewed distributions or presence of outliers.

```r
# Example with outlier in max_donation
basetable$max_donation
# 100, 1000000, 100, 40, 120

# Mean heavily influenced by outlier
mean(basetable$max_donation, na.rm = TRUE)    # 200,052
median(basetable$max_donation, na.rm = TRUE)  # 110

# Use median for robustness
replacement <- median(basetable$max_donation, na.rm = TRUE)
basetable$max_donation[is.na(basetable$max_donation)] <- repla

# Result: Missing value now 110 (much more reasonable!)
```

Why median? The €1,000,000 donor skews the mean but doesn't affect the median.

When to use: Missing means "absence of activity" or "zero count."

```r
# Example: donations_last_year
# NA means they didn't donate (zero donations)
basetable[c("donor_id", "donations_last_year")]
#   donor_id donations_last_year
# 1        5                 130
# 2        3                  10
# 3        2                  NA  # No donations
# 4        8                  40
# 5        1                 120

# Replace NA with 0 (no donations)
basetable$donations_last_year[
  is.na(basetable$donations_last_year)
] <- 0

# Result: NA becomes 0 (meaningful zero)
```

```r
# Flexible function for different strategies
replace_missing <- function(x, method = "mean") {
  if (method == "mean") {
    replacement <- mean(x, na.rm = TRUE)
  } else if (method == "median") {
    replacement <- median(x, na.rm = TRUE)
  } else if (method == "zero") {
    replacement <- 0
  } else {
    stop("Method must be 'mean', 'median', or 'zero'")
  }

  x[is.na(x)] <- replacement
  return(x)
}

# Usage
basetable$age <- replace_missing(basetable$age, method = "mean
```

Concept: Sometimes "missingness" itself is informative!

```
# Example: email address
basetable[c("donor_id", "email")]
#   donor_id email
# 1    32770 person32770@provider.com
# 2    32776 NA
# 3    32777 person32777@provider.com
# 4    65552 NA

# Create indicator: 1 if missing, 0 if present
basetable$no_email <- as.integer(is.na(basetable$email))

# Result
#   donor_id email                         no_email
# 1    32770 person32770@provider.com             0
# 2    32776 NA                                   1
# 3    32777 person32777@provider.com             0
# 4    65552 NA                                   1
```

Use cases where missingness is predictive:

1. **Missing email** → Lower tech-savviness, privacy concerns
2. **Missing income** → Reluctance to share financial info
3. **Missing phone number** → Reduced contactability
4. **Missing survey responses** → Lower engagement

```r
# Create multiple missingness indicators
basetable$missing_email <- as.integer(is.na(basetable$email))
basetable$missing_phone <- as.integer(is.na(basetable$phone))
basetable$missing_income <- as.integer(is.na(basetable$income)

# These can be powerful predictors in the model!
# Example: Donors without email might be 30% less likely to do
```

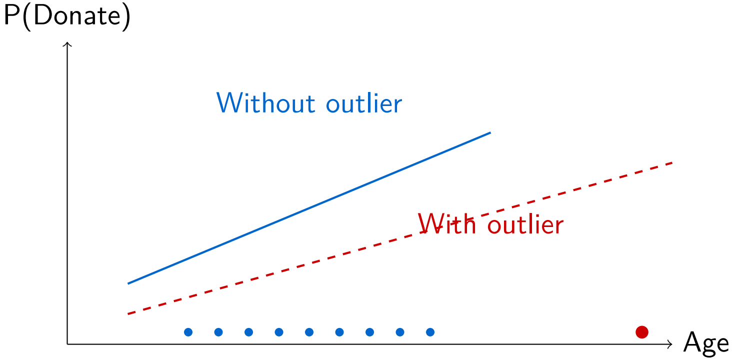
Best practice: Create indicator BEFORE imputing the actual variable.

Section 25

Part 3: Handling Outliers

Outliers distort model coefficients and predictions.

Consider this visualization:



Impact: One extreme age value pulls the entire regression line!

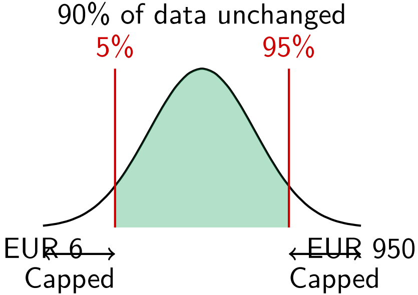
Before removing outliers, understand why they exist:

1. **Data entry errors** $\rightarrow$ Age = 999 (should be 99)
2. **Measurement errors** $\rightarrow$ Equipment malfunction
3. **Truly extreme values** $\rightarrow$ Billionaire donor (real, but rare)
4. **Different population** $\rightarrow$ Corporate donor in individual database

Decision tree:

- Error? $\rightarrow$ Correct or remove
- Measurement issue? $\rightarrow$ Remove
- True extreme? $\rightarrow$ Cap (winsorize) or transform
- Different population? $\rightarrow$ Separate analysis or remove

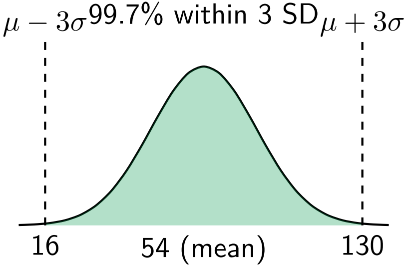
Concept: Cap extreme values at specified percentiles (typically 5th and 95th).



Result: Values below 5th percentile set to 5th percentile value. Values above 95th percentile set to 95th percentile value.

```r
# Using DescTools package
library(DescTools)

# Winsorize at 5% and 95% percentiles
basetable$mean_donation_winsorized <- Winsorize(
  basetable$mean_donation,
  probs = c(0.05, 0.95)
)

# Manual implementation
winsorize_manual <- function(x, lower = 0.05, upper = 0.95) {
  # Calculate percentile thresholds
  lower_limit <- quantile(x, lower, na.rm = TRUE)
  upper_limit <- quantile(x, upper, na.rm = TRUE)

  # Cap values
  x[x < lower_limit] <- lower_limit
  x[x > upper_limit] <- upper_limit
```

Concept: Cap values beyond mean $\pm$ 3 standard deviations.



Why 3 SD? In a normal distribution, 99.7% of data falls within this range. Values beyond are statistically extreme.

```r
# Calculate boundaries
mean_age <- mean(basetable$age, na.rm = TRUE)
sd_age <- sd(basetable$age, na.rm = TRUE)

lower_limit <- mean_age - 3 * sd_age
upper_limit <- mean_age + 3 * sd_age

# Apply capping
basetable$age_no_outliers <- pmin(
  pmax(basetable$age, lower_limit),  # Cap below
  upper_limit                        # Cap above
)

# Alternative using ifelse
basetable$age_no_outliers <- ifelse(
  basetable$age < lower_limit, lower_limit,
  ifelse(basetable$age > upper_limit, upper_limit,
         basetable$age)
)
```

Comparison:

Method	When to Use	Advantage	Disadvantage
Winsorization	Skewed data, many outliers	Preserves distribution shape	Arbitrary percentile choice
Standard Deviation	Normal data, few outliers	Statistically principled	Assumes normality

Example decision:

- **Age variable** (roughly normal) → Use SD method
- **Donation amounts** (heavily skewed) → Use winsorization

```
# Check distribution before choosing
hist(basetable$age)           # Normal? Use SD method
hist(basetable$mean_donation) # Skewed? Use winsorization

# Formal test for normality
```

Section 26

Part 4: Transformations

Problem: Large differences in scale distort model predictions.

Example: Four donors with very different donation amounts:

- **Alice:** €100 (modest donor)
- **Bob:** €1,100 (10$\times$ more than Alice)
- **Carol:** €10,000 (100$\times$ more than Alice)
- **Dave:** €11,000 (only 10% more than Carol)

Issue: The difference between Alice and Bob (€1,000) seems similar to the difference between Carol and Dave (€1,000), but the relative importance is very different!

Solution: Transform to capture relative rather than absolute differences.

Logarithm converts multiplication into addition:

- Alice: €100 $\rightarrow$ log(100) $=$ 4.6
- Bob: €1,100 $\rightarrow$ log(1,100) $=$ 7.0
- Carol: €10,000 $\rightarrow$ log(10,000) $=$ 9.2
- Dave: €11,000 $\rightarrow$ log(11,000) $=$ 9.3

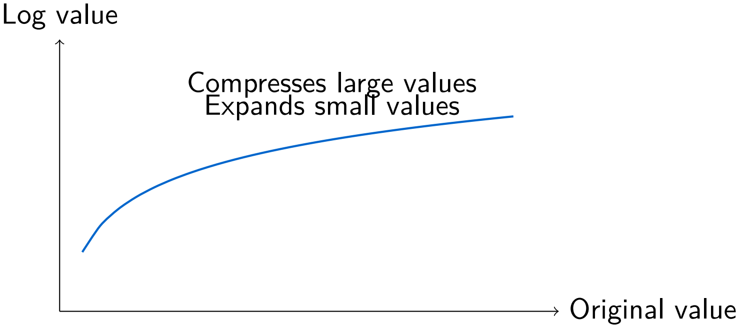
Key insight: The gap between Alice and Bob (7.0 - 4.6 $=$ 2.4) is now comparable to the gap between Carol and Dave (9.3 - 9.2 $=$ 0.1). This better reflects the relative differences in giving capacity!

Mathematical properties:

$$\log(a \times b) = \log(a) + \log(b)$$

This means percentage changes become additive:

- 10% increase: $\log(1.1 \times x) = \log(x) + 0.095$
- Same for all values of x!



```r
# Apply natural logarithm
basetable$log_mean_donation <- log(basetable$mean_donation)

# Handle zeros (log(0) is undefined!)
# Add small constant before logging
basetable$log_mean_donation <- log(basetable$mean_donation + 1

# Alternative: log1p function (more numerically stable)
basetable$log_mean_donation <- log1p(basetable$mean_donation)

# Compare distributions
par(mfrow = c(1, 2))
hist(basetable$mean_donation, main = "Original",
     xlab = "Mean Donation")
hist(basetable$log_mean_donation, main = "Log-transformed",
     xlab = "Log(Mean Donation)")
```

When to use: Monetary values, counts, ratios, or any right-skewed

Square root transformation (milder than log):

```r
basetable$sqrt_donations <- sqrt(basetable$mean_donation)
```

Inverse transformation (for extreme skew):

```r
basetable$inv_recency <- 1 / (basetable$days_since_last + 1)
# Recent donors get high values, old donors get low values
```

Box-Cox transformation (automatically finds best power):

```r
library(car)
bc <- powerTransform(basetable$mean_donation)
basetable$bc_donation <- bcPower(basetable$mean_donation,
                                  bc$lambda)
```

Section 27

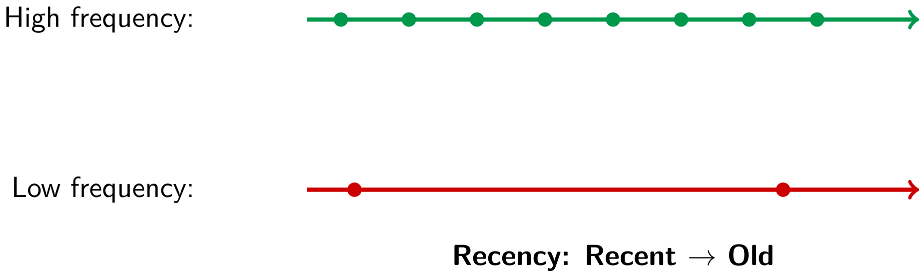
Part 5: Interaction Features

Problem: Features may have combined effects that are greater than the sum of their parts.

Example: Donor engagement depends on BOTH frequency and recency:

- **High frequency + Recent donation** $\rightarrow$ Very likely to donate
- **High frequency + Old donation** $\rightarrow$ Moderate likelihood
- **Low frequency + Recent donation** $\rightarrow$ Moderate likelihood
- **Low frequency + Old donation** $\rightarrow$ Very unlikely to donate

The effect of frequency depends on recency (and vice versa)!



Key insight: The combination of high frequency AND recent donation creates the strongest prediction signal.

```r
# Multiplicative interaction
basetable$freq_recency_interaction <-
  basetable$donation_count * basetable$days_since_last

# Interpretation: Higher values = active donors
# Recent + frequent donors get high scores
# Old + infrequent donors get low scores

# Alternative: Inverse recency for intuition
basetable$freq_recency_interaction <-
  basetable$donation_count / (basetable$days_since_last + 1)

# Now higher values = more engaged
# Recent frequent donors: 12 / 10 = 1.2
# Old infrequent donors: 2 / 365 = 0.005

# Multiple interactions
basetable$rfm_interaction <-
```

```r
# Create comprehensive engagement score
compute_rfm_score <- function(recency, frequency, monetary) {
  # Normalize recency (inverse so recent = high)
  r_score <- 1 / (recency + 1)

  # Normalize frequency (already in right direction)
  f_score <- frequency

  # Normalize monetary
  m_score <- monetary

  # Combined multiplicative score
  rfm_score <- r_score * f_score * m_score

  return(rfm_score)
}

# Apply to all donors
```

```r
# Create interactions between all numeric variables
numeric_vars <- c("days_since_last", "donation_count",
                  "mean_donation", "max_donation")

# All pairwise interactions
for (i in 1:(length(numeric_vars)-1)) {
  for (j in (i+1):length(numeric_vars)) {
    var1 <- numeric_vars[i]
    var2 <- numeric_vars[j]

    interaction_name <- paste0(var1, "_x_", var2)
    basetable[[interaction_name]] <- basetable[[var1]] *
                                     basetable[[var2]]
  }
}

# Result: Creates
# days_since_last_x_donation_count
```

Section 28

Part 6: Complete Preprocessing Pipeline

```r
# Step 1: Handle missing values
basetable$age[is.na(basetable$age)] <-
  median(basetable$age, na.rm = TRUE)

basetable$donations_last_year[is.na(basetable$donations_last_y

basetable$missing_email <- as.integer(is.na(basetable$email))

# Step 2: Handle outliers
basetable$mean_donation_capped <- cap_outliers(
  basetable$mean_donation,
  n_sd = 3
)

# Step 3: Create dummy variables
basetable <- dummy_cols(
  basetable,
  select_columns = c("segment", "country"),
```

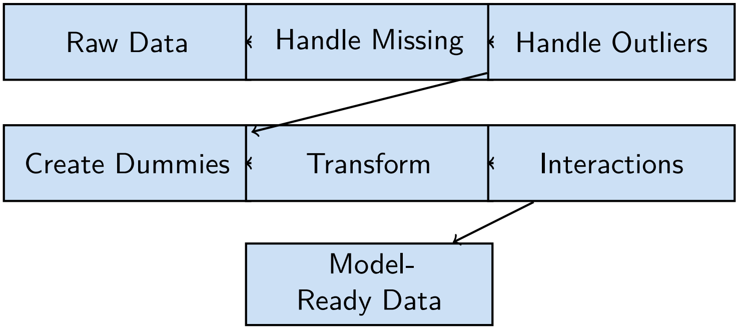
Before modeling, verify:

- Missing values handled for ALL variables
- Outliers addressed in key variables
- Categorical variables converted to dummies
- Skewed variables transformed
- Meaningful interactions created
- No infinite or undefined values (check log(0))
- Feature names are clear and documented
- Original variables retained (for interpretation)

```
# Final data quality check
summary(basetable)  # Check for NA, Inf
sapply(basetable, function(x) sum(is.na(x)))  # Count NAs
sapply(basetable, function(x) sum(is.infinite(x)))  # Count I
```

Good practice: Use clear, consistent names.

```r
# Bad naming
basetable$var1 <- ...
basetable$x_new <- ...
basetable$temp2 <- ...

# Good naming
basetable$age_median_imputed <- ...
basetable$mean_donation_log <- ...
basetable$rfm_interaction_score <- ...

# Pattern: [variable]_[transformation]_[method]
# Examples:
# - age_winsorized_95
# - donation_log_plus1
# - recency_inverse
# - freq_rec_interaction
# - segment_Gold_dummy
```



Each step prepares data for better model performance!

Mistake 1: Forgetting to drop one dummy category

- Creates multicollinearity
- Model fails to converge

Mistake 2: Taking log of zero

- Returns -Inf
- Use `log1p()` or add small constant

Mistake 3: Not documenting imputation choices

- Can't reproduce results
- Don't remember why you chose median vs. mean

Mistake 4: Creating too many interactions

- 10 variables $\rightarrow$ 45 pairwise interactions!
- Use domain knowledge to select meaningful ones

Mistake 5: Transforming the target variable

- Makes interpretation difficult

With preprocessed features, you can now:

1. **Split data** into training and testing sets
2. **Build models** using clean, numeric features
3. **Evaluate performance** without data quality issues
4. **Interpret results** using meaningful transformations

Coming up next:

- Model selection and training
- Cross-validation strategies
- Performance evaluation metrics
- Model interpretation and deployment

Questions?

Section 29

Part 7: Scaling and Normalization

Problem: Different features have vastly different scales.

Consider a logistic regression model predicting donations:

Feature	Range	Coefficient	Scaled Impact
Age	18-90 years	0.02	$0.02 \times 72 = 1.44$
Income	€20,000-€200,000	0.00001	$0.00001 \times 180{,}000 =$
Days since last gift	1-3,650 days	0.001	$0.001 \times 3{,}649 = 3.65$

Issue: The coefficient magnitudes don't reflect true importance!

Solution: Standardize all features to comparable scales.

Concept: Transform values to range [0, 1].

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
# Min-max scaling function
min_max_scale <- function(x) {
  (x - min(x, na.rm = TRUE)) /
    (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}

# Apply to age variable
basetable$age_scaled <- min_max_scale(basetable$age)

# Result: All values between 0 and 1
# Original: 18, 25, 45, 90
# Scaled:   0.00, 0.10, 0.38, 1.00
```

Advantage: Bounded output, preserves zero

```r
# Example data
age_original <- c(18, 25, 35, 45, 60, 90)
age_scaled <- (age_original - min(age_original)) /
              (max(age_original) - min(age_original))

example_data <- tibble(
  donor_id = 1:6,
  age_original = age_original,
  age_scaled = round(age_scaled, 3)
)

kable(example_data)
```

donor_id	age_original	age_scaled
1	18	0.000
2	25	0.097
3	35	0.236
4	45	0.375

Concept: Transform to mean = 0, standard deviation = 1.

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

```r
# Standardization function
standardize <- function(x) {
  (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
}

# Apply to donation amounts
basetable$mean_donation_std <- standardize(basetable$mean_dona

# Result: Values centered at 0
# Original: 50, 100, 150, 200
# Std:      -1.16, -0.39, 0.39, 1.16
```

Advantage: Not bounded, interpretable units.

Disadvantage: No fixed range

```r
# Example data
donation <- c(50, 100, 150, 200, 250, 1000)
donation_std <- (donation - mean(donation)) / sd(donation)

std_example <- tibble(
  donor_id = 1:6,
  donation_original = donation,
  donation_std = round(donation_std, 2)
)

kable(std_example)
```

donor_id	donation_original	donation_std
1	50	-0.68
2	100	-0.54
3	150	-0.40
4	200	-0.26
5	250	-0.12

Use Min-Max Scaling when:

- Need bounded output [0, 1]
- Working with neural networks
- Features have known, fixed ranges
- Preserving exact zero is important

Use Standardization when:

- Working with SVM, logistic regression
- Data contains outliers
- Want interpretable units (standard deviations)
- Using regularization (L1/L2)

Rule of thumb: Standardization is default for most models.

```r
# Identify numeric columns to scale
numeric_cols <- c("age", "mean_donation", "donation_count",
                  "days_since_last")

# Method 1: Using base R scale()
basetable[paste0(numeric_cols, "_std")] <-
  scale(basetable[numeric_cols])

# Method 2: Using dplyr
basetable <- basetable %>%
  mutate(across(
    all_of(numeric_cols),
    ~scale(.)[,1],
    .names = "{.col}_std"
  ))

# Method 3: Using recipes package
library(recipes)
```

Problem: Must use SAME scaling parameters for test data.

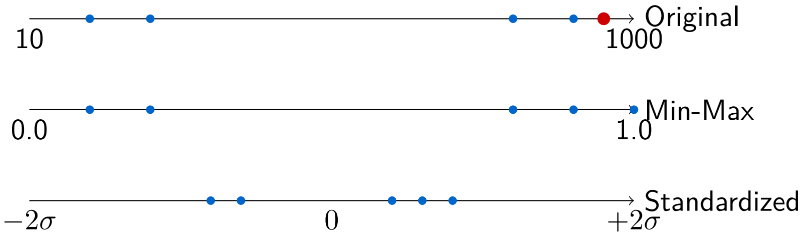
```r
# WRONG: Scale test data independently
test_scaled_wrong <- scale(test_data)  # Uses test mean/sd!

# RIGHT: Use training parameters
train_mean <- mean(train_data$age, na.rm = TRUE)
train_sd <- sd(train_data$age, na.rm = TRUE)

test_data$age_std <- (test_data$age - train_mean) / train_sd

# Better: Save parameters explicitly
scaling_params <- list(
  age_mean = mean(basetable$age, na.rm = TRUE),
  age_sd = sd(basetable$age, na.rm = TRUE),
  donation_mean = mean(basetable$mean_donation, na.rm = TRUE),
  donation_sd = sd(basetable$mean_donation, na.rm = TRUE)
)
```

Problem: Standard scaling uses mean/sd (sensitive to outliers).

Solution: Use median and IQR instead.

$$x_{\text{robust}} = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

```r
# Robust scaling function
robust_scale <- function(x) {
  med <- median(x, na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  (x - med) / iqr
}

# Apply to donation amounts
basetable$mean_donation_robust <- robust_scale(
  basetable$mean_donation
)
```


Original
10
1000
Min-Max
0.0
1.0
Standardized
-2σ
0
$+2\sigma$

Section 30

Part 8: Feature Engineering Best Practices

Before finalizing your feature set:

1. **Timeline compliance**
   - ☐ All features use only past data
   - ☐ No data leakage from target period
2. **Data quality**
   - ☐ Missing values handled appropriately
   - ☐ Outliers addressed or documented
   - ☐ No infinite or undefined values
3. **Encoding**
   - ☐ Categorical variables converted to dummies
   - ☐ Reference categories documented
4. **Transformations**
   - ☐ Skewed variables transformed
   - ☐ Variables scaled appropriately

5. **Feature creation**
   - ☐ Interaction terms included
   - ☐ Domain-specific features created
   - ☐ Temporal patterns captured
6. **Documentation**
   - ☐ Feature catalog created
   - ☐ Calculation formulas documented
   - ☐ Scaling parameters saved
7. **Validation**
   - ☐ Distributions examined
   - ☐ Correlations checked
   - ☐ Feature importance assessed

Mistake 1: Data Leakage

```r
# WRONG: Using target period data
basetable$donations_total <- sum_all_donations()  # Includes

# RIGHT: Use only historical data
basetable$donations_historical <- sum_donations_before(obs_dat
```

Mistake 2: Forgetting to Save Parameters

```r
# WRONG: No way to reproduce scaling
train_scaled <- scale(train_data)

# RIGHT: Save and reuse parameters
scaling_params <- list(mean = mean(train_data$age),
                       sd = sd(train_data$age))
saveRDS(scaling_params, "params.rds")
```

Mistake 3: Overfitting on Training Data

Anti-pattern 1: "Let's add everything!"

- Creates overfitting
- Makes model uninterpretable
- Increases computational cost

Anti-pattern 2: "Let's drop everything with missing values!"

- Loses valuable information
- Reduces sample size unnecessarily
- May introduce bias

Anti-pattern 3: "Let's use the same features for every problem!"

- Ignores domain specifics
- Misses important signals
- Reduces model performance

```r
# 1. Correlation with target
correlations <- cor(basetable[numeric_cols],
                    basetable$target,
                    use = "complete.obs")
print(sort(abs(correlations), decreasing = TRUE))

# 2. Information value (IV)
library(Information)
IV <- create_infotables(data = basetable,
                        y = "target",
                        bins = 10)
print(IV$Summary)  # IV > 0.1 is useful

# 3. Variance inflation factor (VIF)
library(car)
vif_values <- vif(lm(target ~ ., data = basetable))
print(vif_values[vif_values > 5])  # VIF > 5 = collinearity
```

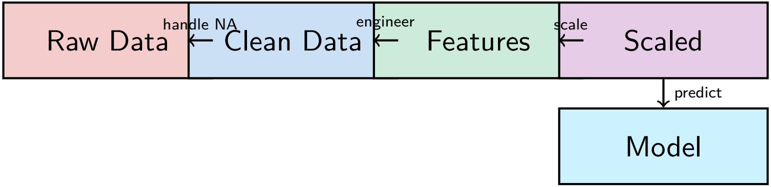
```r
library(corrplot)

# Select numeric features
numeric_features <- basetable %>%
  select(where(is.numeric), -donor_id)

# Compute correlation matrix
cor_matrix <- cor(numeric_features, use = "complete.obs")

# Visualize
corrplot(cor_matrix,
         method = "color",
         type = "upper",
         tl.col = "black",
         tl.srt = 45,
         addCoef.col = "black",
         number.cex = 0.7)
```

Section 31

Part 9: Production Pipeline

Development: Ad-hoc feature engineering in notebook

Production: Reproducible, maintainable pipeline



```r
engineer_features <- function(data, reference_date, params = N
  # Step 1: Handle missing values
  data <- data %>%
    mutate(
      age = replace_na(age, median(age, na.rm = TRUE)),
      donations_last_year = replace_na(donations_last_year, 0)
      missing_email = as.integer(is.na(email))
    )

  # Step 2: Create time-based features
  data <- data %>%
    mutate(
      days_since_last = as.numeric(reference_date - last_donat
    )

  # Step 3: Create RFM features
  data <- data %>%
    mutate(
```

```r
# Save configuration
feature_config <- list(
  reference_date = as.Date("2024-01-01"),
  missing_strategy = list(
    age = "median",
    donations_last_year = "zero",
    email = "indicator"
  ),
  outlier_method = "winsorize",
  outlier_probs = c(0.05, 0.95),
  scaling_method = "standardize",
  dummy_variables = c("segment", "country"),
  interaction_terms = list(
    c("donation_count", "days_since_last"),
    c("mean_donation", "donation_count")
  )
)
```

```r
library(recipes)

# Define preprocessing recipe
preprocessing_recipe <- recipe(target ~ ., data = basetable) %

  # Step 1: Remove ID variables
  step_rm(donor_id) %>%

  # Step 2: Create dummy variables
  step_dummy(all_nominal(), -all_outcomes(),
             one_hot = FALSE) %>%

  # Step 3: Impute missing values
  step_impute_median(all_numeric(), -all_outcomes()) %>%

  # Step 4: Remove zero variance features
  step_zv(all_predictors()) %>%
```

```r
# Prepare recipe on training data
prepped_recipe <- prep(preprocessing_recipe,
                       training = train_data)

# Apply to training data
train_processed <- bake(prepped_recipe,
                        new_data = train_data)

# Apply to test data (uses training parameters!)
test_processed <- bake(prepped_recipe,
                       new_data = test_data)

# Apply to new scoring data
new_data_processed <- bake(prepped_recipe,
                           new_data = new_donors)

# Save recipe for production
saveRDS(prepped_recipe, "preprocessing_recipe.rds")
```

```r
library(testthat)

test_that("Timeline compliance is maintained", {
  # Create test data with known dates
  test_data <- tibble(
    donor_id = 1,
    donation_date = as.Date(c("2023-01-01", "2024-06-01")),
    amount = c(100, 50)
  )

  ref_date <- as.Date("2024-01-01")

  # Process features
  features <- engineer_features(test_data, ref_date)

  # Assert: Only Jan 2023 donation should count
  expect_equal(features$donations_historical[1], 100)
  expect_equal(features$donation_count[1], 1)
```

```r
monitor_drift <- function(new_data, baseline_data) {

  numeric_features <- names(baseline_data)[
    sapply(baseline_data, is.numeric)
  ]

  drift_report <- map_df(numeric_features, function(feat) {

    # Kolmogorov-Smirnov test
    ks_result <- ks.test(baseline_data[[feat]],
                         new_data[[feat]])

    # Calculate distribution statistics
    baseline_mean <- mean(baseline_data[[feat]], na.rm = TRUE
    current_mean <- mean(new_data[[feat]], na.rm = TRUE)

    tibble(
      feature = feat,
```

```r
# Load baseline (training) data
baseline <- readRDS("training_data_jan2024.rds")

# New data from production
new_batch <- readRDS("production_data_nov2024.rds")

# Check for drift
drift_analysis <- monitor_drift(new_batch, baseline)

# Flag significant drifts
drift_analysis %>%
  filter(drift_detected == TRUE) %>%
  arrange(ks_pvalue) %>%
  select(feature, ks_statistic, ks_pvalue, mean_change_pct)
```

Section 32

Part 10: Advanced Topics

```r
# Method 1: Correlation-based
cor_with_target <- cor(basetable[numeric_cols],
                        basetable$target)
top_features <- names(sort(abs(cor_with_target),
                           decreasing = TRUE)[1:20])

# Method 2: Chi-square for categorical
library(FSelector)
chi_scores <- chi.squared(target ~ ., data = basetable)
top_categorical <- cutoff.k(chi_scores, k = 10)
```

```r
library(caret)

# Define control
ctrl <- rfeControl(
  functions = rfFuncs,
  method = "cv",
  number = 5
)

# Run RFE
rfe_results <- rfe(
  x = basetable %>% select(-target, -donor_id),
  y = basetable$target,
  sizes = c(5, 10, 15, 20, 25),
  rfeControl = ctrl
)

# Optimal features
```

```r
library(glmnet)

# Prepare matrix
X <- model.matrix(target ~ . - donor_id, data = basetable)[,-1
y <- basetable$target

# Fit LASSO
lasso_model <- cv.glmnet(X, y,
                         family = "binomial",
                         alpha = 1,
                         nfolds = 5)

# Extract non-zero coefficients
lasso_coefs <- coef(lasso_model, s = "lambda.min")
selected_features <- rownames(lasso_coefs)[lasso_coefs[,1] !=

print(selected_features)
```

```r
# Create polynomial features
basetable <- basetable %>%
  mutate(
    # Quadratic terms
    age_squared = age^2,
    donation_squared = mean_donation^2,

    # Cubic terms
    age_cubed = age^3,

    # Square root
    age_sqrt = sqrt(age),

    # Ratio
    age_to_donation_ratio = age / (mean_donation + 1)
  )

# Automated polynomial creation
```

```r
# Extract temporal patterns
basetable <- basetable %>%
  mutate(
    # Day of week
    donation_day_of_week = wday(last_donation_date),

    # Month
    donation_month = month(last_donation_date),

    # Quarter
    donation_quarter = quarter(last_donation_date),

    # Is weekend?
    is_weekend = wday(last_donation_date) %in% c(1, 7),

    # Is December?
    is_december = month(last_donation_date) == 12,
```

```r
# Calculate donations by month
monthly_donations <- gifts %>%
  mutate(month = floor_date(date, "month")) %>%
  group_by(donor_id, month) %>%
  summarize(monthly_total = sum(amount), .groups = "drop")

# Create lag features
monthly_donations <- monthly_donations %>%
  group_by(donor_id) %>%
  arrange(month) %>%
  mutate(
    donation_lag1 = lag(monthly_total, 1),
    donation_lag2 = lag(monthly_total, 2),
    donation_lag3 = lag(monthly_total, 3),

    # Rolling average
    donation_ma3 = (donation_lag1 + donation_lag2 +
                    donation_lag3) / 3,
```

```r
# Calculate target rate by country
country_encoding <- basetable %>%
  group_by(country) %>%
  summarize(
    target_rate = mean(target, na.rm = TRUE),
    n = n()
  )

# Add smoothing
overall_rate <- mean(basetable$target, na.rm = TRUE)
smoothing_factor <- 10

country_encoding <- country_encoding %>%
  mutate(
    target_rate_smoothed = (target_rate * n +
                           overall_rate * smoothing_factor) /
                           (n + smoothing_factor)
  )
```

```r
library(FeatureHashing)

# Hash country into buckets
hashed_features <- hashed.model.matrix(
  ~ country,
  data = basetable,
  hash.size = 2^10,  # 1024 buckets
  signed.hash = FALSE
)

# Add to basetable
basetable <- cbind(basetable, as.data.frame(hashed_features))
```

Section 33

Part 11: Complete Case Study

Organization: International nonprofit

Goal: Predict €50+ donations in next 3 months

Data: 5 years history, 100,000 donors

Timeline: Build model January 2024

```r
# Observation date
observation_date <- as.Date("2024-01-01")

# Target period
target_start <- observation_date
target_end <- observation_date + months(3)

# Historical window
history_start <- observation_date - years(2)
history_end <- observation_date

# Partition data
gifts_historical <- gifts %>%
  filter(date >= history_start & date < history_end)

gifts_target <- gifts %>%
  filter(date >= target_start & date < target_end)
```

```r
# Eligible donors
donors_with_history <- gifts_historical %>%
  distinct(donor_id)

# Exclude buffer period
buffer_start <- observation_date - months(1)
buffer_donations <- gifts %>%
  filter(date >= buffer_start & date < observation_date) %>%
  distinct(donor_id)

# Final population
population <- setdiff(
  donors_with_history$donor_id,
  buffer_donations$donor_id
)

# Create basetable
basetable <- tibble(donor_id = population)
```

```r
# Aggregate target
target_donations <- gifts_target %>%
  group_by(donor_id) %>%
  summarize(target_amount = sum(amount), .groups = "drop")

# Join and create binary target
basetable <- basetable %>%
  left_join(target_donations, by = "donor_id") %>%
  mutate(
    target_amount = replace_na(target_amount, 0),
    target = as.integer(target_amount >= 50)
  )

# Check distribution
table(basetable$target)
```

```r
# RFM features
rfm_features <- gifts_historical %>%
  group_by(donor_id) %>%
  summarize(
    # Recency
    days_since_last = as.numeric(observation_date - max(date))

    # Frequency
    donation_count = n(),
    unique_months = n_distinct(floor_date(date, "month")),

    # Monetary
    total_donated = sum(amount),
    mean_donation = mean(amount),
    median_donation = median(amount),
    max_donation = max(amount),
    min_donation = min(amount),
    cv_donation = sd(amount) / mean(amount),
```

```r
# Recent vs previous
recent_3m <- gifts_historical %>%
  filter(date >= observation_date - months(3)) %>%
  group_by(donor_id) %>%
  summarize(donations_3m = sum(amount), .groups = "drop")

previous_3m <- gifts_historical %>%
  filter(date >= observation_date - months(6),
         date < observation_date - months(3)) %>%
  group_by(donor_id) %>%
  summarize(donations_prev3m = sum(amount), .groups = "drop")

trend_features <- recent_3m %>%
  full_join(previous_3m, by = "donor_id") %>%
  mutate(
    donations_3m = replace_na(donations_3m, 0),
    donations_prev3m = replace_na(donations_prev3m, 0),
    trend_absolute = donations_3m - donations_prev3m,
```

```r
# Load demographics
demographics <- read_csv("donor_demographics.csv")

# Calculate age
demographics <- demographics %>%
  mutate(
    age = year(observation_date) - year(birth_date),
    tenure_days = as.numeric(observation_date - first_donatio
  )

# Join
basetable <- basetable %>%
  left_join(
    demographics %>% select(donor_id, age, gender,
                            country, tenure_days),
    by = "donor_id"
  )
```

```r
# Handle missing values
basetable <- basetable %>%
  mutate(
    # Impute age
    age = replace_na(age, median(age, na.rm = TRUE)),

    # Zero for donations
    across(starts_with("donations_"),
           ~replace_na(., 0)),

    # Missing indicators
    missing_gender = as.integer(is.na(gender)),
    missing_country = as.integer(is.na(country))
  )

# Handle outliers
basetable <- basetable %>%
  mutate(
```

```r
# Log transformations
basetable <- basetable %>%
  mutate(
    log_mean_donation = log1p(mean_donation_capped),
    log_total_donated = log1p(total_donated),
    log_tenure = log1p(tenure_days)
  )

# Interactions
basetable <- basetable %>%
  mutate(
    rfm_score = (1 / (days_since_last + 1)) *
                donation_count * mean_donation,
    freq_recency = donation_count / (days_since_last + 1),
    trend_strength = abs(trend_percent) * total_donated
  )

# Dummies
```

```r
library(caret)

# Stratified split
set.seed(42)
train_index <- createDataPartition(
  basetable$target,
  p = 0.7,
  list = FALSE
)

train_data <- basetable[train_index, ]
test_data <- basetable[-train_index, ]

# Verify split
table(train_data$target) / nrow(train_data)
table(test_data$target) / nrow(test_data)
```

```r
# Numeric features
numeric_features <- c(
  "days_since_last", "donation_count", "total_donated",
  "mean_donation_capped", "log_mean_donation", "rfm_score",
  "age", "tenure_days", "trend_percent"
)

# Calculate parameters
scaling_params <- train_data %>%
  summarize(across(all_of(numeric_features),
                   list(mean = ~mean(., na.rm = TRUE),
                        sd = ~sd(., na.rm = TRUE))))

saveRDS(scaling_params, "scaling_params.rds")

# Apply to train
train_scaled <- train_data
for (feat in numeric_features) {
```

```r
library(glmnet)

# Prepare matrices
X_train <- model.matrix(target ~ . - donor_id - target_amount,
                        data = train_scaled)[,-1]
y_train <- train_scaled$target

X_test <- model.matrix(target ~ . - donor_id - target_amount,
                       data = test_scaled)[,-1]
y_test <- test_scaled$target

# Train LASSO
cv_model <- cv.glmnet(X_train, y_train,
                      family = "binomial",
                      alpha = 1,
                      nfolds = 5,
                      type.measure = "auc")
```

```r
library(pROC)

# Predictions
test_predictions <- predict(final_model,
                            newx = X_test,
                            type = "response")[,1]

# ROC curve
roc_obj <- roc(y_test, test_predictions)
auc_value <- auc(roc_obj)

print(paste("Test AUC:", round(auc_value, 3)))

# Plot
plot(roc_obj, main = "ROC Curve",
     col = "blue", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")
```

```r
# Extract coefficients
coefficients <- coef(final_model)
coef_df <- data.frame(
  feature = rownames(coefficients),
  coefficient = as.vector(coefficients)
) %>%
  filter(coefficient != 0, feature != "(Intercept)") %>%
  arrange(desc(abs(coefficient)))

# Top 10
head(coef_df, 10)
```

```r
# Save artifacts
saveRDS(final_model, "donor_prediction_model.rds")
saveRDS(scaling_params, "scaling_parameters.rds")

# Scoring function
score_new_donors <- function(new_data,
                             model_file = "donor_prediction_mo
                             scaling_file = "scaling_paramete
  # Load
  model <- readRDS(model_file)
  scaling <- readRDS(scaling_file)

  # Engineer features
  new_data <- engineer_features(new_data, Sys.Date())

  # Scale
  new_data_scaled <- scale_features(new_data, scaling)
```

```r
# Monitor predictions
monitor_predictions <- function(scored_data, baseline_stats) {
  current_mean <- mean(scored_data$prediction)
  current_sd <- sd(scored_data$prediction)

  alert <- FALSE

  # Check distribution shift
  if (abs(current_mean - baseline_stats$mean) > 0.1) {
    warning("Prediction mean shifted!")
    alert <- TRUE
  }

  if (abs(current_sd - baseline_stats$sd) /
      baseline_stats$sd > 0.3) {
    warning("Prediction variance changed >30%!")
    alert <- TRUE
  }
```

Section 34

Summary

Remember

1. **Timeline integrity** is non-negotiable
2. **RFM + Trends** capture 80% of predictive power
3. **Feature quality** > Feature quantity
4. **Document everything**
5. **Test systematically**
6. **Monitor continuously**

Better features make better models!

Books:

- *Feature Engineering for Machine Learning* by Zheng & Casari
- *Feature Engineering and Selection* by Kuhn & Johnson
- *Data Science for Business* by Provost & Fawcett

R Packages:

- `recipes`: Production-ready feature engineering
- `caret`: Machine learning framework
- `tidyverse`: Data manipulation

Topic: Model Selection and Evaluation

- Logistic regression in depth
- Tree-based models
- Cross-validation strategies
- Performance metrics
- Model interpretation

Thank You!

Questions?