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

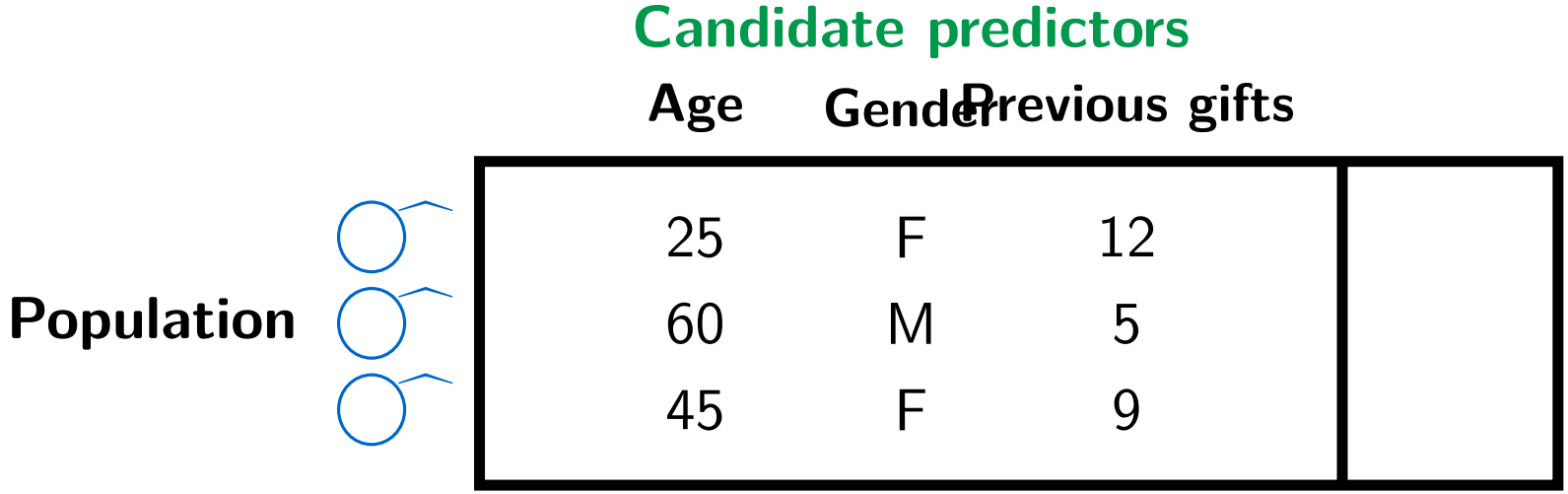
Definition

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)

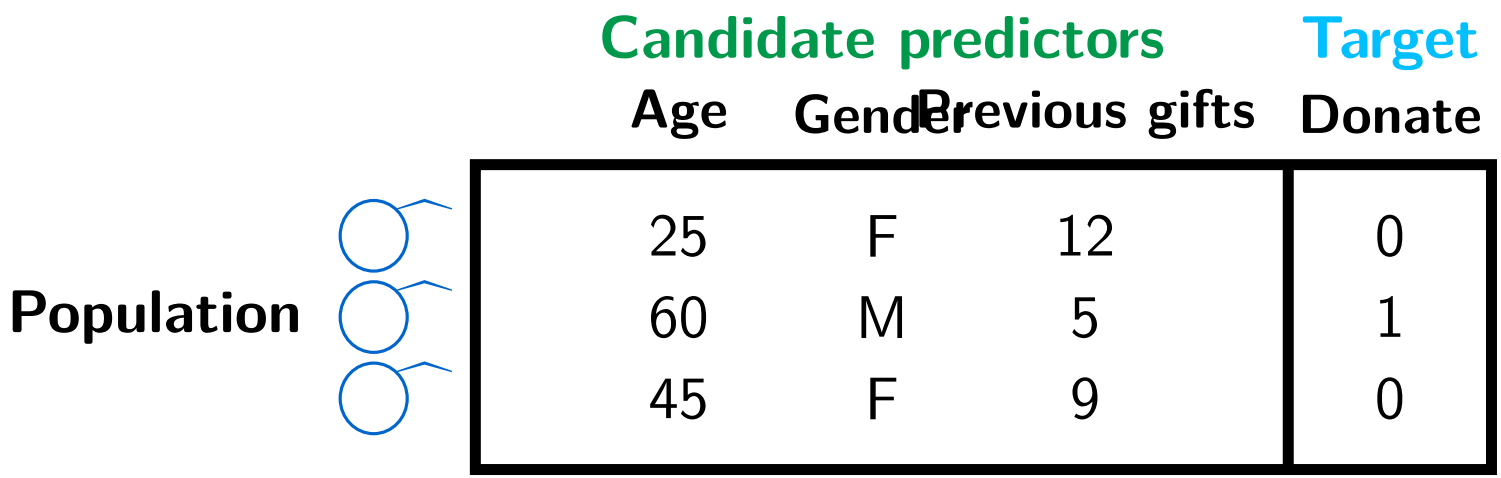
Population

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



Candidate Predictors

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



Target Variable

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

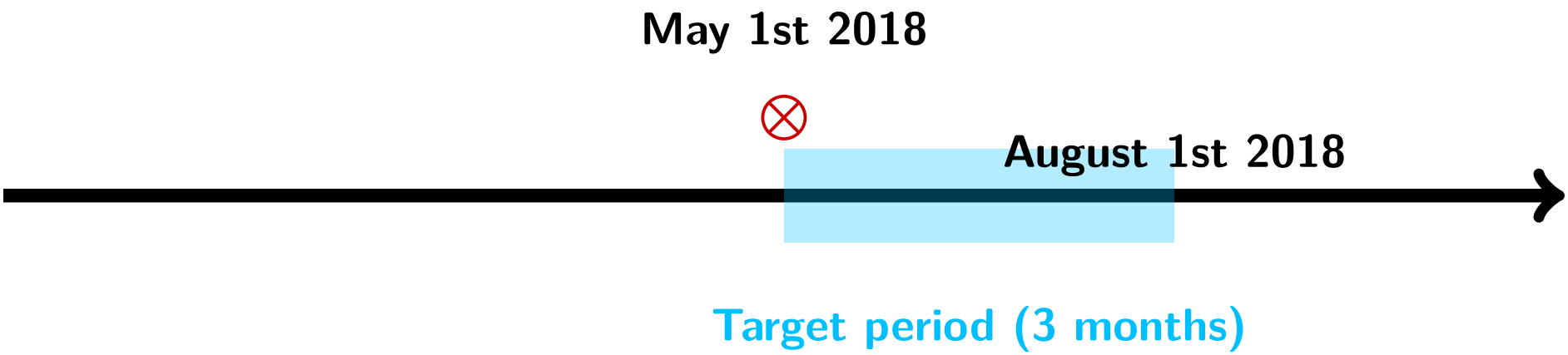
Section 3

The Timeline

Temporal Structure

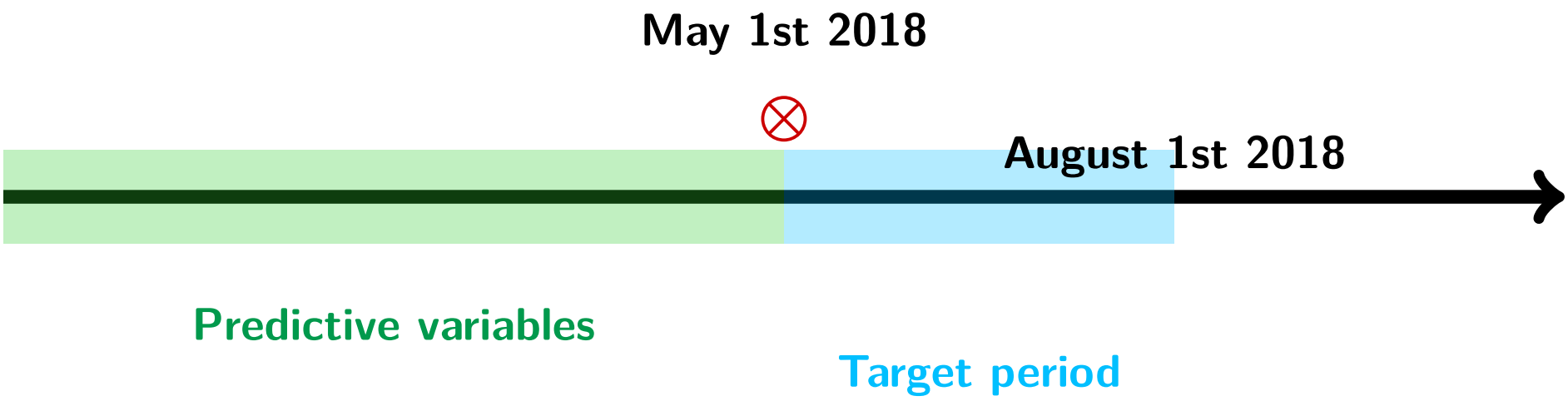
Predictive modeling requires a clear **temporal separation** between:

- **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?

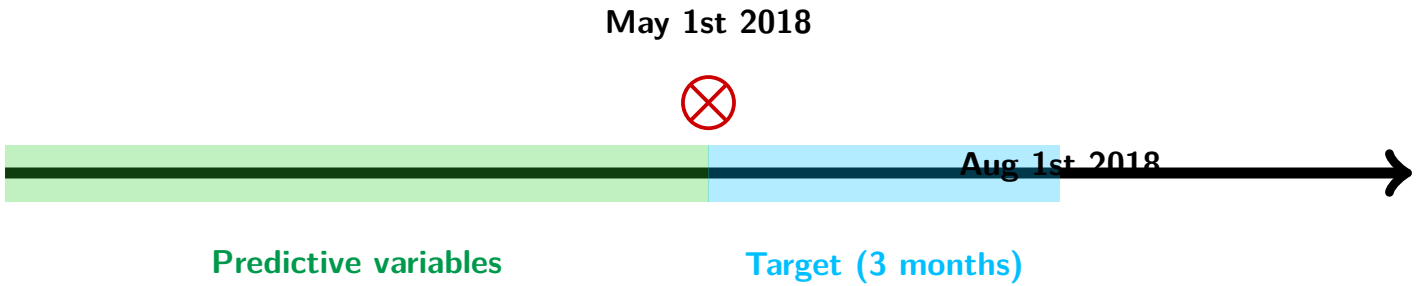
1. **Prevents data leakage**: Ensures predictors don't contain future information
2. **Mimics deployment**: Replicates real-world prediction scenarios
3. **Valid evaluation**: Enables honest assessment of model performance
4. **Temporal validity**: Accounts for time-dependent patterns

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.

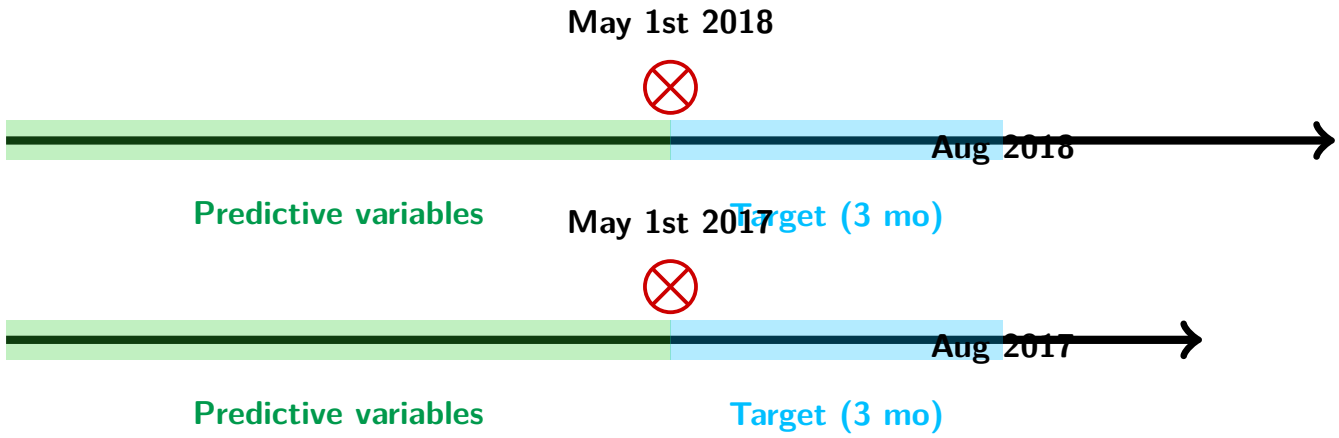
Section 4

Reconstructing History



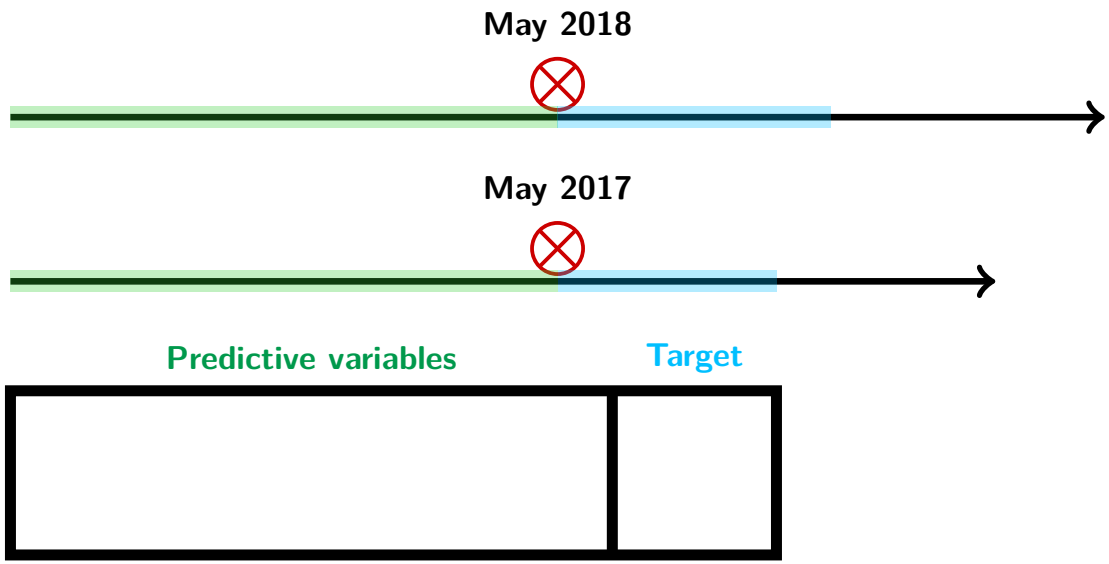
Training Data Construction

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

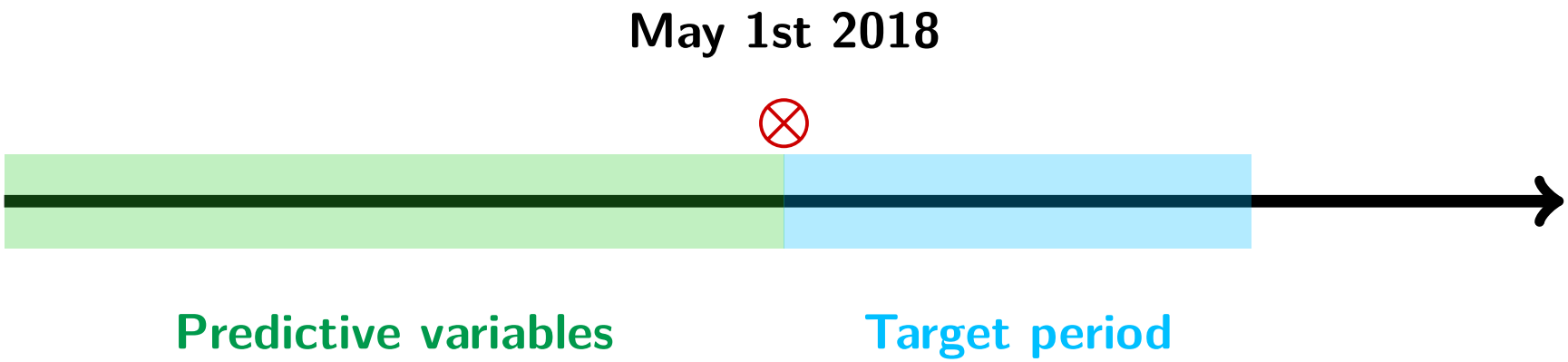
Population = donors who:

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

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

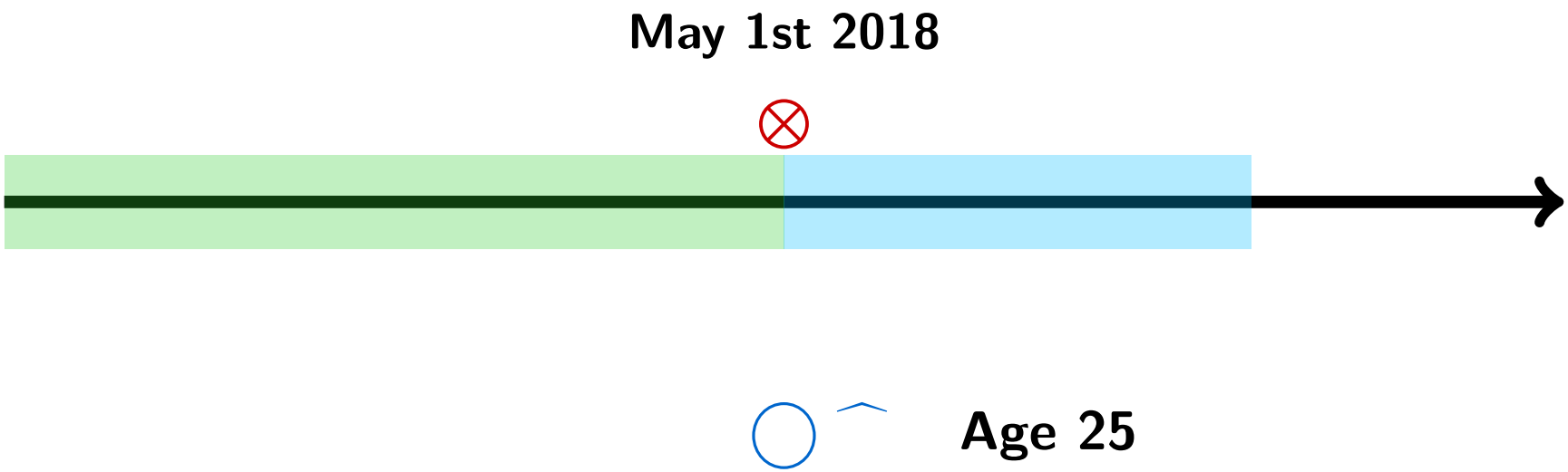
Eligibility Criteria

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



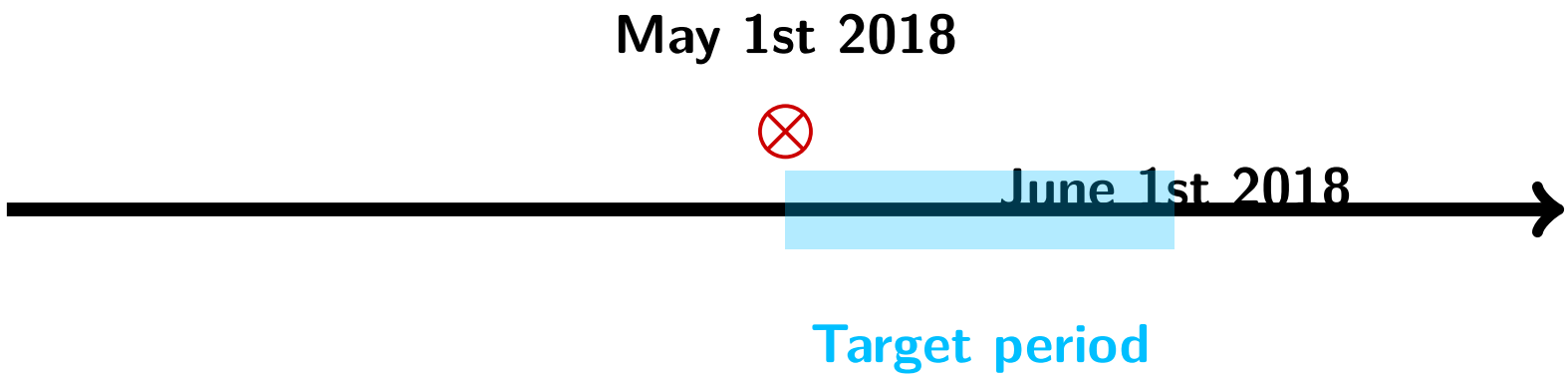
Temporal Consistency

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



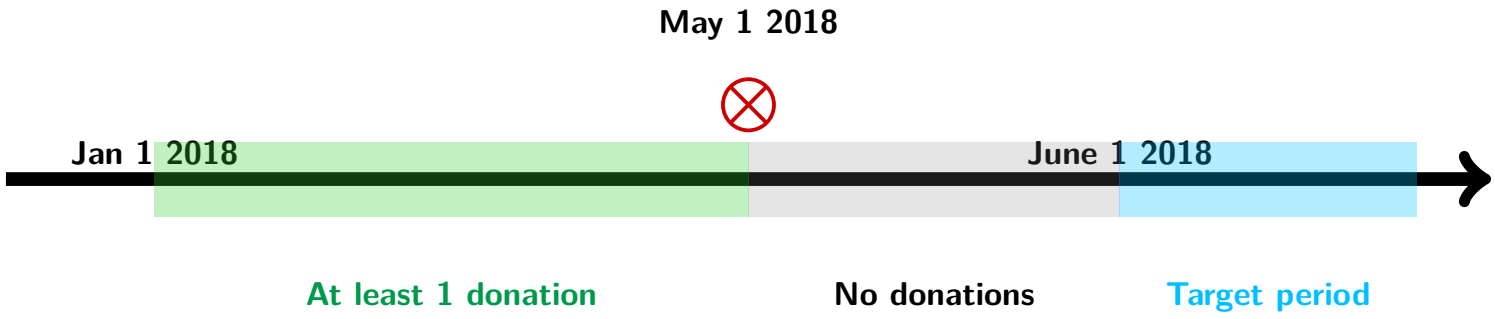
Implementation

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



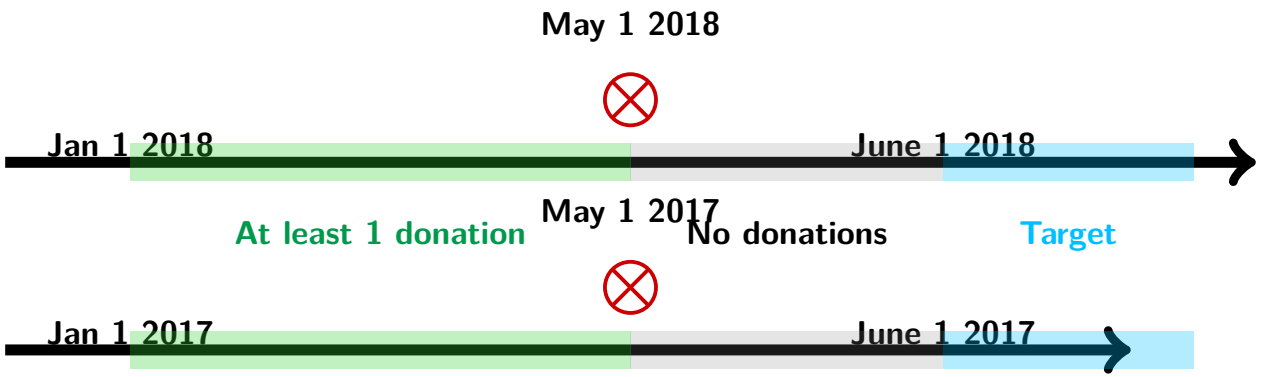
Population Filtering by Donation History

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.

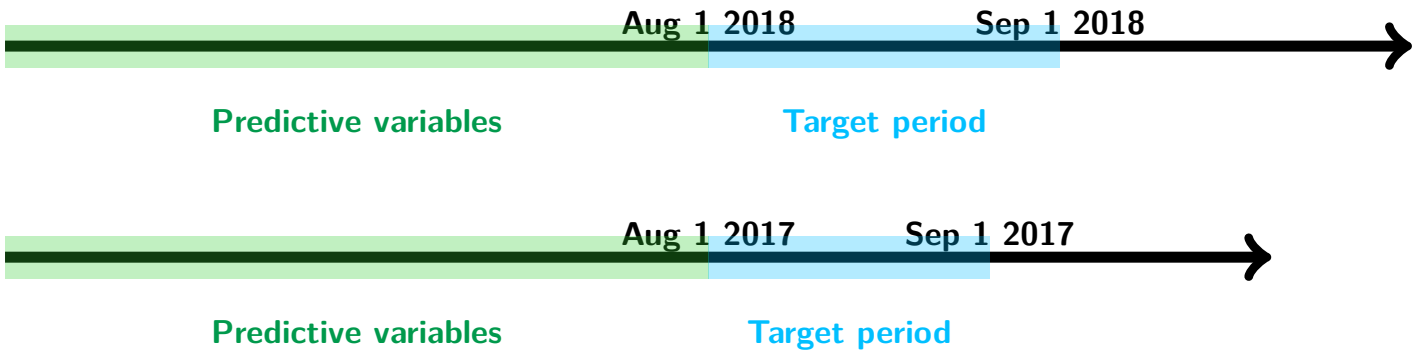
Types of Targets

- **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

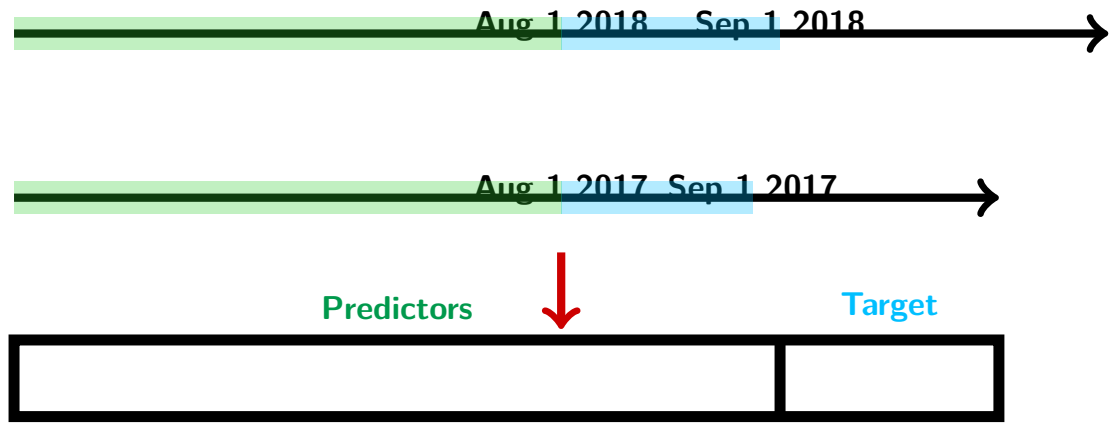
The target period should:

- 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

```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
```

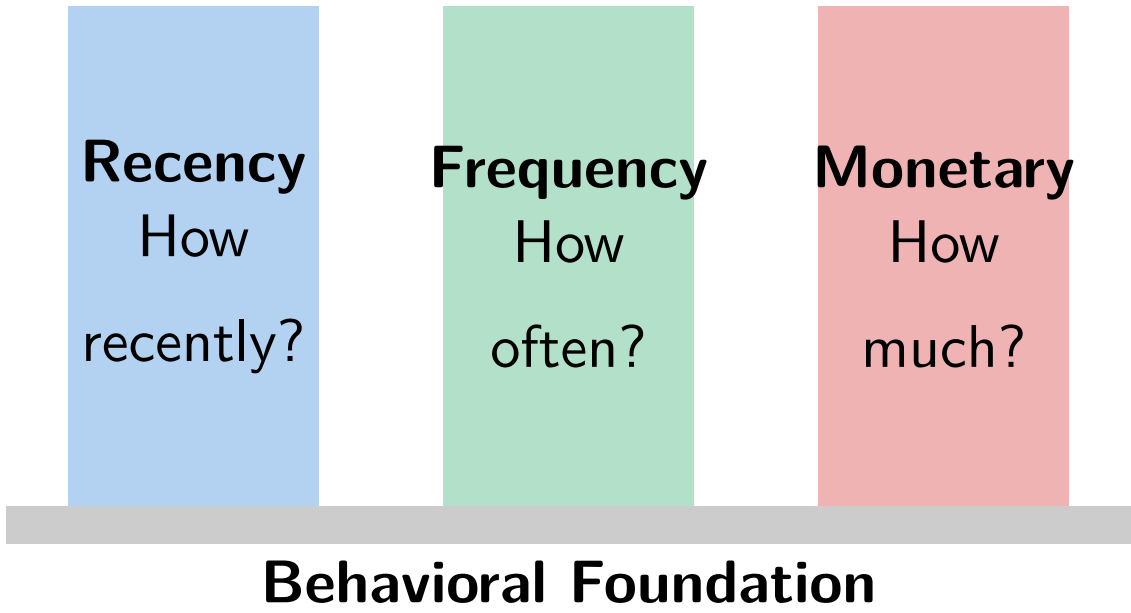
Donor	3m Total	12m Total	Interpretation
101	€150	€500	Slowing down (30% of annual in recent quarter)
102	€0	€400	Went quiet recently! **(!)**
103	€300	€350	Accelerating! (86% of annual in last 3m)

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 variation
  ) %>%
  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

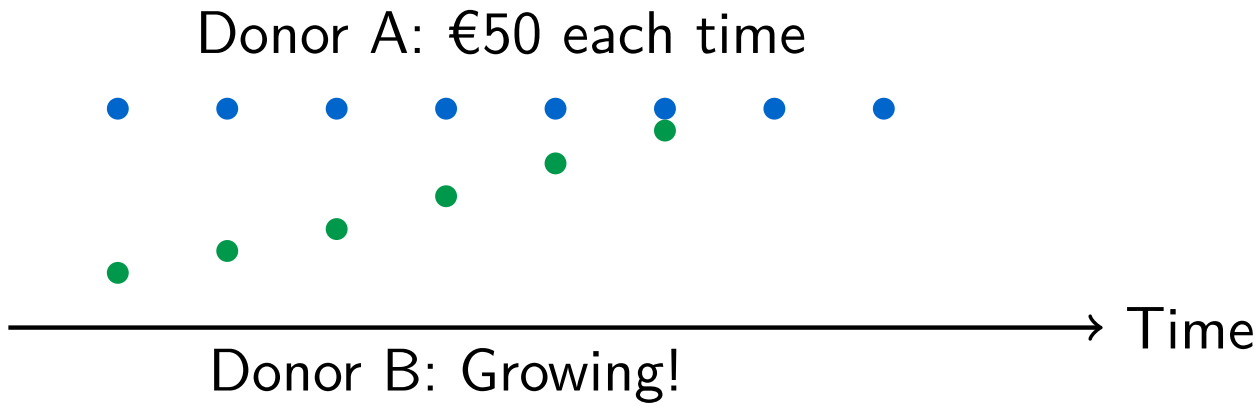
Marketing insight: Segment 155 (At Risk) $\rightarrow$ immediate intervention!

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%	↗ Accelerating
Bob	€200	€150	-25%	↘ Declining
Carol	€0	€50	New!	⋆ 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
  )
```

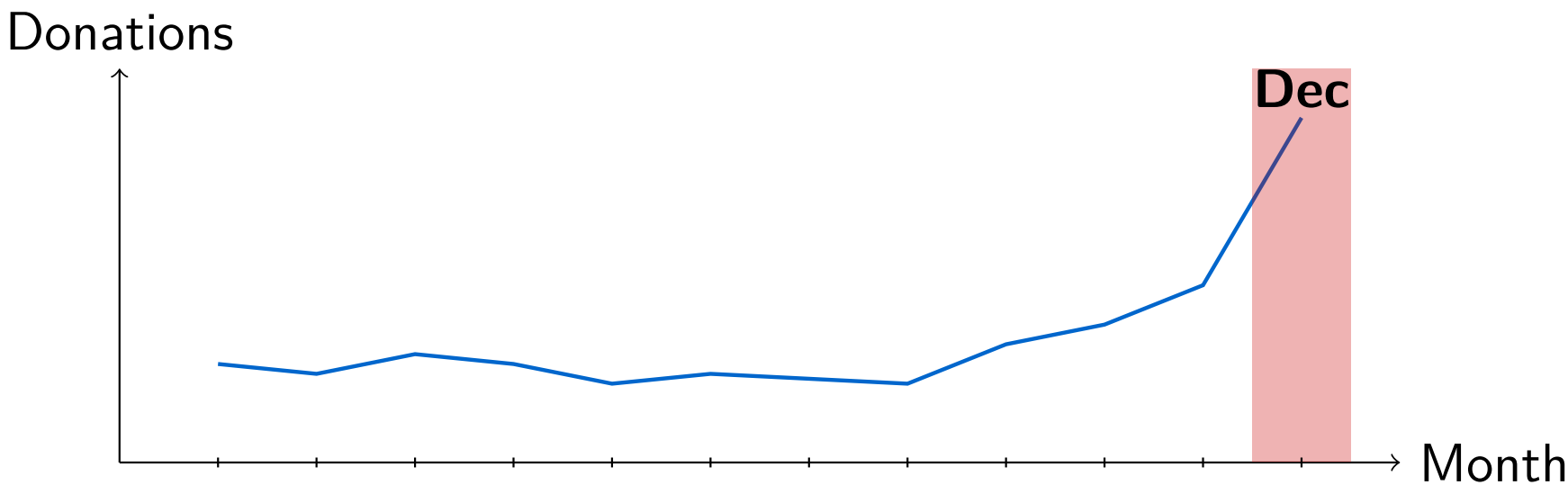
Why categories? Easier for business users to understand and act on

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

```
# 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

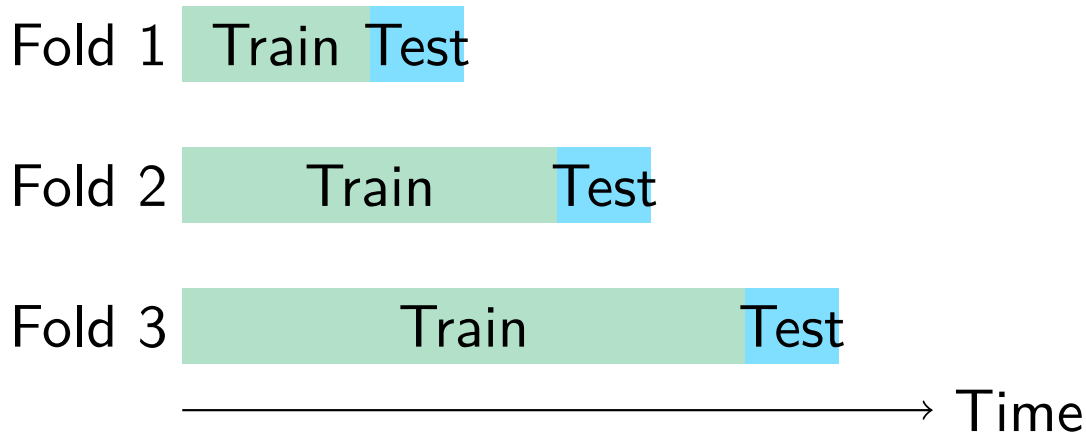
Surprise: Demographics (age, gender) ranked 25+!

Section 19

Part 10: Validation Strategy

Problem: Random CV violates timeline!

Solution: Walk forward through time



Key: Test set always **after** training set $\rightarrow$ 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!

```
# 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:

```
# 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 $\rightarrow$ Random Forest $\rightarrow$ 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