Temporal and Unsupervised Learning

Lecture 5: From Sequences to Hidden Patterns (MSc Data Science)

Prof. Asc. Endri Raco

Department of Mathematical Engineering, Polytechnic University of Tirana

November 2025

Section 1

Introduction: A New Paradigm

Slide 1: Course Overview and Transition

Previous Lectures (L1-L4):

- L1-L2: Logistic Regression (supervised, static)
- L3: Ensemble Methods (supervised, static)
- L4: Neural Networks and XAI (supervised, static)

Lecture 5 - Two New Dimensions:

1. **Temporal:** Data with time ordering (sequences matter!)
2. **Unsupervised:** No labels (find hidden structure)

Key Shift: From prediction to pattern discovery and forecasting

Slide 2: Why Temporal Data is Different

Static Data: Each observation is independent

- Example: Predicting house prices from features

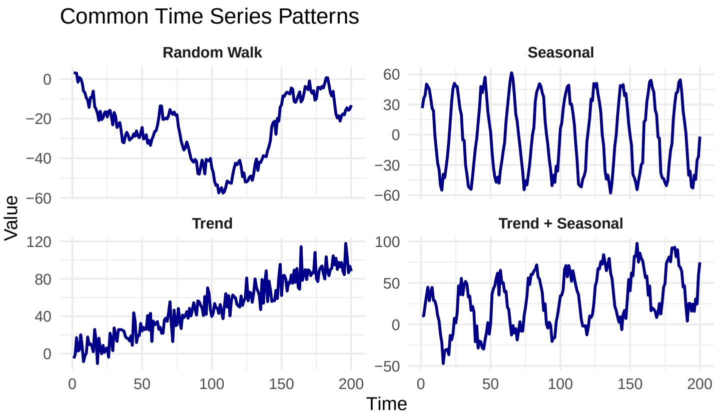
Temporal Data: Order matters, observations are dependent

- Example: Stock prices, temperature, heart rate

Key Characteristics:

- **Autocorrelation:** Values correlated with past values
- **Trends:** Long-term increases/decreases
- **Seasonality:** Regular periodic patterns
- **Non-stationarity:** Statistical properties change over time

Slide 3: Time Series Examples



Slide 4: Why Unsupervised Learning?

Supervised Learning Limitations:

- Requires labeled data (expensive, time-consuming)
- Can't discover unknown patterns
- Limited to predefined categories

Unsupervised Learning Goals:

1. **Clustering:** Group similar observations
2. **Dimensionality Reduction:** Find compact representations
3. **Association Rules:** Discover relationships
4. **Anomaly Detection:** Find unusual patterns

Use Cases: Customer segmentation, market basket analysis, exploratory data analysis

Slide 5: Lecture 5 Roadmap

Part I: Time Series Analysis (Slides 1-90)

- Time series components and decomposition
- Stationarity and transformations
- ARIMA models
- Forecasting techniques

Part II: Unsupervised Learning (Slides 91-180)

- K-Means clustering
- Hierarchical clustering
- Association rule mining
- Practical applications

Section 2

Time Series Fundamentals

Slide 6: What is a Time Series?

Definition: A sequence of observations recorded at successive time points

$$\{y_t : t = 1, 2, \dots, T\}$$

Examples:

- Daily stock prices
- Monthly sales figures
- Hourly temperature readings
- Annual GDP growth

Key Properties:

- **Temporal ordering:** t matters
- **Frequency:** Regular intervals (hourly, daily, monthly)

Slide 7: Loading and Visualizing Time Series in R

```r
# Create a time series object
# Example: Monthly airline passengers (1949-1960)
data(AirPassengers)

# View structure
str(AirPassengers)
```

```
##  Time-Series [1:144] from 1949 to 1961: 112 118 132 129 12
```

```r
# Basic plot
plot(AirPassengers,
     main = "Monthly Airline Passengers",
     xlab = "Year",
     ylab = "Passengers (thousands)",
     col = "darkblue",
     lwd = 2)
```

Slide 8: Time Series Components

Additive Decomposition:

$$Y_t = T_t + S_t + R_t$$

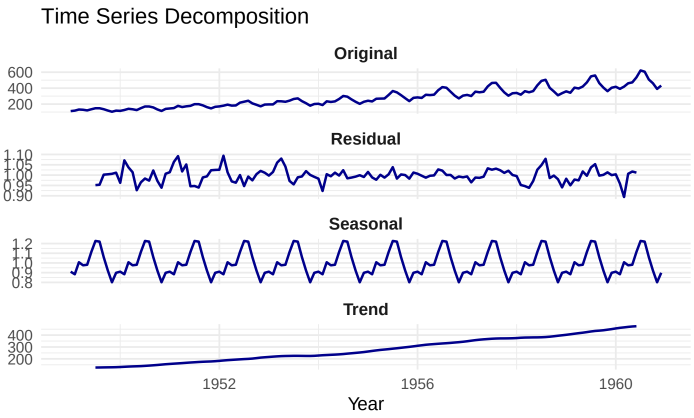
Multiplicative Decomposition:

$$Y_t = T_t \times S_t \times R_t$$

where:

- T_t = **Trend** (long-term direction)
- S_t = **Seasonal** (regular periodic variation)
- R_t = **Residual/Irregular** (random noise)

Slide 9: Visualizing Time Series Components



Slide 10: Time Series Decomposition in R

```r
# Multiplicative decomposition
decomp_mult <- decompose(AirPassengers,
                         type = "multiplicative")

# Additive decomposition
decomp_add <- decompose(AirPassengers,
                        type = "additive")

# Plot decomposition
plot(decomp_mult)

# Access components
trend <- decomp_mult$trend
seasonal <- decomp_mult$seasonal
random <- decomp_mult$random

# Remove seasonality
```

Slide 11: Stationarity - The Foundation

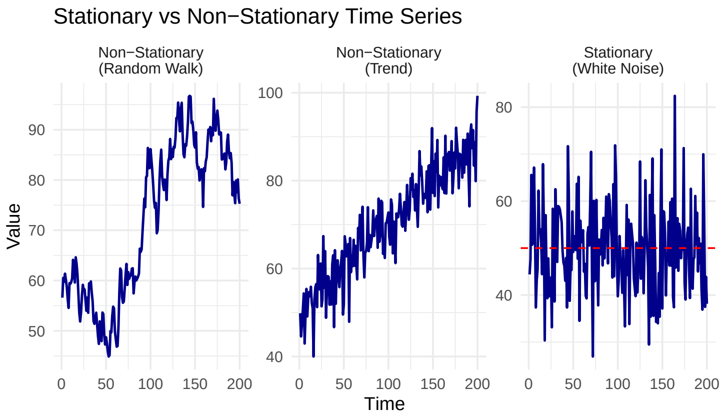
Definition: A time series is **stationary** if its statistical properties don't change over time.

Requirements:

1. **Constant mean:** $E[Y_t] = \mu$ for all t
2. **Constant variance:** $Var(Y_t) = \sigma^2$ for all t
3. **Constant autocovariance:** $Cov(Y_t, Y_{t-k})$ depends only on k, not t

Why Important: Most time series models (ARIMA) assume stationarity!

Slide 12: Stationary vs Non-Stationary



Slide 13: Testing for Stationarity - Augmented Dickey-Fuller Test

Null Hypothesis: Series has a unit root (non-stationary)

Alternative: Series is stationary

Decision Rule:

- p-value $< 0.05 \rightarrow$ Reject null $\rightarrow$ Series is **stationary**
- p-value 0.05 $\rightarrow$ Fail to reject $\rightarrow$ Series is **non-stationary**

```r
library(tseries)

# Test AirPassengers for stationarity
adf.test(AirPassengers)

# Typical output:
# Dickey-Fuller = -1.6, p-value = 0.73
# Conclusion: Non-stationary (p > 0.05)
```

Slide 14: Making Series Stationary - Differencing

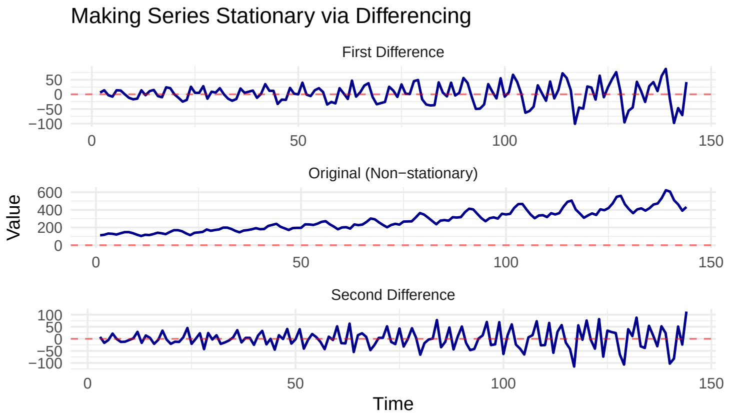
First Differencing: $\Delta Y_t = Y_t - Y_{t-1}$

Second Differencing: $\Delta^2 Y_t = \Delta Y_t - \Delta Y_{t-1}$

In R:

```r
# First difference
diff1 <- diff(AirPassengers)
plot(diff1, main = "First Difference")

# Check stationarity
adf.test(diff1)

# Second difference (if needed)
diff2 <- diff(diff1)
plot(diff2, main = "Second Difference")
```

Slide 15: Differencing Example



After first differencing: Series becomes stationary!

Slide 16: Log Transformation for Variance Stabilization

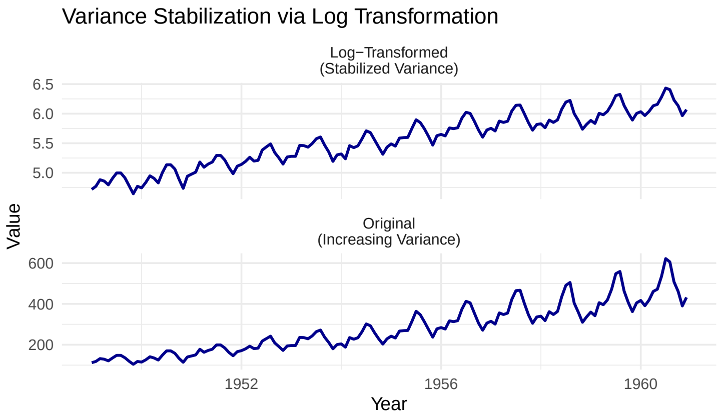
Problem: Variance increases with level (heteroscedasticity)

Solution: Log transformation

$$Y_t' = \log(Y_t)$$

```r
# Log transformation
log_series <- log(AirPassengers)
plot(log_series, main = "Log-transformed Series")

# Compare variance
var(AirPassengers)
var(log_series)

# Often combine: log + differencing
log_diff <- diff(log(AirPassengers))
plot(log_diff)
```

Slide 17: Log Transformation Example



Slide 18: Autocorrelation Function (ACF)

Definition: Correlation between Y_t and Y_{t-k} at different lags k

$$\rho_k = \frac{Cov(Y_t, Y_{t-k})}{\sqrt{Var(Y_t)Var(Y_{t-k})}}$$

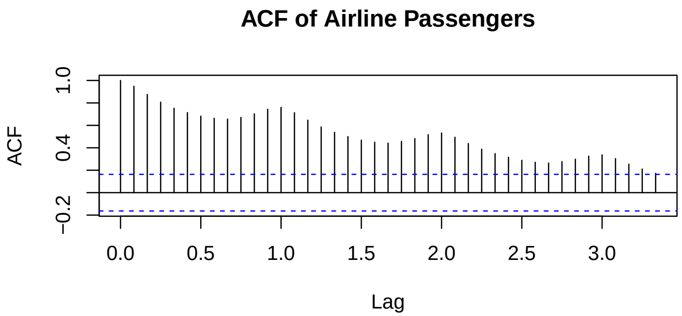
ACF Plot: Shows correlation at each lag

Interpretation:

- High ACF at lag k $\rightarrow$ Strong relationship with past k periods
- ACF cuts off $\rightarrow$ MA process
- ACF decays slowly $\rightarrow$ AR process or non-stationary

Slide 19: ACF in R

```r
# Compute and plot ACF
acf(AirPassengers,
    main = "ACF of Airline Passengers",
    lag.max = 40)
```



```r
# Blue dashed lines = significance bounds
# Values outside bounds = significant correlation
```

Slide 20: Partial Autocorrelation Function (PACF)

Definition: Correlation between Y_t and Y_{t-k} after removing effects of intermediate lags

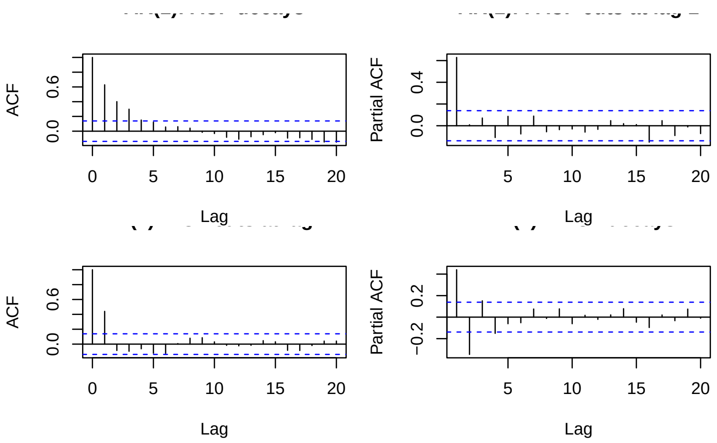
Use Case: Identify order of AR process

Interpretation:

- PACF cuts off at lag $p \rightarrow$ AR(p) process
- PACF decays gradually $\rightarrow$ MA process

```r
# Compute and plot PACF
pacf(AirPassengers,
     main = "PACF of Airline Passengers",
     lag.max = 40)
```

Slide 21: ACF vs PACF - Model Identification



Slide 22: White Noise - The Baseline

Definition: Series with no autocorrelation

$$Y_t \sim N(0, \sigma^2), \quad \text{all } t$$

Properties:

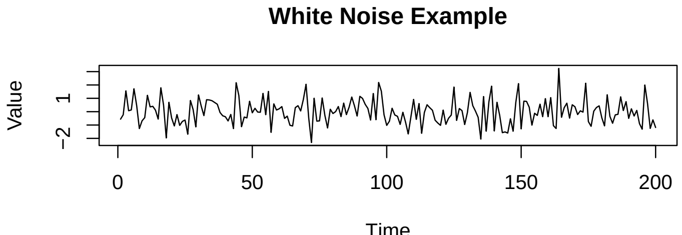
- Mean $=$ 0 (or constant)
- Constant variance
- No correlation between observations
- Unpredictable (best forecast $=$ mean)

Test: Ljung-Box test

- H0: Series is white noise
- p-value $<$ 0.05 $\rightarrow$ Reject null $\rightarrow$ Has structure (predictable)

Slide 23: Testing for White Noise

```r
set.seed(123)

# Generate white noise
white_noise <- rnorm(200, mean = 0, sd = 1)

# Plot
plot(white_noise, type = "l",
     main = "White Noise Example",
     ylab = "Value", xlab = "Time")
```



Slide 24: Random Walk - The Simplest Non-Stationary Process

Definition: Current value = previous value + random shock

$$Y_t = Y_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2)$$

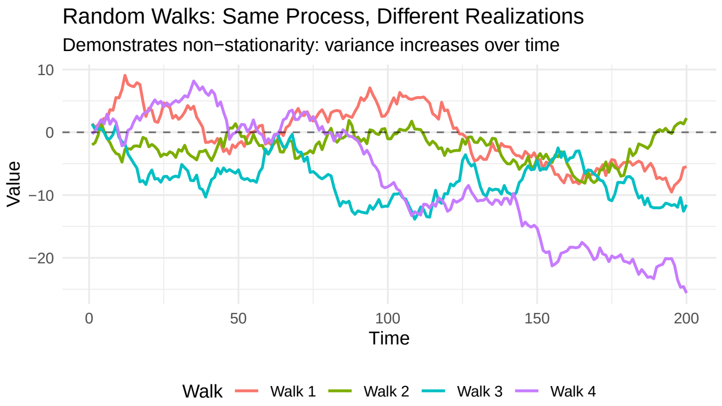
Properties:

- Non-stationary (variance increases over time)
- Best forecast = current value
- Common in financial data (stock prices)

Make stationary: First differencing

$$\Delta Y_t = Y_t - Y_{t-1} = \epsilon_t \quad \text{(white noise!)}$$

Slide 25: Random Walk Visualization



Slide 26: Autoregressive (AR) Models

AR(p) Model: Current value depends on p past values

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t$$

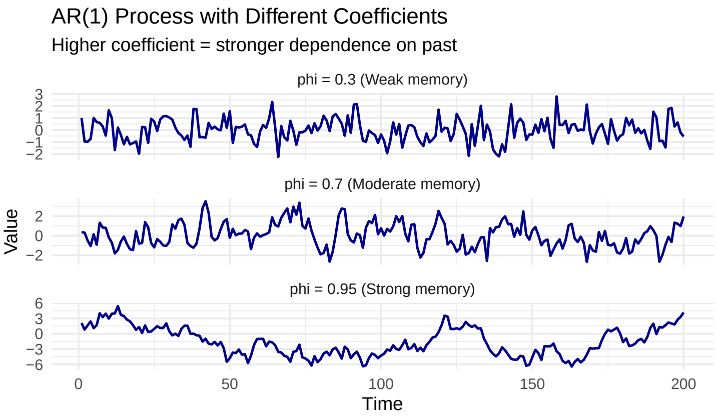
Example - AR(1):

$$Y_t = c + \phi_1 Y_{t-1} + \epsilon_t$$

Stationarity Condition: $|\phi_1| < 1$

Interpretation: If $\phi_1 = 0.7$, then 70% of previous value carries forward

Slide 27: AR(1) Process Examples



Slide 28: Moving Average (MA) Models

MA(q) Model: Current value depends on past q error terms

$$Y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

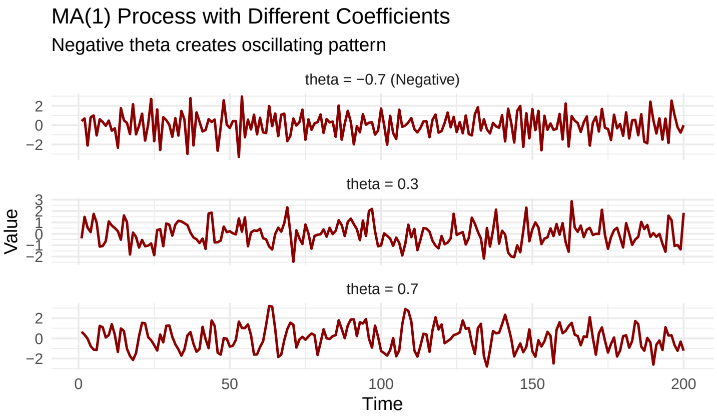
Example - MA(1):

$$Y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1}$$

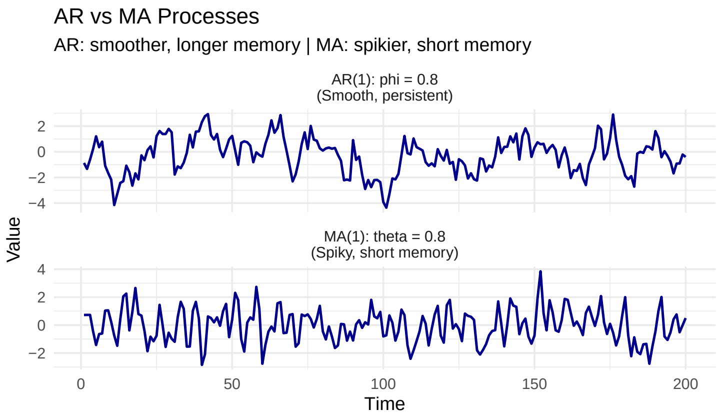
Key Difference from AR:

- AR: Depends on past **values**
- MA: Depends on past **errors**

Slide 29: MA(1) Process Examples



Slide 30: AR vs MA - Visual Comparison



Section 3

ARIMA Models

Slide 31: Combining AR and MA - ARMA Models

ARMA(p,q) Model: Combines AR(p) and MA(q)

$$Y_t = c + \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

Why Combine?

- More flexible than pure AR or MA
- Often need fewer parameters
- Better fits real-world data

Example - ARMA(1,1):

$$Y_t = c + \phi_1 Y_{t-1} + \epsilon_t + \theta_1 \epsilon_{t-1}$$

Slide 32: ARIMA - Adding Integration

ARIMA(p,d,q) Model:

- **p:** Order of AR component
- **d:** Degree of differencing (Integration)
- **q:** Order of MA component

Process:

1. Difference the series d times to achieve stationarity
2. Fit ARMA(p,q) to the differenced series

Common Models:

- ARIMA(1,0,0) $=$ AR(1)
- ARIMA(0,1,1) $=$ Random walk $+$ MA(1) shock
- ARIMA(1,1,1) $=$ Most common in practice

Slide 33: ARIMA Model Selection - The Box-Jenkins Method

Iterative 4-Step Process:

1. **Identification:** Determine p, d, q
   - Check stationarity (ADF test)
   - Plot ACF/PACF
   - Choose candidate models
2. **Estimation:** Fit model parameters
3. **Diagnostic Checking:** Validate residuals
   - Should be white noise
   - Check ACF of residuals
4. **Forecasting:** Use the validated model

Slide 34: Step 1 - Model Identification Guide

ARIMA Model Identification Rules

ACF	PACF	Model
Cuts off at lag q	Decays gradually	MA(q)
Decays gradually	Cuts off at lag p	AR(p)
Decays gradually	Decays gradually	ARMA(p,q)

'Cuts off' = drops to zero after lag k
'Decays' = gradually approaches zero

Slide 35: Determining d - Differencing Order

Strategy:

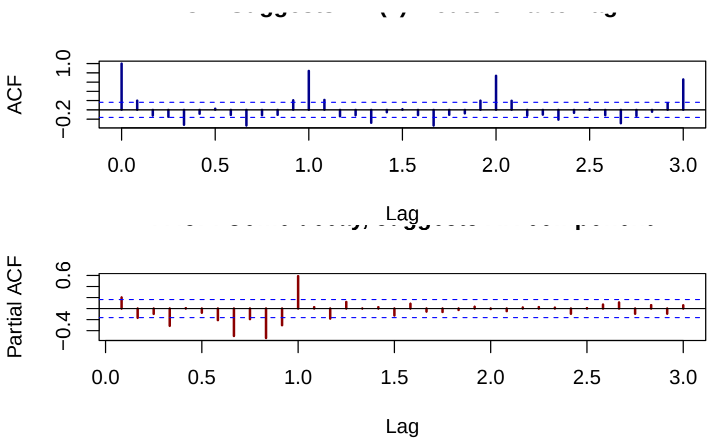
1. Plot the series
2. Run ADF test
3. If p-value $> 0.05 \rightarrow$ Apply first difference (d=1)
4. Test again
5. Repeat if needed (rarely d > 2)

```
# Test original series
adf.test(AirPassengers)  # p-value = 0.01 (non-stationary)

# First difference
diff1 <- diff(AirPassengers)
adf.test(diff1)  # p-value = 0.01 (stationary!)

# Conclusion: d = 1
```

Slide 36: Determining p and q - ACF/PACF Analysis

```r
# After differencing, examine ACF/PACF
diff_series <- diff(log(AirPassengers))

par(mfrow = c(2, 1))
acf(diff_series, lag.max = 40,
    main = "ACF of Differenced Series")
pacf(diff_series, lag.max = 40,
     main = "PACF of Differenced Series")

# Look for:
# - ACF: Significant spike at lag q → MA(q)
# - PACF: Significant spike at lag p → AR(p)
# - Both: ARMA(p,q)
```

Slide 37: ACF/PACF for Model Selection



Interpretation: Try ARIMA(1,1,1) or ARIMA(0,1,1) as starting points

Slide 38: Fitting ARIMA Models in R

```r
library(forecast)

# Manual specification
model1 <- arima(AirPassengers, order = c(1, 1, 1))
summary(model1)

# Auto ARIMA (automated selection)
model_auto <- auto.arima(AirPassengers,
                         seasonal = TRUE,
                         stepwise = TRUE,
                         trace = TRUE)

summary(model_auto)

# Compare models
AIC(model1)
AIC(model_auto)
```

Slide 39: Model Comparison - Information Criteria

Akaike Information Criterion (AIC):

$$AIC = -2\log(L) + 2k$$

Bayesian Information Criterion (BIC):

$$BIC = -2\log(L) + k\log(n)$$

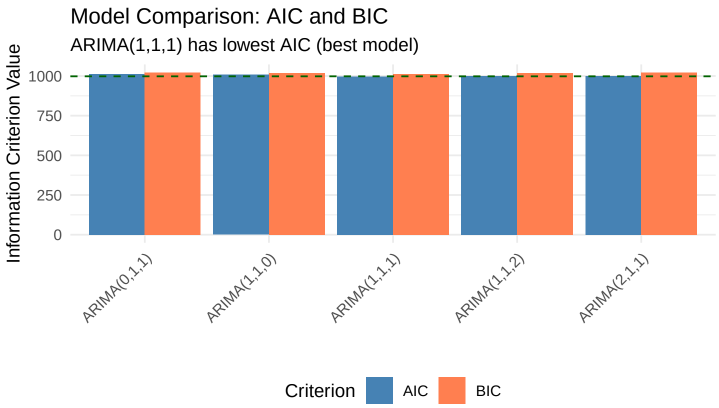
where:

- L = likelihood
- k = number of parameters
- n = sample size

Rule: Lower is better (balances fit vs complexity)

BIC: Penalizes complexity more heavily than AIC

Slide 40: Model Comparison Example



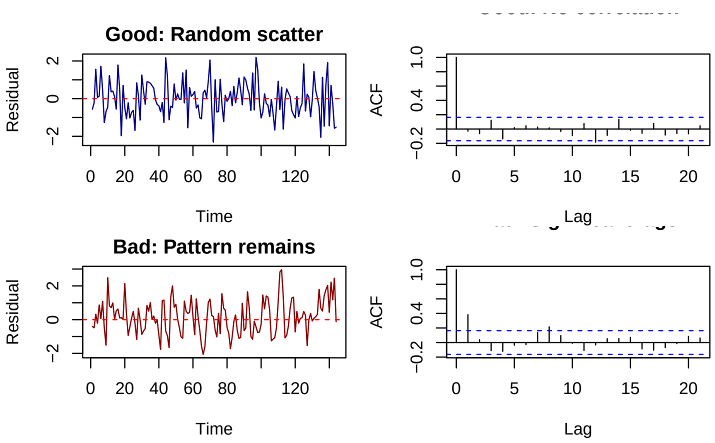
Slide 41: Diagnostic Checking - Residual Analysis

After fitting, check residuals should be white noise:

1. **Plot residuals:** No patterns
2. **ACF of residuals:** No significant lags
3. **Ljung-Box test:** p-value > 0.05
4. **Normality:** QQ-plot

```r
# Fit model
model <- arima(AirPassengers, order = c(1, 1, 1))

# Extract residuals
residuals <- residuals(model)

# Diagnostic plots
tsdiag(model)

# Ljung-Box test
Box.test(residuals, lag = 20, type = "Ljung-Box")
# p > 0.05 → Residuals are white noise (good!)
```

Slide 42: Residual Diagnostics Visualization



Slide 43: Complete ARIMA Workflow Example

```r
library(forecast)

# 1. Load and visualize data
data(AirPassengers)
plot(AirPassengers)

# 2. Check stationarity
adf.test(AirPassengers)  # Non-stationary

# 3. Transform and difference
log_ap <- log(AirPassengers)
diff_ap <- diff(log_ap)
adf.test(diff_ap)  # Stationary!

# 4. Examine ACF/PACF
acf(diff_ap)
pacf(diff_ap)
```

Slide 44: Forecasting with ARIMA

Point Forecast: Expected future value

$$\hat{Y}_{T+h} = E[Y_{T+h} | Y_1, \ldots, Y_T]$$

Prediction Interval: Uncertainty around forecast

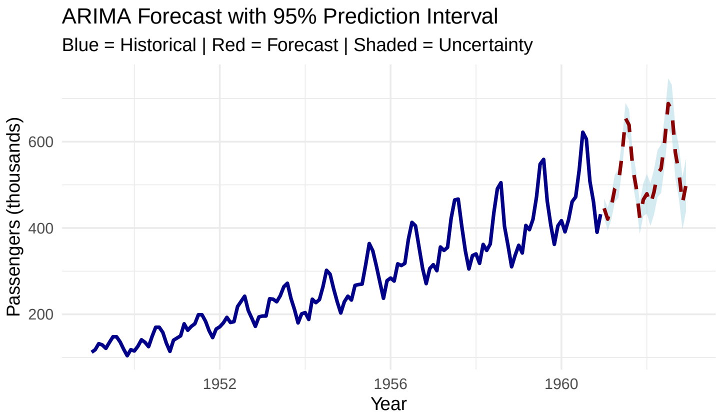
- 80% interval: 80% chance true value falls within
- 95% interval: 95% chance true value falls within

Key Property: Intervals widen as horizon h increases (more uncertainty)

Slide 45: Generating Forecasts in R

```r
# Fit model
model <- auto.arima(AirPassengers)

# Forecast 24 months ahead
forecasts <- forecast(model, h = 24)

# View forecasts
print(forecasts)

# Plot forecast with prediction intervals
plot(forecasts,
     main = "24-Month Forecast",
     xlab = "Year",
     ylab = "Passengers")

# Dark gray = 80% interval
# Light gray = 95% interval
```

Slide 46: Forecast Visualization



Slide 47: Forecast Accuracy Metrics

Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Lower is better for all metrics

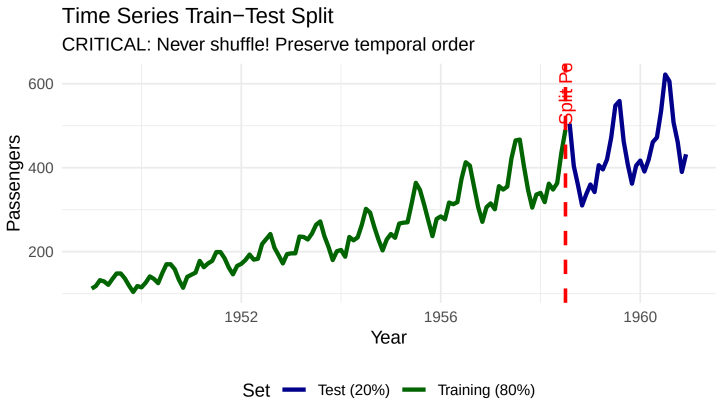
Slide 48: Cross-Validation for Time Series

Time Series Cross-Validation (Rolling Origin):

1. Train on data up to time t
2. Forecast next period(s)
3. Compare with actual
4. Move origin forward, repeat

```r
# Time series cross-validation
cv_results <- tsCV(AirPassengers,
                   forecastfunction = function(x, h) {
                     forecast(auto.arima(x), h = h)
                   },
                   h = 1)  # 1-step ahead forecast

# Compute accuracy
mae <- mean(abs(cv_results), na.rm = TRUE)
rmse <- sqrt(mean(cv_results^2, na.rm = TRUE))

cat("MAE:", mae, "\n")
```

Slide 49: Train-Test Split for Time Series



Warning: Never shuffle time series data!

Slide 50: Seasonal ARIMA (SARIMA)

SARIMA(p,d,q)(P,D,Q)s Model:

- Lowercase: Non-seasonal components
- Uppercase: Seasonal components
- s: Seasonal period (e.g., 12 for monthly, 4 for quarterly)

Example: SARIMA(1,1,1)(1,1,1)

- Regular ARIMA(1,1,1)
- Seasonal ARIMA(1,1,1) with period 12

Use Case: Data with strong seasonal patterns (AirPassengers!)

Slide 51: SARIMA Model Equation

Full SARIMA Equation:

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D Y_t = \theta(B)\Theta(B^s)\epsilon_t$$

where:

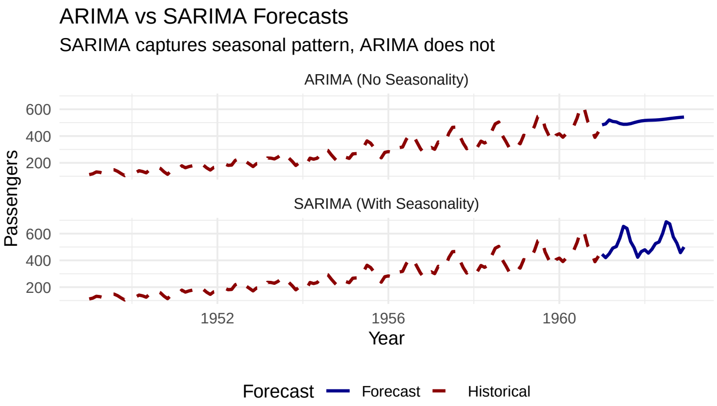
- B = Backshift operator: $BY_t = Y_{t-1}$
- $\phi(B)$ = Non-seasonal AR polynomial
- $\Phi(B^s)$ = Seasonal AR polynomial
- $\theta(B)$ = Non-seasonal MA polynomial
- $\Theta(B^s)$ = Seasonal MA polynomial

Don't panic! R handles this complexity automatically

Slide 52: Fitting SARIMA in R

```r
# Manual SARIMA specification
# SARIMA(1,1,1)(1,1,1)[12]
sarima_model <- arima(AirPassengers,
                      order = c(1, 1, 1),
                      seasonal = list(order = c(1, 1, 1),
                                      period = 12))

summary(sarima_model)

# Auto SARIMA
auto_sarima <- auto.arima(AirPassengers,
                          seasonal = TRUE,
                          stepwise = FALSE,
                          approximation = FALSE)

summary(auto_sarima)
# Typically finds: ARIMA(0,1,1)(0,1,1)[12]
```

Slide 53: SARIMA vs ARIMA Comparison



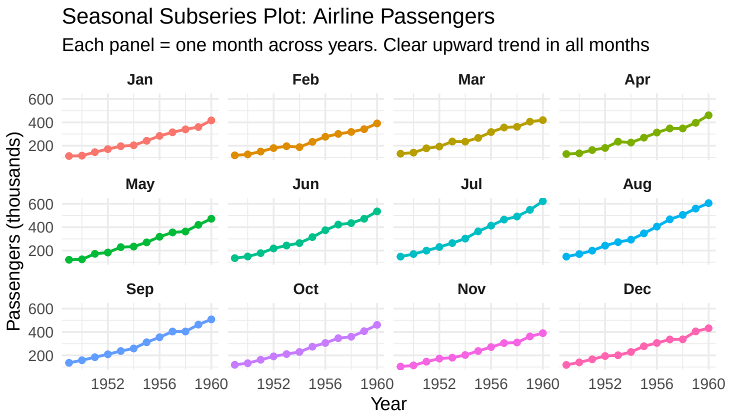
Slide 54: Identifying Seasonal Components

Check for seasonality:

1. **Visual inspection:** Regular peaks/valleys
2. **Seasonal subseries plot:** Compare same periods
3. **ACF:** Spikes at seasonal lags (12, 24, 36...)

```r
# Seasonal subseries plot
monthplot(AirPassengers,
          main = "Seasonal Subseries Plot")

# ACF shows seasonal pattern
acf(AirPassengers, lag.max = 48)
# Look for spikes at lags 12, 24, 36...

# Seasonal decomposition
decompose_result <- decompose(AirPassengers)
plot(decompose_result)
```

Slide 55: Seasonal Subseries Plot



Slide 56: Advanced Forecasting - Exponential Smoothing

Alternative to ARIMA: Exponential Smoothing

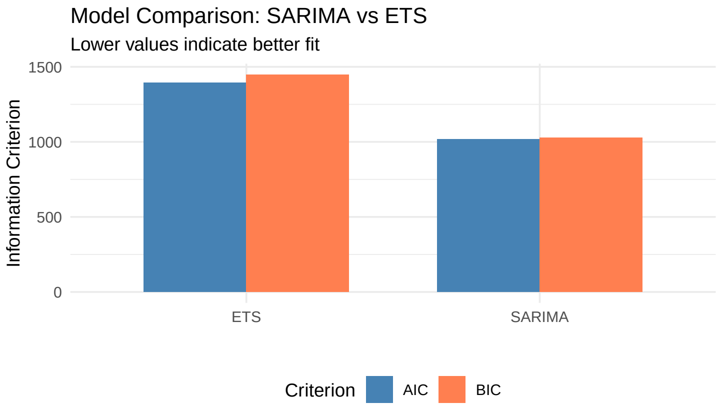
- **Simple:** No trend, no seasonality
- **Holt:** With trend
- **Holt-Winters:** With trend and seasonality

Advantage: Often more robust, easier to understand

ETS (Error, Trend, Seasonal) Framework:

```r
# Automatic ETS model selection
ets_model <- ets(AirPassengers)
summary(ets_model)

# Forecast
ets_forecast <- forecast(ets_model, h = 24)
plot(ets_forecast)
```

Slide 57: ETS vs ARIMA



Conclusion: Try both, compare performance on validation set!

Slide 58: Forecast Combination

Ensemble Forecasting: Average multiple forecasts

$$\hat{Y}_{combined} = w_1 \hat{Y}_{ARIMA} + w_2 \hat{Y}_{ETS} + w_3 \hat{Y}_{others}$$

Benefits:

- Reduces forecast variance
- Often outperforms individual models
- Robust to model misspecification

```
# Generate multiple forecasts
fc1 <- forecast(auto.arima(AirPassengers), h = 24)
fc2 <- forecast(ets(AirPassengers), h = 24)

# Simple average
combined_forecast <- (fc1$mean + fc2$mean) / 2

# Plot
```

Slide 59: Dealing with Multiple Seasonality

Problem: Data with multiple seasonal patterns

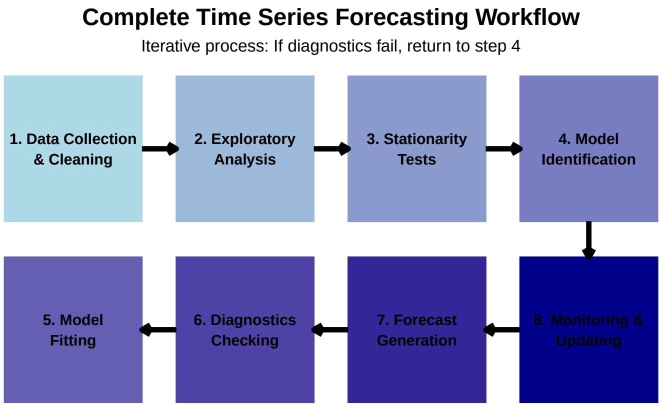
- Daily data: Weekly (7) + Yearly (365) seasonality
- Hourly data: Daily (24) + Weekly (168) seasonality

Solution: TBATS Model

- **T**rigonometric
- **B**ox-Cox transformation
- **A**RMA errors
- **T**rend
- **S**easonal components

```r
# Fit TBATS model
tbats_model <- tbats(time_series_data)

# Forecast
tbats_forecast <- forecast(tbats_model, h = 100)
plot(tbats_forecast)
```

Slide 60: Real-World Forecasting Workflow



Section 4

Advanced Time Series Topics

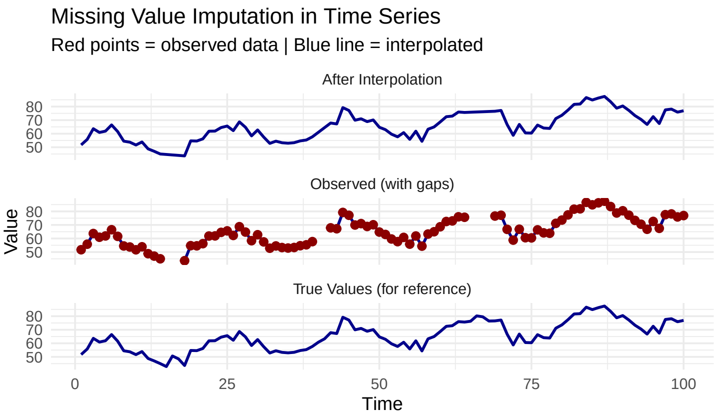
Slide 61: Handling Missing Values in Time Series

Common Approaches:

1. **Linear Interpolation:** Connect neighboring points
2. **Spline Interpolation:** Smooth curve fitting
3. **Last Observation Carried Forward (LOCF)**
4. **Model-based Imputation:** Use ARIMA to predict missing values

```
library(zoo)

# Introduce missing values (for demo)
ts_with_na <- AirPassengers
ts_with_na[c(10, 25, 50)] <- NA

# Linear interpolation
ts_interpolated <- na.approx(ts_with_na)

# Spline interpolation
ts_spline <- na.spline(ts_with_na)
```

Slide 62: Missing Value Imputation Visualization



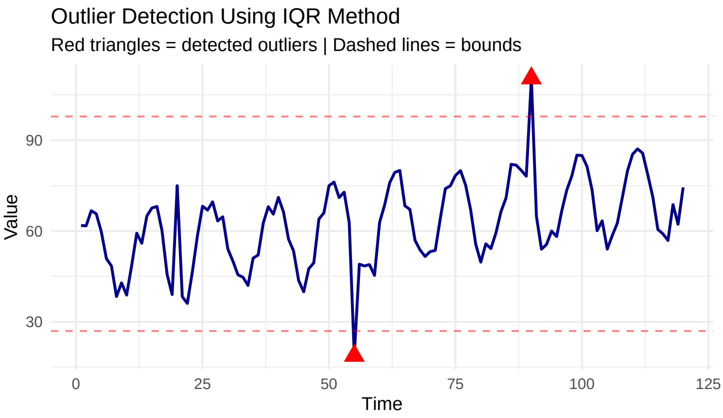
Slide 63: Outlier Detection in Time Series

Methods:

1. **Statistical:** Values beyond mean $\pm$ 3
2. **IQR Method:** Beyond Q1 - 1.5$\times$IQR or Q3 + 1.5$\times$IQR
3. **Model-based:** Residuals from fitted model

```r
# Fit model
model <- auto.arima(AirPassengers)

# Extract residuals
residuals <- residuals(model)

# Detect outliers (|residual| > 2.5 SD)
threshold <- 2.5 * sd(residuals)
outliers <- which(abs(residuals) > threshold)

# Plot
plot(AirPassengers)
points(time(AirPassengers)[outliers]
```

Slide 64: Outlier Detection Example



Slide 65: Multivariate Time Series - VAR Models

Vector Autoregression (VAR): Multiple time series that influence each other

$$\mathbf{Y}_t = \mathbf{c} + {}_1\mathbf{Y}_{t-1} + {}_2\mathbf{Y}_{t-2} + \cdots + {}_t$$

Example: Stock prices of related companies

- Apple stock $\rightarrow$ Microsoft stock
- Microsoft stock $\rightarrow$ Apple stock
- Mutual influence captured

```r
library(vars)

# Create multivariate time series
data(Canada)

# Fit VAR model
var_model <- VAR(Canada, p = 2, type = "const")
```

Slide 66: Granger Causality

Question: Does time series X "Granger-cause" Y?

Definition: X Granger-causes Y if past values of X help predict Y beyond what Y's own past values provide

Test:

- H0: X does not Granger-cause Y
- p-value $< 0.05 \rightarrow$ Reject H0 $\rightarrow$ X Granger-causes Y

```r
library(lmtest)

# Test if x Granger-causes y
grangertest(y ~ x, order = 2)

# Bidirectional test
grangertest(x ~ y, order = 2)
```

Slide 67: Structural Breaks in Time Series

Structural Break: Permanent change in time series behavior

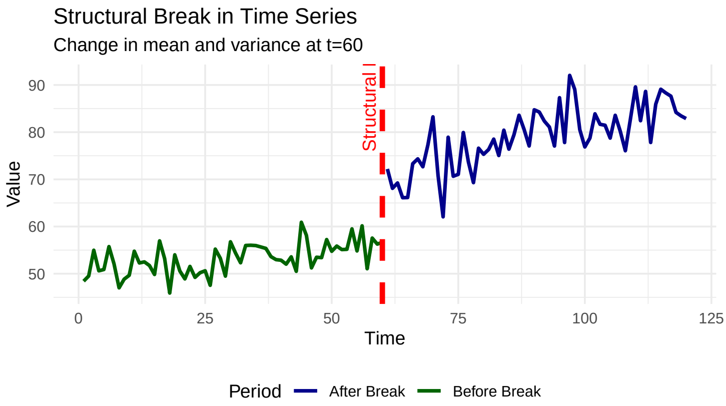
Examples:

- Policy changes (new regulations)
- Economic shocks (2008 financial crisis)
- Technology adoption (internet boom)

Detection:

```r
library(strucchange)

# Test for structural breaks
bp_test <- breakpoints(AirPassengers ~ time(AirPassengers))
summary(bp_test)

# Plot breaks
plot(bp_test)
plot(AirPassengers)
```

Slide 68: Structural Break Visualization



Slide 69: GARCH Models - Volatility Modeling

GARCH (Generalized AutoRegressive Conditional Heteroskedasticity):

Models time-varying variance (volatility clustering)

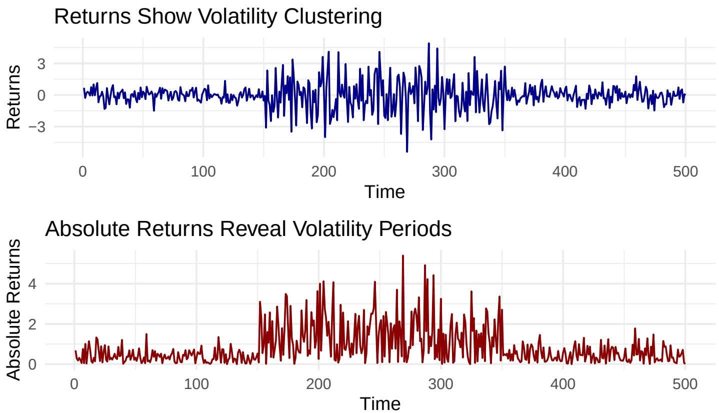
$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2$$

Use Case: Financial returns

- Periods of high volatility cluster together
- Periods of low volatility cluster together

```r
library(rugarch)

# Specify GARCH(1,1) model
spec <- ugarchspec(variance.model = list(model = "sGARCH",
                                          garchOrder = c(1, 1))

# Fit model
```

Slide 70: Volatility Clustering Example



Slide 71: Time Series Case Study - Retail Sales Forecasting

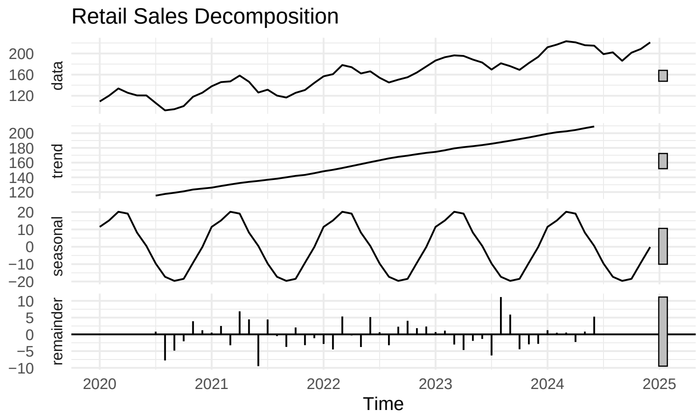
Business Problem: Predict next quarter's sales for inventory planning

Data: Monthly retail sales (5 years)

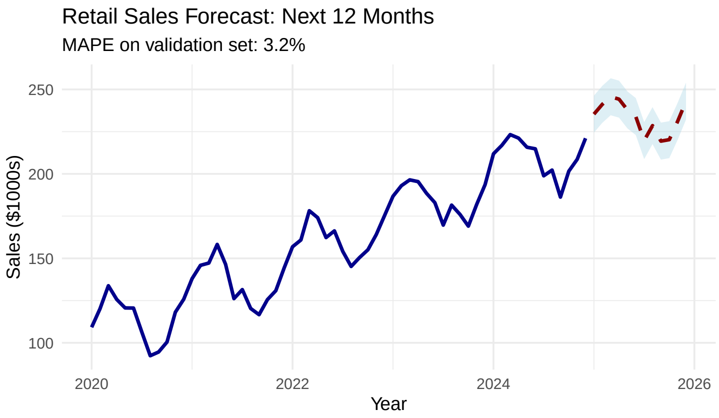
Approach:

1. EDA: Identify trend and seasonality
2. Transform: Log + seasonal differencing
3. Model: SARIMA(1,0,1)(1,1,1)
4. Validate: MAPE = 3.2% on test set
5. Deploy: Generate rolling forecasts

Slide 72: Retail Sales - Data Exploration



Slide 73: Retail Sales - Model and Forecast



Slide 74: Time Series Forecast Intervals - Interpretation

Key Points:

1. **80% PI:** 80% chance true value falls within
2. **95% PI:** 95% chance true value falls within
3. **Wider intervals = More uncertainty**

Factors Affecting Width:

- Forecast horizon (longer = wider)
- Model uncertainty
- Historical volatility
- Structural breaks

Business Use: Risk management and scenario planning

Slide 75: Common Time Series Pitfalls

Top 10 Mistakes:

1. **Ignoring stationarity** $\rightarrow$ Spurious results
2. **Over-differencing** $\rightarrow$ Destroys information
3. **Forgetting seasonality** $\rightarrow$ Poor forecasts
4. **Using future data** $\rightarrow$ Data leakage
5. **Not checking residuals** $\rightarrow$ Model inadequacy
6. **Extrapolating too far** $\rightarrow$ Unreliable forecasts
7. **Ignoring structural breaks** $\rightarrow$ Model misspecification
8. **Wrong frequency** $\rightarrow$ Seasonal pattern mismatch
9. **Not updating models** $\rightarrow$ Performance degradation
10. **Assuming stationarity** $\rightarrow$ Wrong inference

Slide 76: Time Series Checklist

Before Forecasting:

- ☐ Plot the series (visual inspection)
- ☐ Check for missing values
- ☐ Identify trend component
- ☐ Identify seasonal component
- ☐ Test for stationarity (ADF test)
- ☐ Apply transformations if needed
- ☐ Split into train/test sets
- ☐ Check for outliers
- ☐ Examine ACF/PACF plots
- ☐ Consider external events

Slide 77: Forecasting Best Practices

Production Forecasting:

1. **Monitor performance:** Track forecast errors continuously
2. **Update regularly:** Retrain as new data arrives
3. **Document assumptions:** Record transformations, parameters
4. **Maintain baselines:** Compare against naive forecasts
5. **Communicate uncertainty:** Always show prediction intervals
6. **Version control:** Track model changes
7. **A/B testing:** Compare model versions
8. **Human oversight:** Expert review of forecasts

Slide 78: Transitioning to Unsupervised Learning

From Time Series to Clustering:

Time Series Focus:

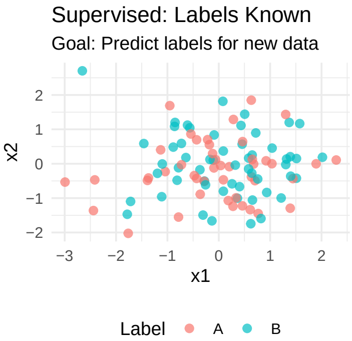
- Temporal dependence
- Forecasting future values
- Understanding trends/seasonality

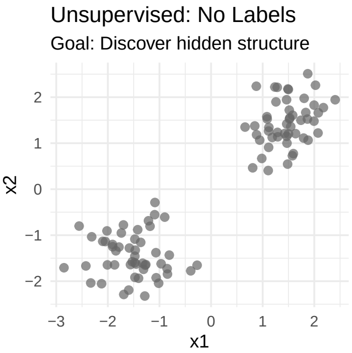
Unsupervised Learning Focus:

- Finding hidden patterns
- Grouping similar observations
- Dimensionality reduction
- **No labels needed!**

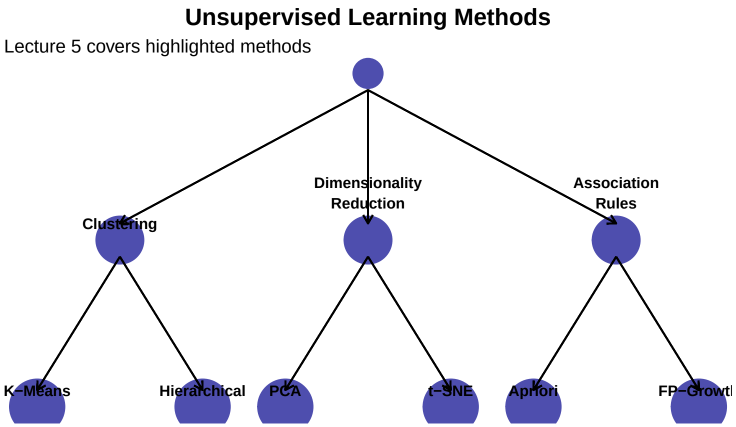
Slide 79: Why Unsupervised Learning?

Motivation:





Slide 80: Unsupervised Learning Taxonomy



Section 5

Introduction to Clustering

Slide 81: What is Clustering?

Definition: Grouping data points so that:

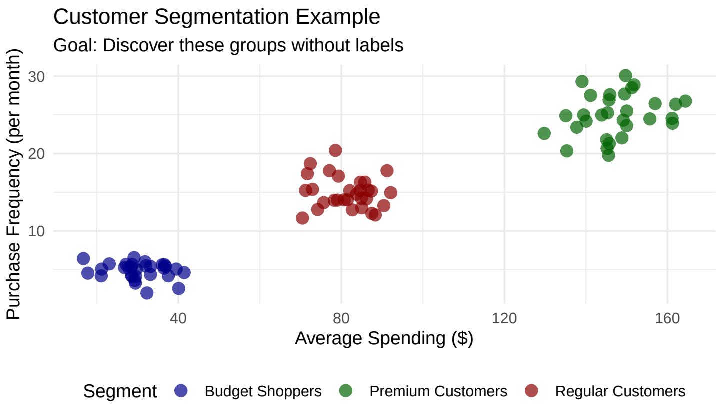
- Points in the same cluster are **similar**
- Points in different clusters are **dissimilar**

No ground truth labels!

Applications:

- Customer segmentation
- Document categorization
- Image compression
- Anomaly detection
- Gene expression analysis

Slide 82: Clustering Example - Customer Segmentation



Slide 83: Distance Metrics - Measuring Similarity

Common Distance Measures:

1. **Euclidean Distance (L2):**

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

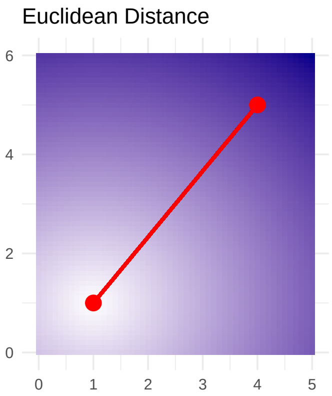
2. **Manhattan Distance (L1):**

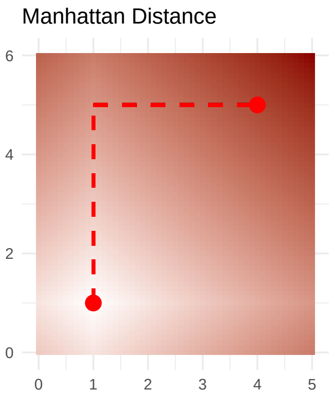
$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

3. **Cosine Similarity:**

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

Slide 84: Distance Metrics Visualization





Slide 85: Feature Scaling - Critical for Clustering

Problem: Features on different scales dominate distance calculations

Example:

- Income: \$20,000 - \$200,000
- Age: 18 - 65

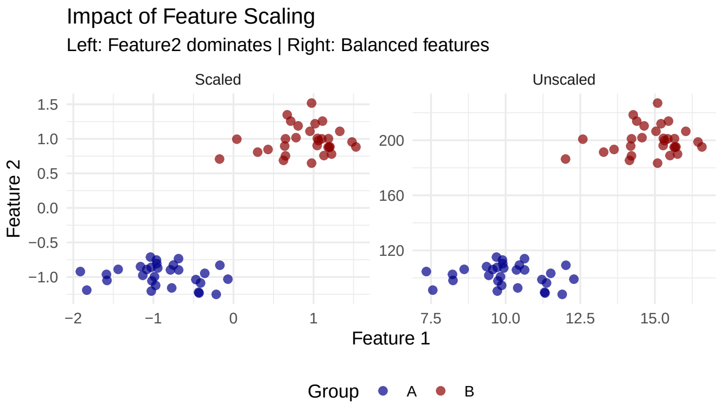
Income will dominate the distance!

Solution: Standardization

$$z = \frac{x - \mu}{\sigma}$$

```r
# Standardize features
data_scaled <- scale(data)

# Alternative: Min-Max scaling [0, 1]
data_minmax <- apply(data, 2, function(x) {
  (x - min(x)) / (max(x) - min(x))
```

Slide 86: Impact of Scaling on Clustering



Slide 87: K-Means Algorithm - Overview

Goal: Partition n observations into k clusters

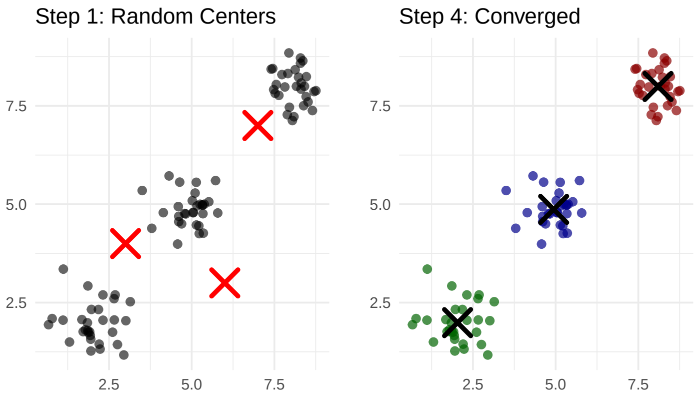
Algorithm:

1. **Initialize:** Randomly select k cluster centers
2. **Assignment:** Assign each point to nearest center
3. **Update:** Recalculate centers as mean of assigned points
4. **Repeat:** Until convergence (centers don't change)

Objective: Minimize within-cluster sum of squares (WCSS)

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Slide 88: K-Means Step-by-Step Visualization



Slide 89: K-Means in R

```r
# Load and prepare data
data(iris)
iris_features <- iris[, 1:4]

# Scale features
iris_scaled <- scale(iris_features)

# Fit K-Means with k=3
set.seed(123)
kmeans_result <- kmeans(iris_scaled,
                        centers = 3,
                        nstart = 25)  # Try 25 random starts

# View results
kmeans_result$centers    # Cluster centers
kmeans_result$cluster    # Cluster assignments
kmeans_result$size       # Cluster sizes
```

Slide 90: Choosing K - The Elbow Method

Problem: How many clusters?

Elbow Method:

1. Run K-Means for different values of k
2. Plot WCSS vs k
3. Look for “elbow” (point of diminishing returns)

```r
# Compute WCSS for k = 1 to 10
wcss <- numeric(10)

for (k in 1:10) {
  kmeans_fit <- kmeans(data_scaled, centers = k, nstart = 25)
  wcss[k] <- kmeans_fit$tot.withinss
}

# Plot elbow curve
plot(1:10, wcss, type = "b",
     xlab = "Number of Clusters (k)"
```

Slide 91: Mining Association Rules - Complete Example

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

# Load grocery transactions
data("Groceries")

# Summary statistics
summary(Groceries)
# Output: 9835 transactions, 169 items

# Item frequency plot
itemFrequencyPlot(Groceries, topN = 20,
                  type = "absolute",
                  main = "Top 20 Most Frequent Items")

# Mine rules
rules <- apriori(Groceries,
```

Slide 92: Silhouette Analysis - Alternative to Elbow

Silhouette Coefficient: Measures how well each point fits its cluster

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where: - $a(i) =$ average distance to points in same cluster - $b(i) =$ average distance to points in nearest other cluster

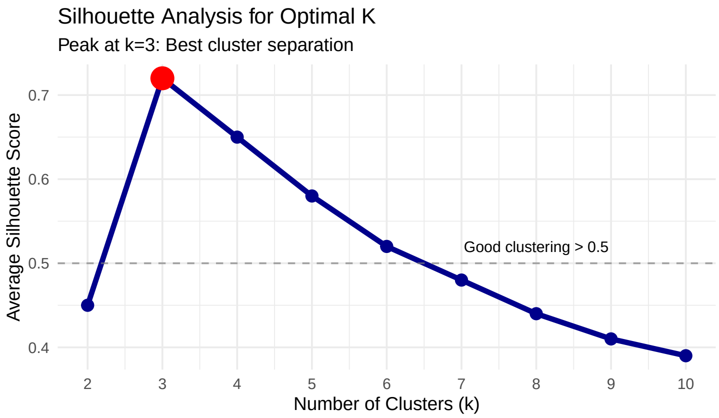
Range: -1 (wrong cluster) to $+1$ (perfect fit)

Rule: Choose k that maximizes average silhouette score

Slide 93: Silhouette Score Calculation

```r
library(cluster)

# Compute silhouette scores for different k
silhouette_scores <- numeric(9)

for (k in 2:10) {
  kmeans_fit <- kmeans(data_scaled, centers = k, nstart = 25)

  # Calculate silhouette
  sil <- silhouette(kmeans_fit$cluster, dist(data_scaled))
  silhouette_scores[k-1] <- mean(sil[, 3])
}

# Plot
plot(2:10, silhouette_scores, type = "b",
     xlab = "Number of Clusters (k)",
     ylab = "Average Silhouette Score",
```

Slide 94: Silhouette Plot Visualization



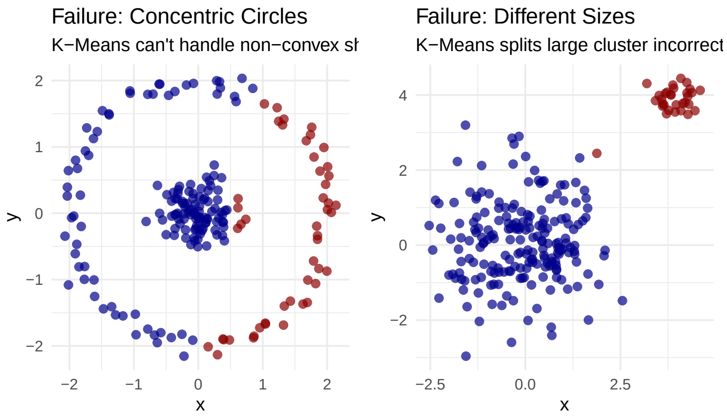
Slide 95: K-Means Limitations

Major Drawbacks:

1. **Must specify K in advance** (chicken-egg problem)
2. **Sensitive to initialization** (local minima)
3. **Assumes spherical clusters** (equal variance)
4. **Sensitive to outliers** (means get pulled)
5. **Hard to interpret in high dimensions**

When K-Means Fails: - Non-convex shapes (crescents, rings) - Different cluster densities - Different cluster sizes

Slide 96: When K-Means Fails - Examples



Slide 97: K-Means++ Initialization

Problem: Random initialization can lead to poor results

K-Means++ Solution: Smart initialization

1. Choose first center randomly
2. For each remaining center:
    - Choose point with probability proportional to distance from nearest existing center
    - Favors points far from current centers

Result: Better initial centers → faster convergence, better clusters

```r
# K-Means++ is default in R
kmeans_result <- kmeans(data,
                        centers = 3,
                        nstart = 25,  # Multiple random start
                        algorithm = "Lloyd")
```

Slide 98: Mini-Batch K-Means for Large Data

Problem: Standard K-Means slow on large datasets

Mini-Batch K-Means:

1. Sample random mini-batch of data
2. Assign points to nearest center
3. Update centers based on mini-batch only
4. Repeat

Advantages: - Much faster (suitable for millions of points) - Similar quality to standard K-Means - Scalable to streaming data

Slide 99: K-Means Case Study - Customer Segmentation

Business Problem: E-commerce company wants to segment 10,000 customers

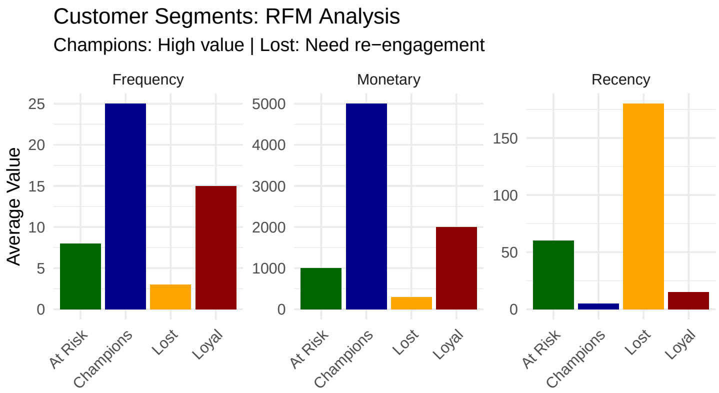
Features: - Recency (days since last purchase) - Frequency (number of purchases) - Monetary (total spending)

Approach: RFM Analysis with K-Means

Slide 100: RFM Segmentation Implementation

```r
# Load customer data
# customers <- read.csv("customer_data.csv")

# Create RFM features
rfm_data <- customers %>%
  group_by(customer_id) %>%
  summarise(
    Recency = as.numeric(max(purchase_date) - Sys.Date()),
    Frequency = n(),
    Monetary = sum(purchase_amount)
  )

# Scale features
rfm_scaled <- scale(rfm_data[, 2:4])

# Determine optimal k using elbow method
# ... (see previous slides)
```

Slide 101: RFM Segments Interpretation



Slide 102: Business Actions from Segmentation

Segment-Specific Strategies:

Segment	Characteristics	Action
Champions	Recent, frequent, high \$	VIP treatment, early access
Loyal	Regular buyers	Loyalty rewards, referrals
At Risk	Haven't bought recently	Win-back campaigns
Lost	Inactive, low value	Minimal marketing spend

ROI: Targeted campaigns $\rightarrow$ 3x conversion vs. mass marketing

Section 6

Hierarchical Clustering

Slide 103: Introduction to Hierarchical Clustering

Key Difference from K-Means:

- **K-Means:** Flat partitioning (must choose K)
- **Hierarchical:** Creates tree structure (dendrogram)

Two Approaches:

1. **Agglomerative (Bottom-Up):**
    - Start: Each point is its own cluster
    - Iteratively merge closest clusters
    - End: One cluster containing all points
2. **Divisive (Top-Down):**
    - Start: All points in one cluster
    - Iteratively split clusters
    - End: Each point in its own cluster

Slide 104: Agglomerative Clustering Algorithm

Algorithm:

1. **Initialize:** Treat each point as a cluster (n clusters)
2. **Repeat:**
    - Find two closest clusters
    - Merge them
    - Update distance matrix
3. **Stop:** When desired number of clusters reached (or all merged)

Output: Dendrogram showing merge history

Advantage: Don't need to specify K upfront!

Slide 105: Linkage Methods - Measuring Cluster Distance

How to measure distance between clusters?

1. **Single Linkage (MIN):**

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

2. **Complete Linkage (MAX):**

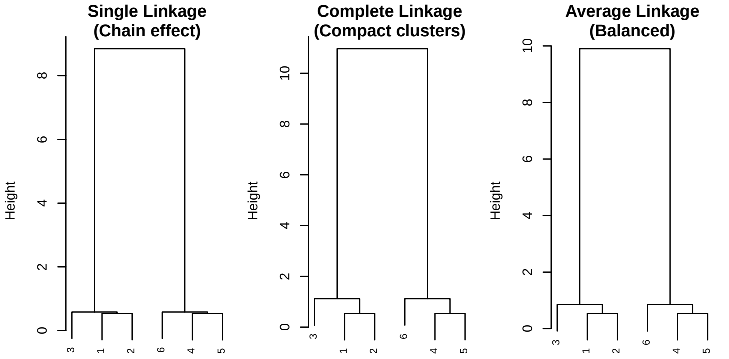
$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y)$$

3. **Average Linkage:**

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$$

4. **Ward's Method:** Minimize within-cluster variance

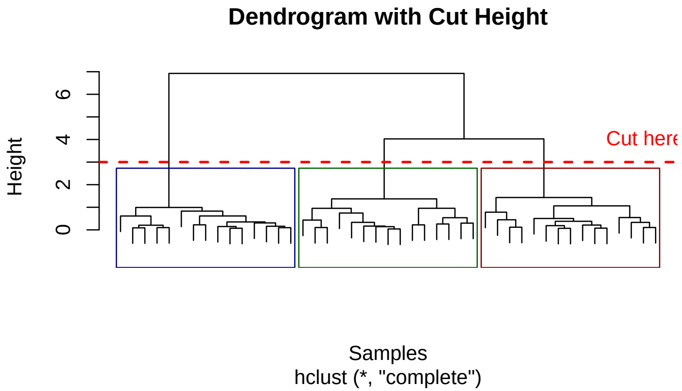
Slide 106: Linkage Methods Comparison



Slide 107: Hierarchical Clustering in R

```
# Prepare data
iris_features <- iris[, 1:4]
iris_scaled <- scale(iris_features)

# Compute distance matrix
dist_matrix <- dist(iris_scaled, method = "euclidean")

# Perform hierarchical clustering
hc_result <- hclust(dist_matrix, method = "complete")

# Plot dendrogram
plot(hc_result,
     main = "Hierarchical Clustering Dendrogram",
     xlab = "Sample Index",
     ylab = "Distance",
     cex = 0.6)
```

Slide 108: Dendrogram Interpretation



Reading the Dendrogram: - Height = dissimilarity between merged clusters - Lower merges = more similar - Horizontal line = choose number of clusters

Slide 109: Cutting the Dendrogram

```r
# Method 1: Cut at specific height
clusters_height <- cutree(hc_result, h = 5)

# Method 2: Specify number of clusters
clusters_k <- cutree(hc_result, k = 3)

# Compare cluster assignments
table(clusters_k, iris$Species)

# Visualize clusters
pairs(iris[, 1:4],
      col = clusters_k,
      pch = 19,
      main = "Hierarchical Clustering Results")
```

Slide 110: Hierarchical vs K-Means Comparison

Aspect	K-Means	Hierarchical
K Selection	Must specify upfront	Can decide later from dendrogram
Scalability	Fast (linear)	Slow (quadratic)
Cluster Shape	Spherical only	Any shape
Deterministic	No (random init)	Yes
Memory	Low	High (distance matrix)
Interpretation	Hard	Easy (dendrogram)

Rule of Thumb: - n $<$ 5,000 $\rightarrow$ Hierarchical - n $>$ 5,000 $\rightarrow$ K-Means

Slide 111: Hierarchical Clustering Case Study - Gene Expression

Problem: Cluster genes based on expression patterns

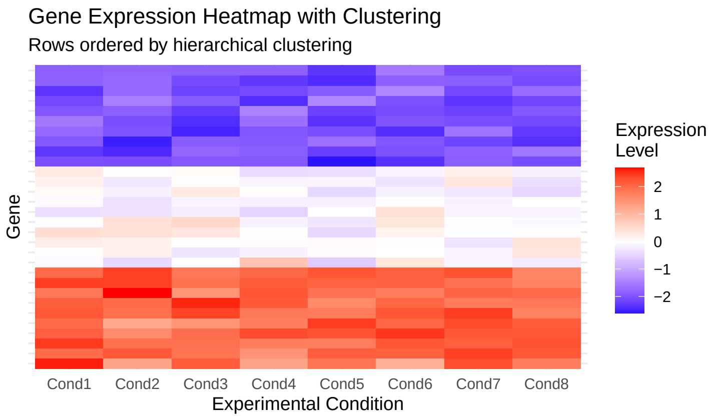
Data: Gene expression levels across different conditions

Goal: Identify co-regulated genes (similar expression patterns)

Approach:

1. Compute correlation between gene expression profiles
2. Convert correlation to distance: $d = 1 - |r|$
3. Hierarchical clustering with average linkage
4. Visualize with heatmap + dendrogram

Slide 112: Gene Expression Heatmap



Section 7

Association Rule Mining

Slide 113: Introduction to Association Rules

Goal: Find interesting relationships in transaction data

Classic Example: Market Basket Analysis

- Transaction: $\{$Bread, Milk, Eggs$\}$
- Rule: $\{$Bread, Milk$\} \rightarrow \{$Eggs$\}$
- Interpretation: "Customers who buy bread and milk also buy eggs"

Applications:

- Retail: Product recommendations
- Web: Clickstream analysis
- Healthcare: Symptom $\rightarrow$ disease associations

Slide 114: Association Rule Terminology

Key Concepts:

- **Itemset:** Set of items, e.g., {Bread, Milk}
- **Transaction:** A collection of items purchased together
- **Rule:** Implication of the form $X \to Y$

Example Transaction Database:

TID	Items
1	{Bread, Milk}
2	{Bread, Diaper, Beer, Eggs}
3	{Milk, Diaper, Beer, Cola}
4	{Bread, Milk, Diaper, Beer}
5	{Bread, Milk, Diaper, Cola}

Slide 115: Support, Confidence, and Lift

Three Key Metrics:

1. **Support:** Frequency of itemset

$$\text{Support}(X) = \frac{\text{number of transactions containing } X}{\text{total number of transactions}}$$

2. **Confidence:** Conditional probability

$$\text{Confidence}(X \to Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$$

3. **Lift:** Independence measure

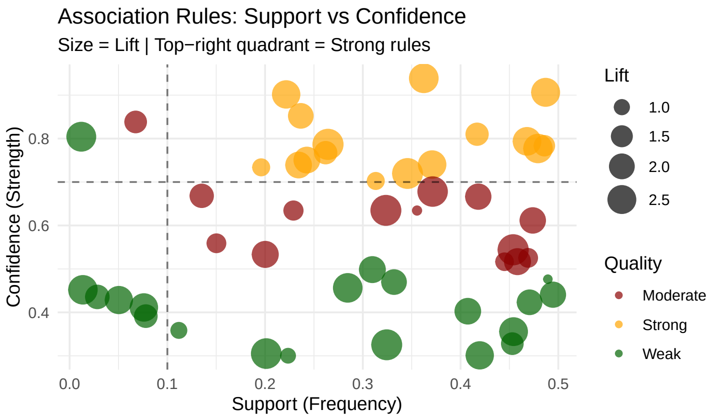
$$\text{Lift}(X \to Y) = \frac{\text{Confidence}(X \to Y)}{\text{Support}(Y)}$$

## Slide 116: Interpreting Support, Confidence, Lift

Support = 0.4 (40%): - Rule occurs in 40% of transactions - High support = frequent pattern

Confidence = 0.8 (80%): - 80% of customers who buy X also buy Y - High confidence = strong rule

Slide 117: Support-Confidence Framework



Slide 118: The Apriori Algorithm

Problem: Exponential search space (2^n possible itemsets)

Apriori Principle: If an itemset is frequent, all its subsets must be frequent

Algorithm:

1. Find all frequent 1-itemsets (support   min_support)
2. Generate candidate 2-itemsets from frequent 1-itemsets
3. Prune candidates using Apriori principle
4. Count support, keep frequent 2-itemsets
5. Repeat for k=3, 4, ... until no more frequent itemsets

Slide 119: Apriori Algorithm Illustration

Apriori Algorithm: Level−wise Search

Level 1: {A}, {B}, {C}, {D}, {E}

Prune: {E} (support < threshold)

Level 2: {A,B}, {A,C}, {A,D}, {B,C}, {B,D}, {C,D}

Prune: {A,D}, {C,D} (support < threshold)

Level 3: {A,B,C}, {A,B,D}, {B,C,D}

Prune: {B,C,D} (C,D not frequent in Level 2)

Final: {A,B,C}, {A,B,D}

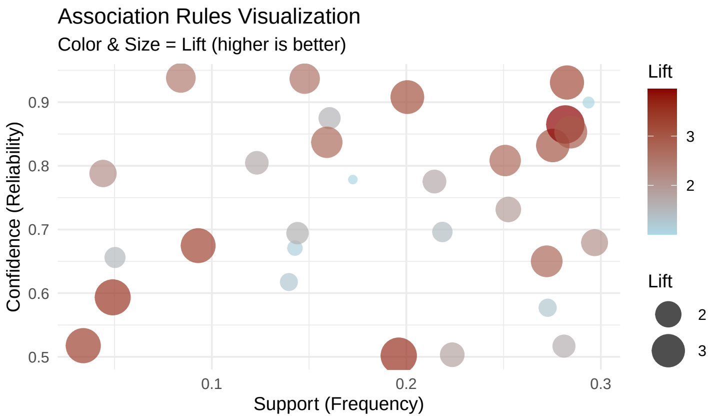
Slide 120: Association Rules in R - arules Package

```r
library(arules)

# Load example data
data("Groceries")

# Inspect transactions
inspect(Groceries[1:5])

# Mine frequent itemsets
frequent_items <- apriori(Groceries,
                          parameter = list(
                            support = 0.01,    # Min 1% suppo
                            target = "frequent itemsets"
                          ))

# Mine association rules
rules <- apriori(Groceries,
```

Slide 121: Mining Association Rules - Complete Example

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

# Load grocery transactions
data("Groceries")

# Summary statistics
summary(Groceries)
# 9835 transactions, 169 items

# Item frequency plot
itemFrequencyPlot(Groceries, topN = 20,
                  type = "absolute",
                  main = "Top 20 Most Frequent Items")

# Mine rules
rules <- apriori(Groceries,
```

Slide 122: Visualizing Association Rules



Interactive visualization: Use `arulesViz::plot(rules, method="graph")`

Slide 123: Redundant Rules and Pruning

Problem: Many rules are redundant

Example: - $\{\text{Milk}\} \rightarrow \{\text{Bread}\}$ [conf=0.8] - $\{\text{Milk, Eggs}\} \rightarrow \{\text{Bread}\}$ [conf=0.8]

If second rule has same confidence, it's **redundant**

Pruning Strategy:

```r
# Remove redundant rules
rules_pruned <- rules[!is.redundant(rules)]

# Compare
length(rules)         # Before: 463 rules
length(rules_pruned)  # After: 231 rules

# Significant rules only (high lift)
significant_rules <- subset(rules_pruned, lift > 2)
inspect(significant_rules)
```

Slide 124: Closed and Maximal Itemsets

Closed Itemset: No superset with same support

Maximal Itemset: No superset is frequent

Why Use Them?

- Reduce redundancy
- Faster mining
- More compact representation

```r
# Mine closed frequent itemsets
closed <- apriori(Groceries,
                  parameter = list(
                    support = 0.01,
                    target = "closed frequent itemsets"
                  ))

# Mine maximal frequent itemsets
maximal <- apriori(Groceries,
```

Slide 125: Sequential Pattern Mining

Extension: Consider order of purchases over time

Example Sequence: - Week 1: {Bread, Milk} - Week 2: {Eggs} - Week 3: {Butter}

Sequential Rule: {Bread, Milk} $\rightarrow$ {Eggs} $\rightarrow$ {Butter}

Applications: - Customer journey analysis - Web navigation patterns - DNA sequence analysis

```r
library(arulesSequences)

# Read sequential data
sequences <- read_baskets("sequences.txt",
                          info = c("sequenceID", "eventID"))

# Mine sequential patterns
seq_rules <- cspade(sequences,
                    parameter = list(support = 0.01)
```

Slide 126: Association Rules Case Study - Retail Recommendations

Business Goal: Increase average basket size through recommendations

Dataset: 100,000 grocery transactions over 6 months

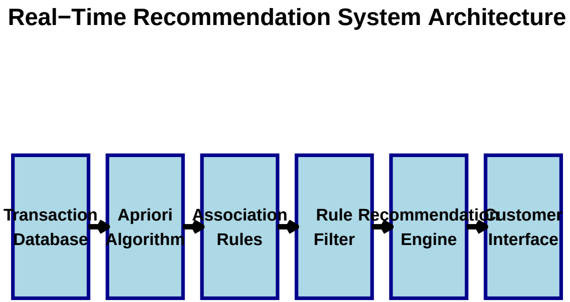
Analysis:

1. Mine rules (support   0.01, confidence   0.6)
2. Filter by lift > 2 (strong associations)
3. Remove redundant rules
4. Deploy top 50 rules to recommendation engine

Results:

- 15% increase in basket size
- 8% increase in revenue
- Most effective rule: $\{$Pasta, Tomato Sauce$\} \rightarrow \{$Parmesan Cheese$\}$

Slide 127: Recommendation System Architecture



Section 8

Density-Based Clustering

Slide 128: DBSCAN - Density-Based Spatial Clustering

Key Idea: Clusters are dense regions separated by sparse regions

Advantages over K-Means:

1. No need to specify number of clusters
2. Can find arbitrarily shaped clusters
3. Robust to outliers
4. Identifies noise points

Parameters:

- **(epsilon):** Neighborhood radius
- **minPts:** Minimum points to form dense region

Slide 129: DBSCAN Concepts

Three Types of Points:

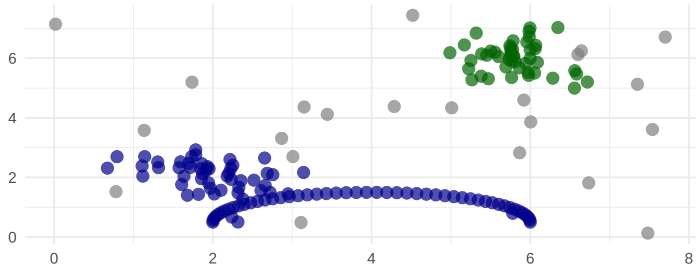
1. **Core Point:** Has minPts within
2. **Border Point:** Within of a core point, but not core itself
3. **Noise Point:** Neither core nor border

Density-Connected: Two points are in same cluster if connected through chain of core points

Slide 130: DBSCAN Visual Intuition

DBSCAN: Finds Arbitrary Shapes + Noise

Gray points = noise (outliers)



Assignment ● Noise ● Cluster 1 ● Cluster 2

Slide 131: DBSCAN Algorithm

Algorithm:

1. **Mark all points** as unvisited
2. **For each unvisited point p:**
    - Mark as visited
    - Find neighborhood N (points within)
    - If $|N| <$ minPts: Mark as **noise**
    - Else:
        - Create new cluster
        - Add p to cluster
        - **Expand cluster** from p's neighbors
3. **Repeat** until all points visited

Slide 132: DBSCAN in R

```r
library(dbscan)

# Prepare data
data_scaled <- scale(data)

# Run DBSCAN
db_result <- dbscan(data_scaled,
                    eps = 0.5,      # Neighborhood radius
                    minPts = 5)      # Min points for core

# View results
db_result$cluster  # Cluster assignments (0 = noise)
table(db_result$cluster)

# Visualize
plot(data_scaled,
     col = db_result$cluster + 1,
     pch = ifelse(db_result$cluster == 0, 4, 19))
```

Slide 133: Choosing DBSCAN Parameters

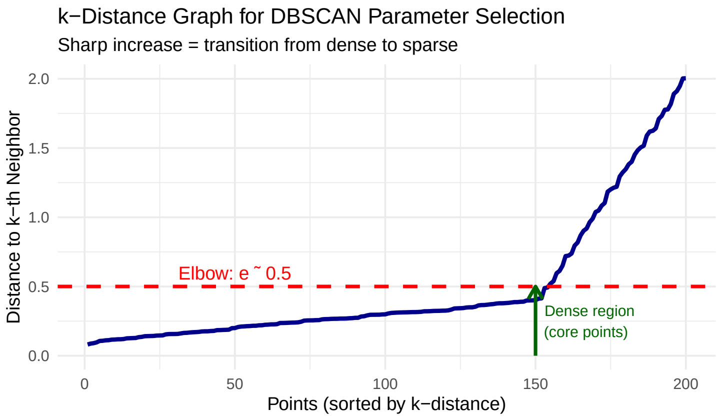
(epsilon) Selection:

Use **k-distance graph** (elbow method for DBSCAN)

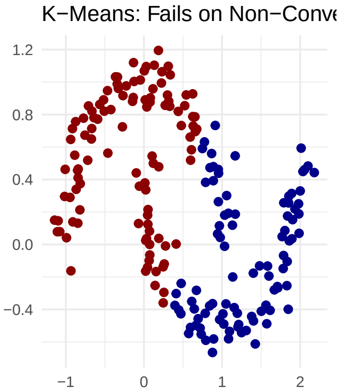
1. For each point, compute distance to k-th nearest neighbor
2. Sort distances
3. Plot sorted k-distances
4. Look for “elbow” → optimal

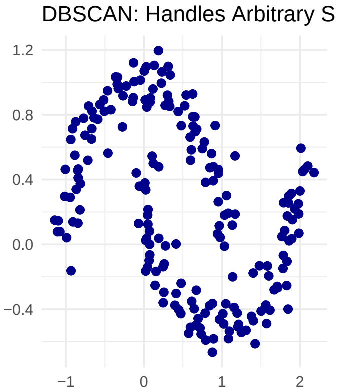
```r
# Compute k-nearest neighbor distances
k <- 5  # Same as minPts
knn_dist <- kNNdist(data_scaled, k = k)

# Sort and plot
knn_sorted <- sort(knn_dist)
plot(knn_sorted,
     type = "l",
     xlab = "Points (sorted)",
     ylab = "k-NN Distance",
```

Slide 134: k-Distance Graph for Parameter Selection



Slide 135: DBSCAN vs K-Means Comparison





Slide 136: HDBSCAN - Hierarchical DBSCAN

Problem with DBSCAN: Single doesn't work for varying densities

HDBSCAN Solution: Hierarchy of DBSCAN results

Advantages:

- Automatically selects for each cluster
- Handles varying density clusters
- More robust parameter selection (only minPts)

```r
library(dbscan)

# Run HDBSCAN
hdb_result <- hdbscan(data_scaled, minPts = 5)

# View cluster tree
plot(hdb_result, show_flat = TRUE)

# Extract flat clustering
```

Section 9

Dimensionality Reduction

Slide 137: The Curse of Dimensionality

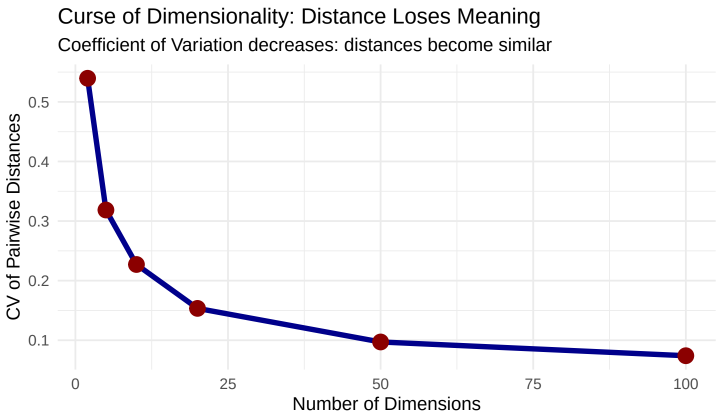
Problem: As dimensions increase:

1. **Distance loses meaning:** All points become equidistant
2. **Sparsity:** Data points spread thin
3. **Computation:** Exponential time/space
4. **Visualization:** Can't plot beyond 3D

Example: 100 features $\rightarrow$ 2$\hat{}$100 possible feature combinations!

Solution: Reduce dimensions while preserving information

Slide 138: Distance Concentration in High Dimensions



Slide 139: Principal Component Analysis (PCA)

Goal: Find directions of maximum variance

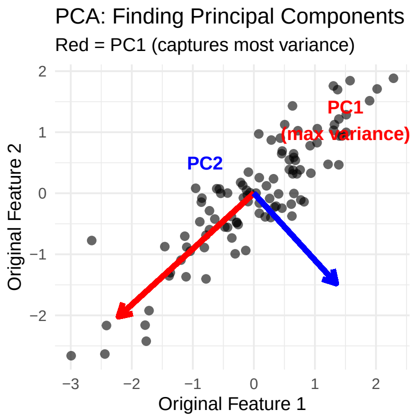
Key Idea:

1. Center the data
2. Compute covariance matrix
3. Find eigenvectors (principal components)
4. Project data onto top k eigenvectors

Result: New uncorrelated features (PC1, PC2, ...)

- PC1: Direction of maximum variance
- PC2: Direction of maximum variance perpendicular to PC1
- ...

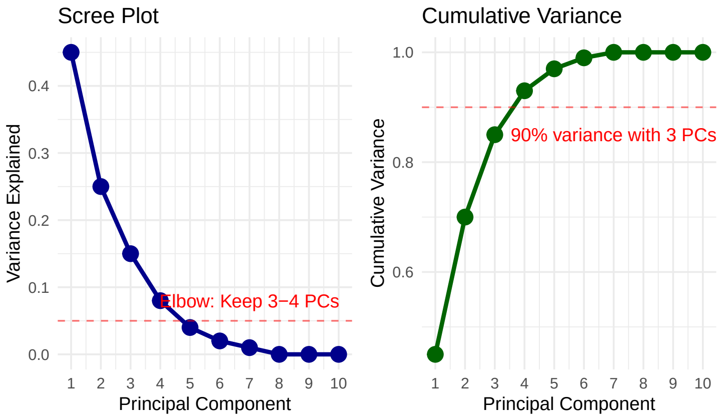
Slide 140: PCA Visualization - 2D Example



Slide 141: PCA in R - Complete Workflow

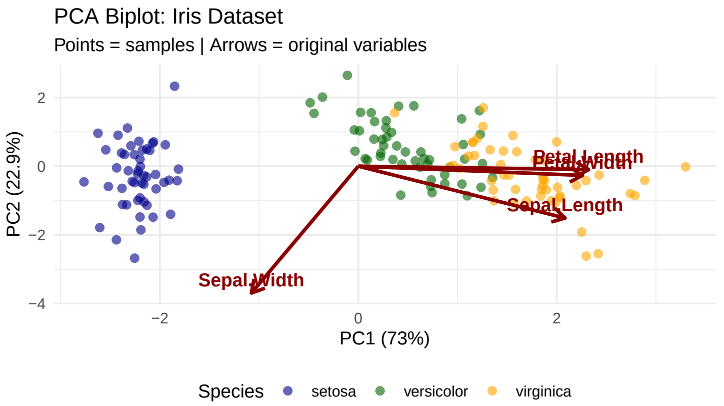
```r
# Load and prepare data
data(iris)
iris_features <- iris[, 1:4]

# Perform PCA (important: center and scale)
pca_result <- prcomp(iris_features,
                     center = TRUE,
                     scale. = TRUE)

# Summary
summary(pca_result)

# Variance explained
pca_var <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumsum(pca_var)  # Cumulative variance

# Scree plot
```

Slide 142: Scree Plot - Choosing Number of Components



Rule of Thumb: Keep PCs that explain $\geq 80 - 90\%$ cumulative variance

Slide 143: PCA Biplot - Variables and Observations



Slide 144: Interpreting PCA Results

PC Loadings Interpretation:

- High positive loading: Variable increases with PC
- High negative loading: Variable decreases with PC
- Near-zero loading: Variable uncorrelated with PC

Example - Iris Dataset:

- **PC1:** Overall size (all measurements positively correlated)
- **PC2:** Contrast between sepal and petal measurements

Use Cases:

- Data visualization (3D $\rightarrow$ 2D)
- Noise reduction
- Feature extraction for ML
- Multicollinearity reduction

Slide 145: PCA Limitations and Alternatives

PCA Limitations:

1. **Linear:** Only finds linear combinations
2. **Variance   Information:** Max variance   best features
3. **Sensitive to scaling:** Must standardize first
4. **Not interpretable:** PCs are abstract combinations

Alternatives:

- **t-SNE:** Nonlinear, preserves local structure (visualization)
- **UMAP:** Faster than t-SNE, preserves global + local
- **Autoencoders:** Deep learning approach (nonlinear)
- **Factor Analysis:** Assumes latent factors

Slide 146: t-SNE for Visualization

t-Distributed Stochastic Neighbor Embedding

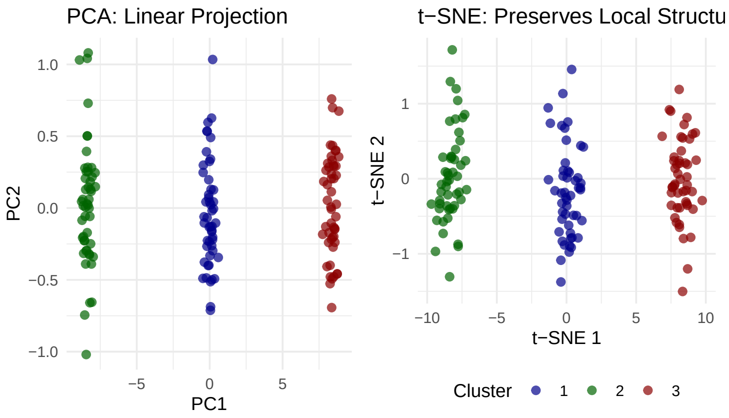
Key Idea: Preserve pairwise similarities in lower dimensions

Advantages: - Excellent for visualization - Reveals cluster structure - Handles nonlinear relationships

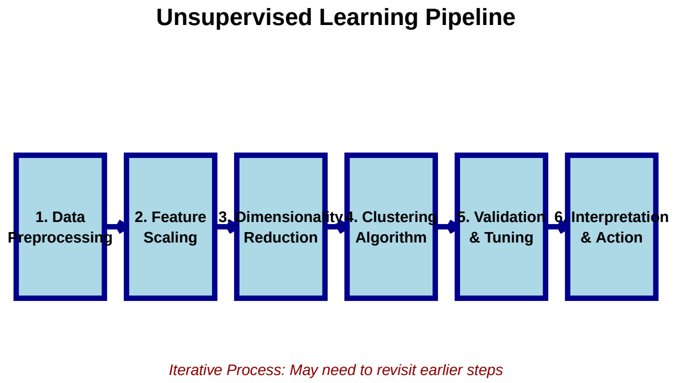
Limitations: - Slow (not for >10k points) - Non-deterministic (different runs $\rightarrow$ different results) - Only for visualization (not feature extraction)

```r
library(Rtsne)

# Prepare data
data_scaled <- scale(data)

# Run t-SNE
tsne_result <- Rtsne(data_scaled,
                     dims = 2,
                     perplexity = 30,
```

Slide 147: PCA vs t-SNE Comparison



Slide 148: Complete Unsupervised Learning Pipeline


Unsupervised Learning Pipeline
1. Data Preprocessing
2. Feature Scaling
3. Dimensionality Reduction
4. Clustering Algorithm
5. Validation & Tuning
6. Interpretation & Action
Iterative Process: May need to revisit earlier steps

Slide 149: Clustering Validation Metrics

Internal Metrics (no ground truth):

1. **Silhouette Score:** [-1, 1], higher better
2. **Davies-Bouldin Index:** Lower better
3. **Calinski-Harabasz Index:** Higher better

External Metrics (with ground truth):

1. **Adjusted Rand Index (ARI):** [-1, 1], 1 = perfect
2. **Normalized Mutual Information (NMI):** [0, 1], 1 = perfect
3. **Purity:** [0, 1], 1 = perfect

```r
library(clusterCrit)

# Internal validation
silhouette_score <- intCriteria(data, clusters, "Silhouette")
davies_bouldin <- intCriteria(data, clusters, "Davies_Bouldin'

# External validation (if labels available)
```

Slide 150: Final Project Ideas - Unsupervised Learning

Project 1: Customer Segmentation Dashboard - RFM clustering on transaction data - Interactive visualization with Shiny - Automated segment reports

Project 2: Anomaly Detection System - DBSCAN on sensor/log data - Real-time outlier alerts - Visualization dashboard

Project 3: Market Basket Analysis - Apriori on retail data - Product recommendation engine - A/B testing of recommendations

Project 4: Document Clustering - Text preprocessing + TF-IDF - K-Means on document vectors - Topic discovery and labeling

Project 5: Image Compression - PCA on image data - Compression ratio analysis - Quality vs. compression tradeoff