Best Practices for Time Series Forecasting

Presentation by

André Bauer &

Marwin Züfle

Umeå, June 20, 2019

Road Map

09:00 Uhr

10:30 Uhr

11:00 Uhr

12:30 Uhr

- Introduction
- Data Pre-Processing
- Feature Engineering
- Method Selection
- Model Fitting
- Evaluation
- Summary

On what you can expect:

- Foundations of Time Series
- Basics of Forecasting
- Basics of Feature Engineering
- Comparing Forecasting Methods
- R Code snippets

Who are we?

André Bauer
In 3rd year of PhD
Research interests:
- Forecasting
- Elasticity
- Auto-scaling
- Self-aware Computing

Marwin Züfle
In 2rd year of PhD
Research interests:
- Forecasting
- Failure Prediction
- Data Analytics

Nikolas Herbst
Post-Doc
Research interests:
- Predictive Data Analytics
- Elasticity
- Serverless

Predictive Data Analytics group is part of Descartes Research (Self-Aware Computing) headed by Samuel Kounev @ University of Würzburg

<u>Published</u>

1. Forecasting Method Selection: Examination and Ways Ahead @ICAC'19
2. Challenges and Approaches: Forecasting for Autonomic Computing @OCDCC'18
3. Telescope: A Hybrid Forecast Method for Univariate Time Series @ITISE'17
4. Online Workload Forecasting. In Self-Aware Computing Systems @Springer'17 Book chapter

<u>Under Review</u>

1. Time Series Forecasting: Review and Evaluation of the State-of-the-Art @Invited Article to PIEEE

Requirements

- Installation of R & RStudio

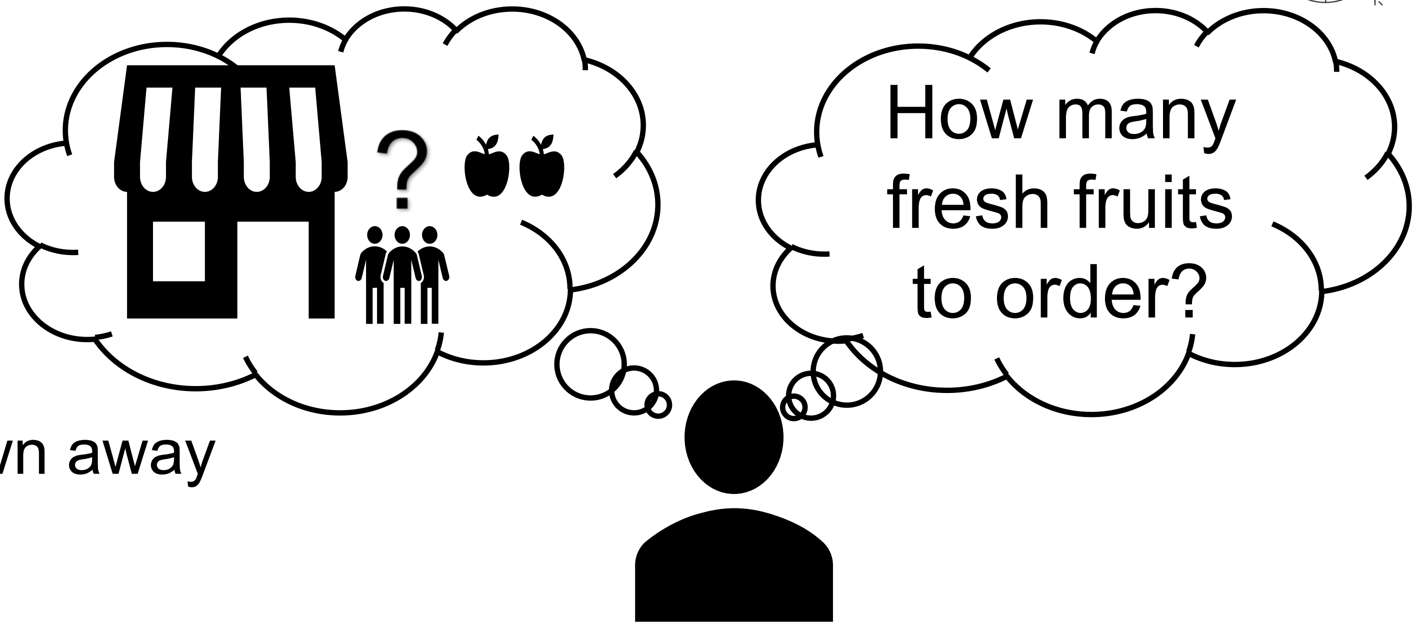
https://cran.rstudio.com/

https://www.rstudio.com/products/rstudio/download/#download

```r
# if not installed
install.packages(c("forecast", "devtools", "zoo", "ggm"))
install.packages("xgboost", "randomForest", "e1071")
```

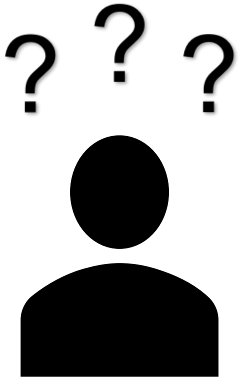
Knowing the future makes life easier!

- If shop owner buys
  - Too few fresh fruits, customers are dissatisfied
  - Too many fresh fruits, remaining fruits have to thrown away



Shop Owner

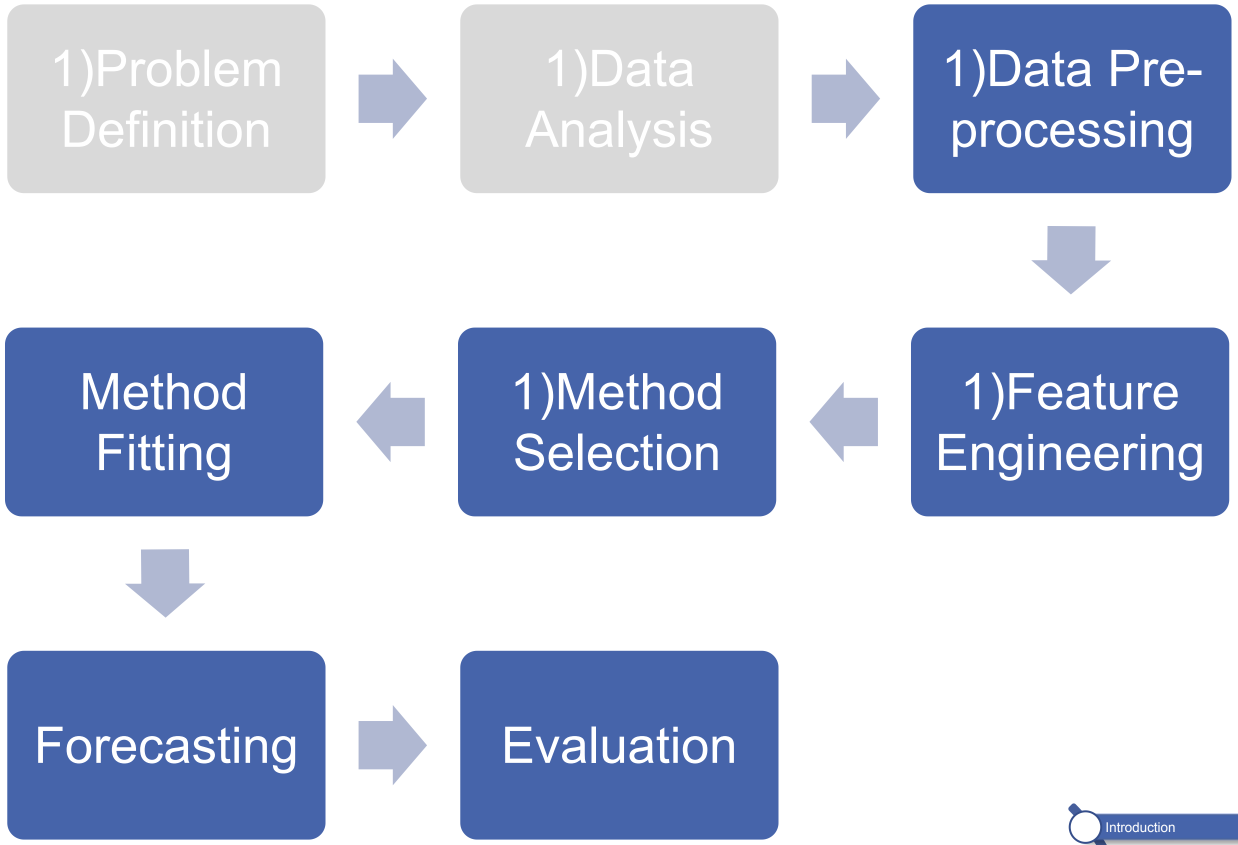
- Collect sales figures
  - Analyze purchasing behavior
  - Forecast number of required fruits



- How to forecast and which method?

Forecasting

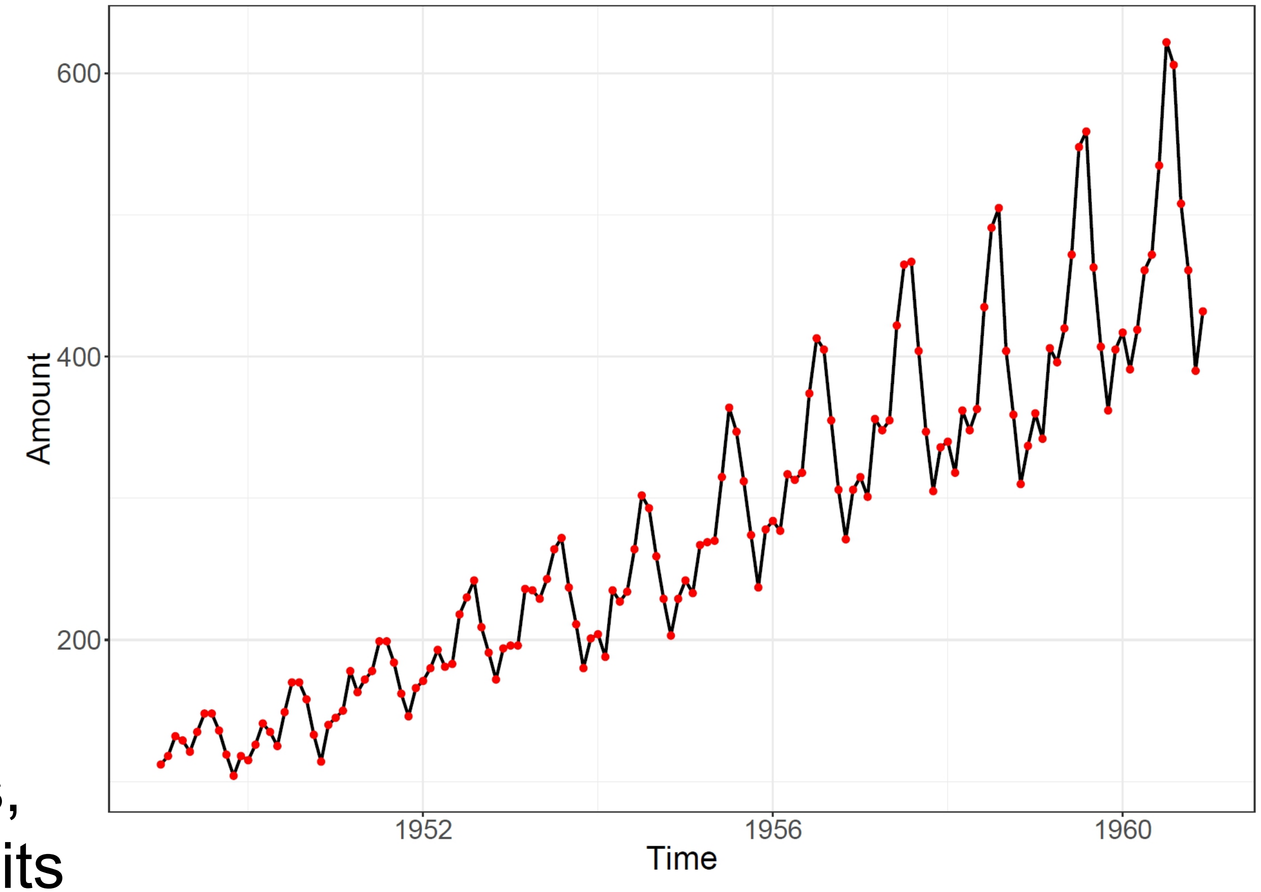
- Expert knowledge
  - Is expensive
  - Cannot be automated
- “No-Free-Lunch Theorem”
  - There is no forecasting method that performs best
  - Each method has its benefits and drawbacks



Introduction
Data Pre-Processing
Feature Engineering
Method Selection
Model Fitting
Evaluation
Summary

What is a time series?

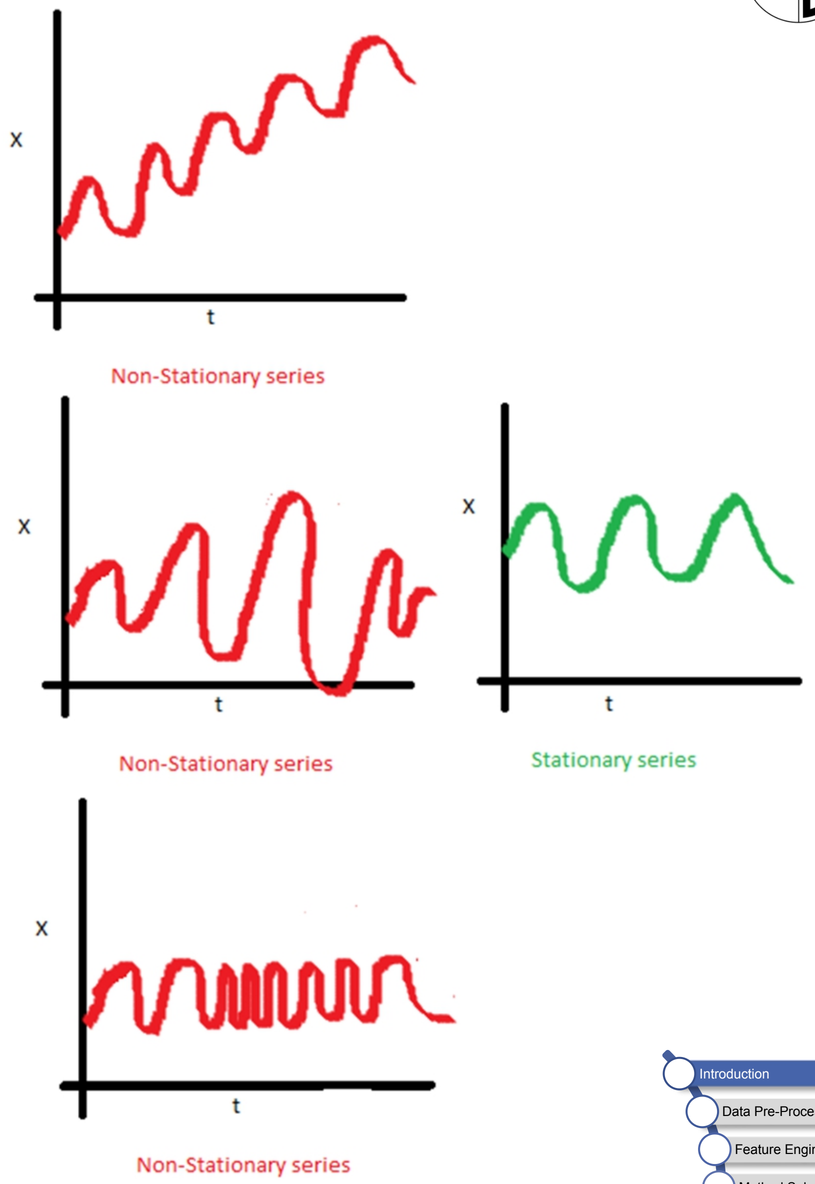
- Univariate time series
  - $Y \coloneqq \{yt : t \in T\}$
  - Ordered collection of values over a specific period
  - Equidistant time steps



- Components
  - Trend: long term movement
  - Seasonality: recurring patterns, e.g., produced by humans habits
  - Cycle: rises and falls without a fixed frequency
  - Irregular: statistical noise distribution

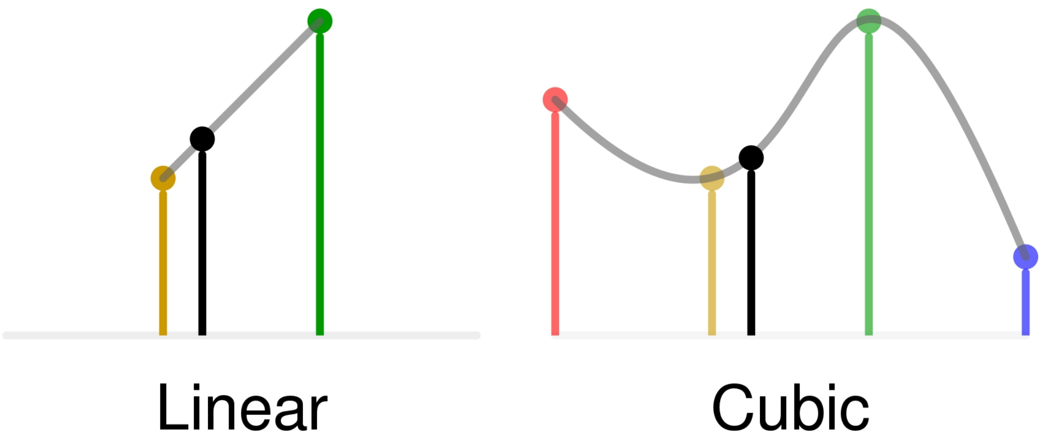
Stationarity

- Most forecasting methods assume
  - Stationarity or
  - Time series can be “stationarized”
- Statistical properties (mean, variance, …) do not change over time
- In practice
  - Time series have trend and/or season
  - Non-stationary



Missing and problematic values

- Most forecasting methods cannot handle missing values
  - At the beginning: removal
  - In between: reconstruction, e.g., interpolation



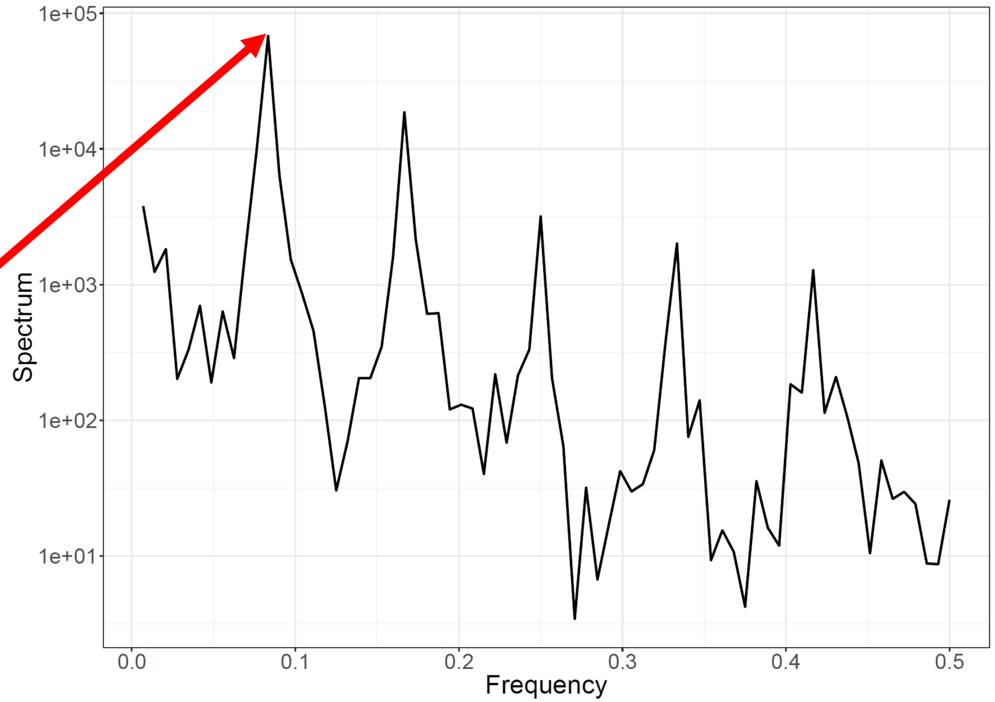
- Some forecasting methods (e.g., ETS) cannot handle negative values
  - Shift time series before forecast to positive
  - Shift time series back after forecast

Detecting seasonal patterns

- Basic idea in mathematics
  - Break down complex objects into simpler parts
  - Time series is a weighted sum of sinusoidal components

- Periodogram
  - Bases on Fourier transformation
  - Each frequency gets “probability”

Highest spectrum =
Most dominant frequency
@1/frequency

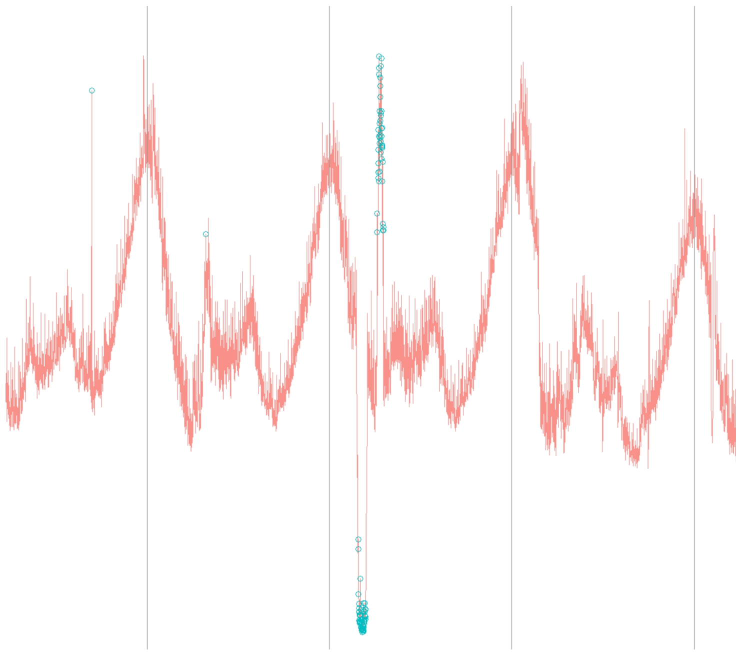


Applying a Periodogram

```r
# load package
library(forecast)

# plot AirPassengers time series
plot(AirPassengers)

# Creating and plotting the periodogram
pgram <- spec.pgram(as.vector(AirPassengers))
# Building data frame with relevant info
pgram_df <- data.frame(freq = pgram$freq, spec = pgram$spec)
# Determining the top 10 frequencies according to the spectrum
head(1/pgram_df[order(pgram_df$spec, decreasing = TRUE),1],n=10)
```

Anomaly Removal

- To increase accuracy, anomalies can be removed
  - Generalized extreme studentized deviate test
  - Replace anomalies by mean of non-anomaly neighbors
  - Twitter offers package (https://github.com/twitter/AnomalyDetection)



- Detection may be too sensitive and find false-positives

Find Anomalies

```r
# if not installed
devtools::install_github("twitter/AnomalyDetection")
# load package
library(AnomalyDetection)

# add anomalies
air <- as.vector(AirPassengers)
air[c(20,100)] <- air[c(20,100)] * 5
anom <- AnomalyDetectionVec(air, period=12, direction='both', plot=TRUE)

data(raw_data)
anom <- AnomalyDetectionVec(raw_data[,2],period=1440,
                            direction='both', plot=TRUE)
```

Feature Engineering

- “At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used” [P. M. Domingos 2012]

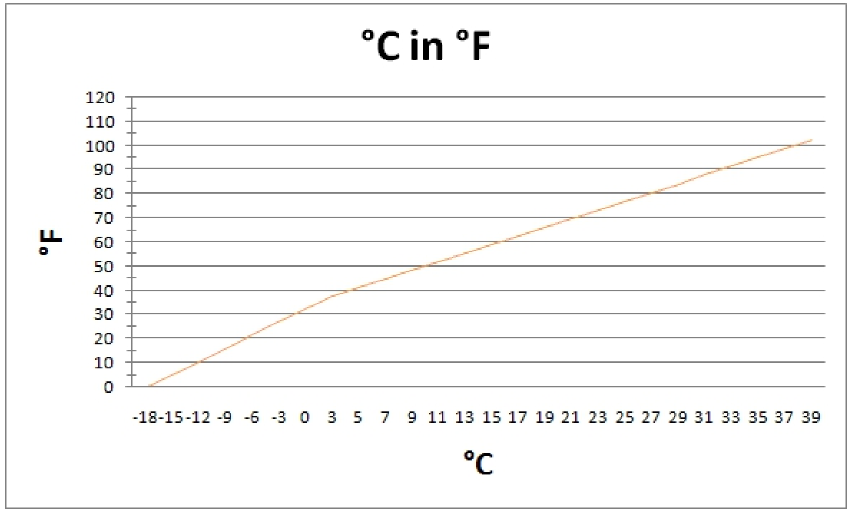
- Data transformation
  - Simplifies the model
  - May lead to better forecast

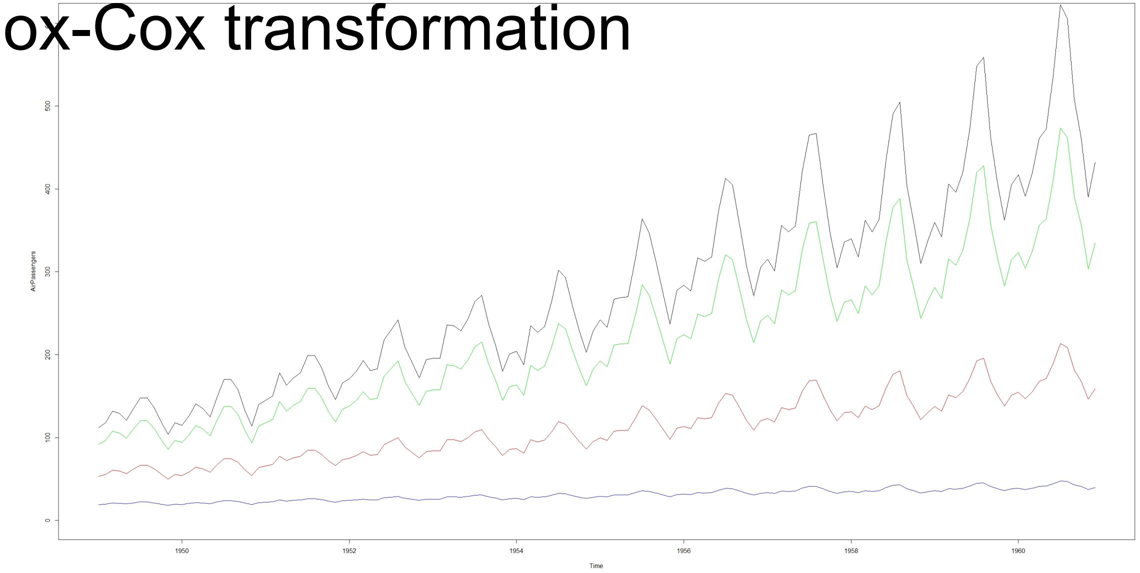
- Feature selection
  - Most statistical methods support only the time series
  - Machine learning methods rely on features

Time Series Transformation

- Time series may be complex
  - High variance
  - Multiplicity effects

- Transformation may lead to easier model
  - Common transformation is logarithm
  - Box-Cox transformation



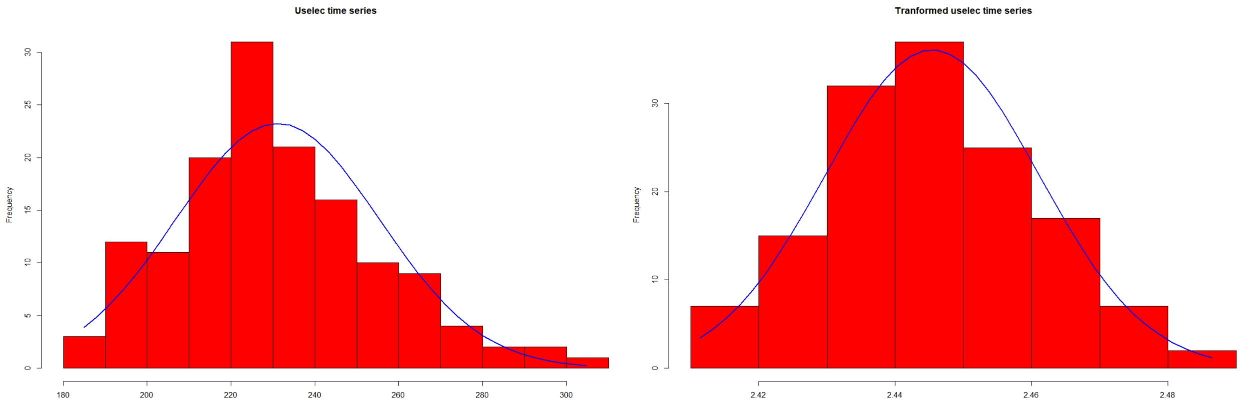


Box-Cox Transformation

- Offers family of power functions:

$$w(t) = \begin{cases} \ln(y), & \lambda = 0 \\ \dfrac{y(t)^{\lambda} - 1}{\lambda}, & otherwise \end{cases}$$

- Tries to “normal-shape” the data



- Power parameter λ can be estimated by the method of Guerrero

Box-Cox Transformation

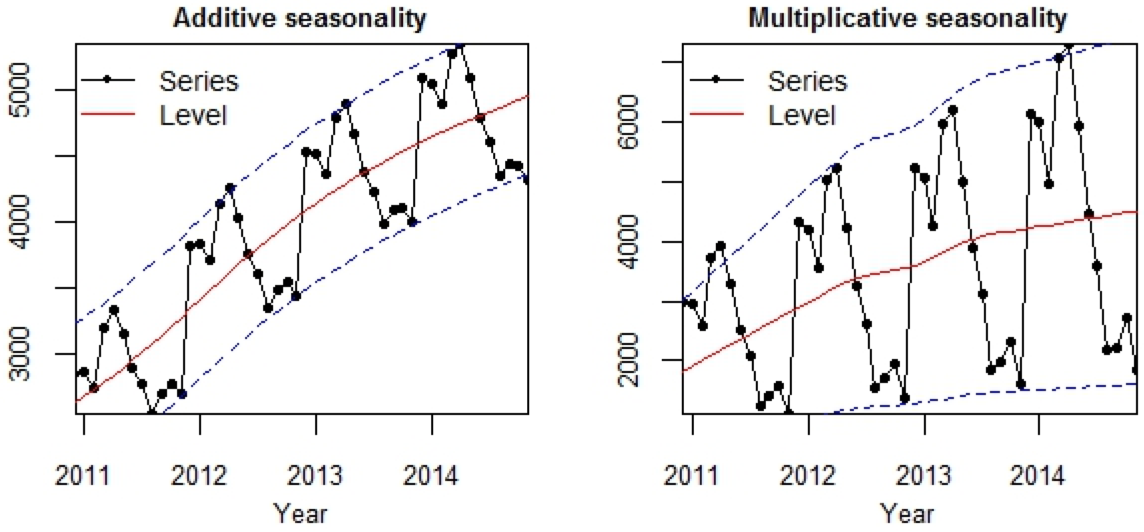
```r
# load package
library(forecast)

timeseries <- AirPassengers

# estimate best lambda
lambda <- BoxCox.lambda(timeseries)

# transform time series
trans <- BoxCox(timeseries, lambda = lambda)
```

Feature Extraction

- Additional info may increase the forecast accuracy
  - Features from external (correlated) data sources
    - Nearby sensors
    - Weather
    - …
  - Features from the given time series
    - Time series components
    - Fourier terms
    - Categorical information
    - …

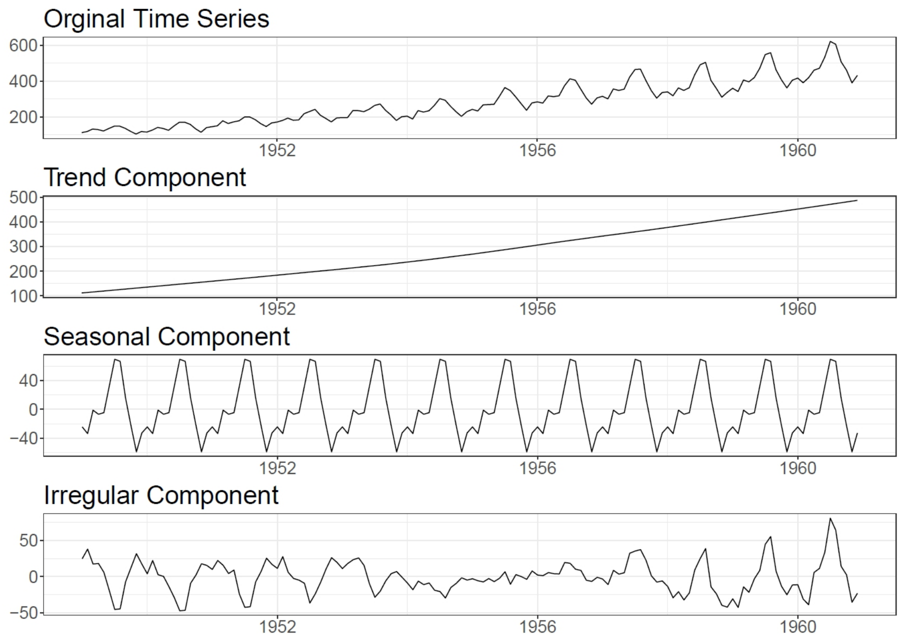
Time Series Decomposition

- Time series can be break down in different components
  - Trend, season, and irregular
  - Linear and non-linear
  - …
- Decomposition is
  - Additive or
  - Multiplicative or
  - Mixed
- Components can be used as features or for modifying the data



STL Decomposition

- STL (Seasonal and Trend decomposition using Loess)
  - Trend, season, and irregular
  - Additive
    - $Y(t) = T(t) + S(T) + I(t)$
    - $Y(t) = T(t) * S(T) * I(t)$
      is equals to $\log(Y(t)) = \log(T(t)) + \log(S(t)) + \log(I(t))$
  - Time series must
    - Be seasonal
    - Have at least two full periods
  - Parameter t.window smooths trend



Checking Decomposition

```r
# load package
library(zoo)

timeseries <- AirPassengers

# plot time series
plot(timeseries)

# get trend
trend<- rollmean(timeseries, frequency(timeseries), fill="extend",
                      align = "right")
detrended_a <- timeseries - trend
detrended_m <- timeseries / trend
```

Checking Decomposition – Cont'd

```
# get remainder
seasonal_a <- mean(detrended_a, na.rm = TRUE)
seasonal_m <- mean(detrended_m, na.rm = TRUE)
residual_a <- detrended_a - seasonal_a
residual_m <- detrended_m / seasonal_m

# calculate auto-correlations
acf_a <- acf(residual_a)
acf_m <- acf(residual_m)

if(sum(acf_a$acf^2) < sum(acf_m$acf^2)){
  print('additive decomposition')
} else {
  print('multiplicative decomposition')
}
```

STL Decomposition

```
# load package
library(forecast)

timeseries <- AirPassengers

# decompose time series
decomp <- stl(timeseries, s.window = 'periodic')
plot(decomp)

# smooth trend
decomp <- stl(timeseries, s.window = 'periodic', t.window =
    length(timeseries)/2)
plot(decomp)
```

Introduction
Data Pre-Processing
Feature Engineering
Method Selection
Model Fitting
Evaluation
Summary

STL Decomposition – Cont'd

```r
# decompose ts with multiplicative decomposition
decomp <- stl(log(timeseries), s.window = 'periodic')
plot(decomp)

timeseries <- taylor

# decomposition with different periods
decomp <- stl(ts(timeseries, frequency = 24), s.window = 'periodic')
plot(decomp)
decomp <- stl(timeseries,s.window = 'periodic')
plot(decomp)

# stl with multiple seasons
decomp <- mstl(taylor, s.window = 'periodic')
plot(decomp)
```

Fourier Terms

- Time series can be written as weighted sum of sinusoidal components

$$f(t) = \frac{a_0}{2} \sum_{k=1}^{\infty} (a_k \cos(kt) + b_k \sin(kt))$$

- For each frequency from Periodogram, Fourier terms can be extracted
  - Approximation of the time series only with dominant frequencies
  - Additional features

Fourier Terms

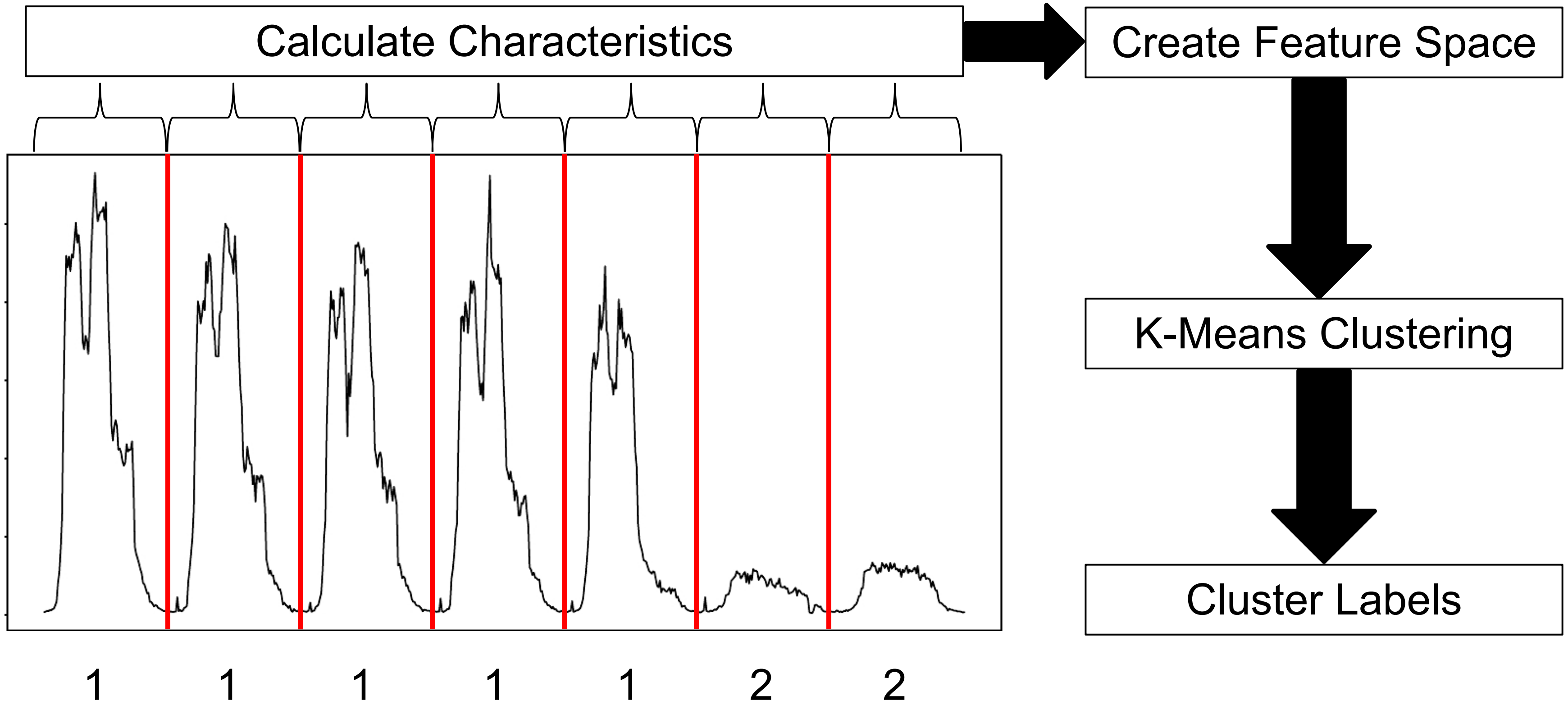
```r
# load package
library(forecast)


timeseries <- AirPassengers
# get top 10 frequencies
pgram <- spec.pgram(as.vector(timeseries))
pgram_df <- data.frame(freq = pgram$freq, spec = pgram$spec)
freqs <- head(1/pgram_df[order(pgram_df$spec, decreasing = TRUE),1],n=10)


# build multi-seasonal time series
mts <- msts(timeseries, seasonal.periods = freqs, ts.frequency =
    frequency(timeseries))
```

Fourier Terms – Cont'd

```r
# get Fourier terms
fourierterms <- fourier(mts, K = rep(1,length(freqs)))

# plot Fourier terms
plot(fourierterms[,1], type='l')
for(i in 2:20){
  readline(prompt="Press [enter] to continue")
  lines(fourierterms[,i], col=i)
}

# continue Fourier Terms
future.fourierterms <- fourier(mts, K = rep(1,length(freqs)), h = 30)
```

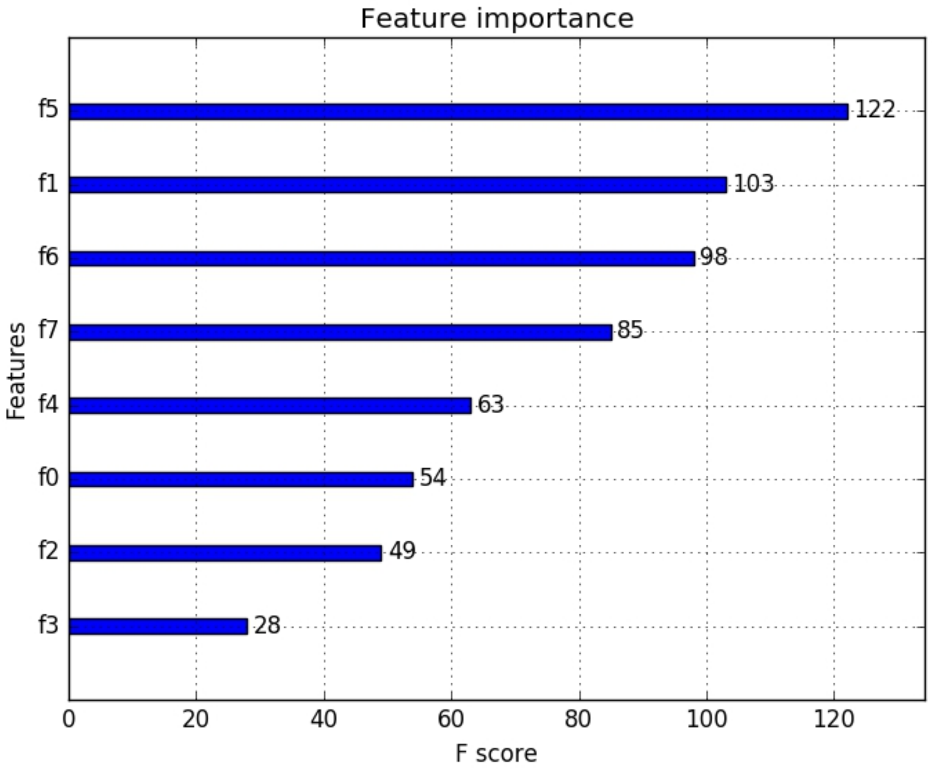
Categorial Information

- Idea: cluster periods of time series
  - Split time series into periods
  - Calculate for each period statistical characteristics



Feature Selection

- Goal: reduce the number of features
  - Preventing from overfitting
  - Speed up training/prediction time
- Statistical feature selection
  - Correlation, anova, …
- Model-internal feature selection
  - Linear models, tree-based models
- Wrapper methods
  - Forward selection, backward elimination



Forward Selection Exhausting Search

```
# load libraries
library(forecast)
library(ggm)

timeseries <- AirPassengers
split <- ceiling(length(timeseries)*0.8)
end <- length(timeseries)

# get top 3 frequencies
pgram <- spec.pgram(as.vector(timeseries))
pgram_df <- data.frame(freq = pgram$freq, spec = pgram$spec)
freqs <- head(1/pgram_df[order(pgram_df$spec, decreasing = TRUE),1],n=3)
```

Introduction
Data Pre-Processing
Feature Engineering
Method Selection
Model Fitting
Evaluation
Summary

Forward Selection – Cont'd

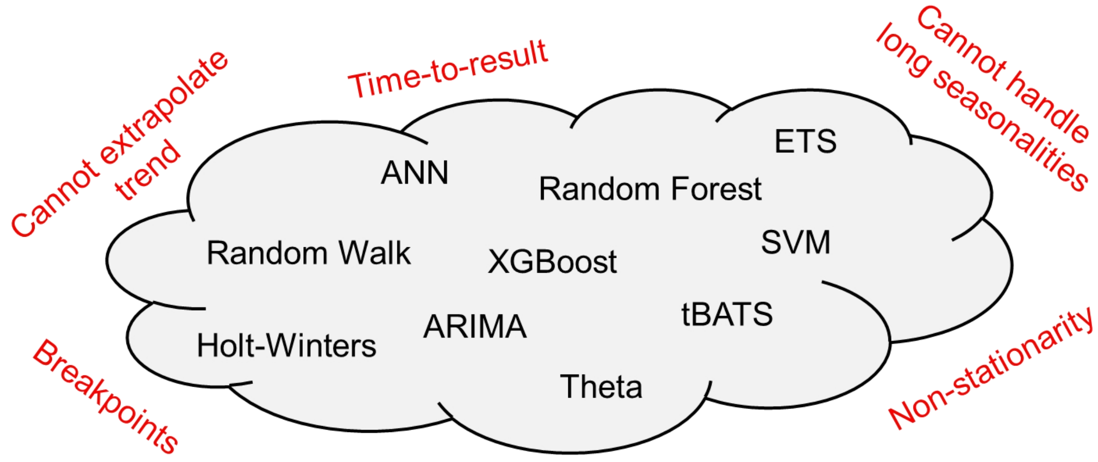
```r
# build multi-seasonal time series
mts <- msts(timeseries, seasonal.periods = freqs,
            ts.frequency = frequency(timeseries))

# decompose time series
decomp <- stl(timeseries, s.window = 'periodic')

# get Fourier terms
fourierterms <- fourier(mts, K = rep(1,length(freqs)))


features <- cbind(timeseries,fourierterms,decomp$time.series[,1:2])

# get powerset of featuer combinations
feature.powerset <- powerset(1:ncol(features))
```

Forward Selection – Cont'd

```r
acc <- c()

# wrapper with exhausting search
for(i in 1:length(feature.powerset)){
  feature.set <- as.matrix(features[,feature.powerset[[i]]])
  model <- nnetar(timeseries[1:split], xreg = feature.set[1:split,])
  fc <- forecast(model, xreg = feature.set[(split+1):end,])
  # get MASE based on validation data
  acc[i] <- accuracy(fc, timeseries[(split+1):end])[12]
}

# get features with lowest MASE
best.set <- features[,feature.powerset[[which(acc == min(acc))]]]
```

Method Selection

- There exist many different forecasting methods
  - Statistical methods
  - Machine learning-based methods



- "No-Free-Lunch Theorem"
  - There is no globally best performing forecasting method
  - Each method has its benefits and drawbacks

- We need additional knowledge on which forecasting method to choose for a particular type of time series

Strength & Weaknesses

Method	Strengths	Weaknesses
sNaïve	+ almost no run-time + very easy to use and intuitive forecast	– provides no useful values for multi-step-ahead forecasting – captures no trend
Theta	+ good for time series with a strong trend	– cannot handle long or multiple seasonalities very well
ETS	+ good for time series with a strong trend + good for detecting sinus-like seasonal patterns	– cannot handle long or multiple seasonalities very well – requires positive values
sARIMA	+ can handle non-stationary time series + option to automatically estimate parameters	– unpredictable and high run-time for model training – insights are limited to parameters
tBATS	+ can handle complex seasonal patterns	– requires positive values
ANN	+ can detect non-linear patterns + data-driven approach	– tends to overfitting of training data – training often computationally expensive
XGBoost	+ fast run-time + accurate method	– cannot handle trend data very well – requires many hyper-parameter settings
Random Forest	+ identifies correlations between features and performance + integrates overfitting prevention	– has poor explainability of the result – cannot extrapolate trend data very well
SVM	+ use mathematical models to prevent overfitting + is robust to small data sets	– is highly sensitive to hyper-parameter settings – training often computationally expensive

How to select a proper forecasting method?

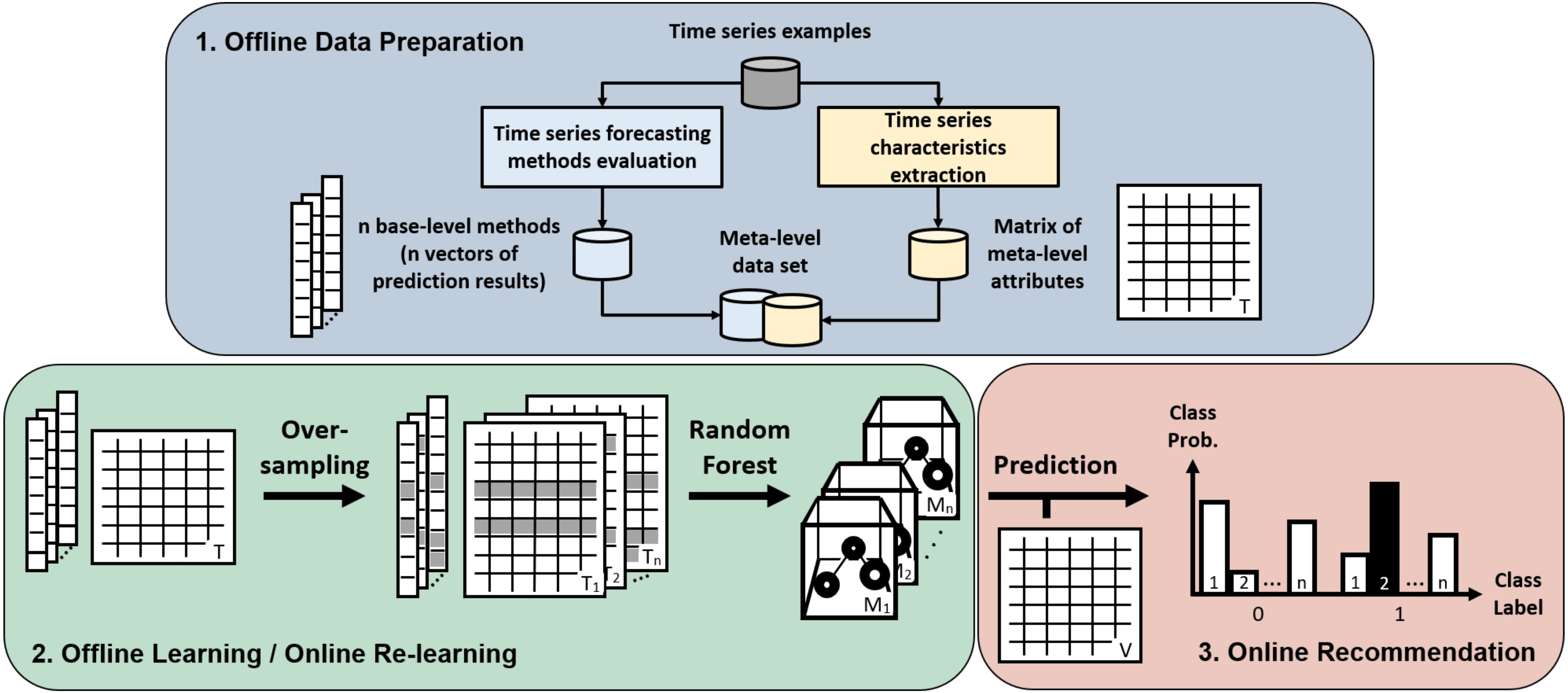
Expert Knowledge	Static Decision Rules	Dynamic Recom. System
Advantages: • No implementation overhead Drawbacks: • Expensive • Does not scale with increasing amount of time series • Decision often cannot be explained objectively	Advantages: • Scale with increasing amount of time series • Expert knowledge only required in design time Drawbacks: • Cannot adapt to new conditions • Does not gain knowledge over time	Advantages: • New rules are learned over time • Ability to adapt to new conditions Drawbacks: • More complex techniques • Implementation required

Static Rules for Method Selection

- Calculate time series characteristics
  - Seasonality
  - Trend
  - Skewness
  - Non-Linearity
  - Chaos
  - ...

- Define simply rules based on expert knowledge
  - IF (Seasonality > 0.15): Do not use ETS
  - IF (Skewness > 0.70 && Non-Linearity < 0.20): Use ARIMA
  - …

Dynamic Recommendation System



Model Fitting

- Fitting forecasting models in R is very easy since there are many libraries existing:
  - forecast
  - xgboost
  - randomForest
  - e1071

- Parameter optimization:
  - Most statistical forecasting models do not require parameter optimization or it is included in the provided implementation
  - Machine-learning based forecasting methods highly depend on parameter optimization ➔ very time-consuming

Model Fitting

```r
library(forecast)


history <- ts(train, frequency = freq)
# sNaive
fc <- snaive(history, h = horizon)
# sARIMA
fit <- auto.arima(history, stepwise = TRUE)
fc <- forecast(fit, h = horizon)
# ETS
fit <- ets(history)
fc <- forecast(fit, h = horizon)
# tBATS
fit <- tbats(history)
fc <- forecast(fit, h = horizon)
# ANN
fit <- nnetar(history)
fc <- forecast(fit, h = horizon)
```

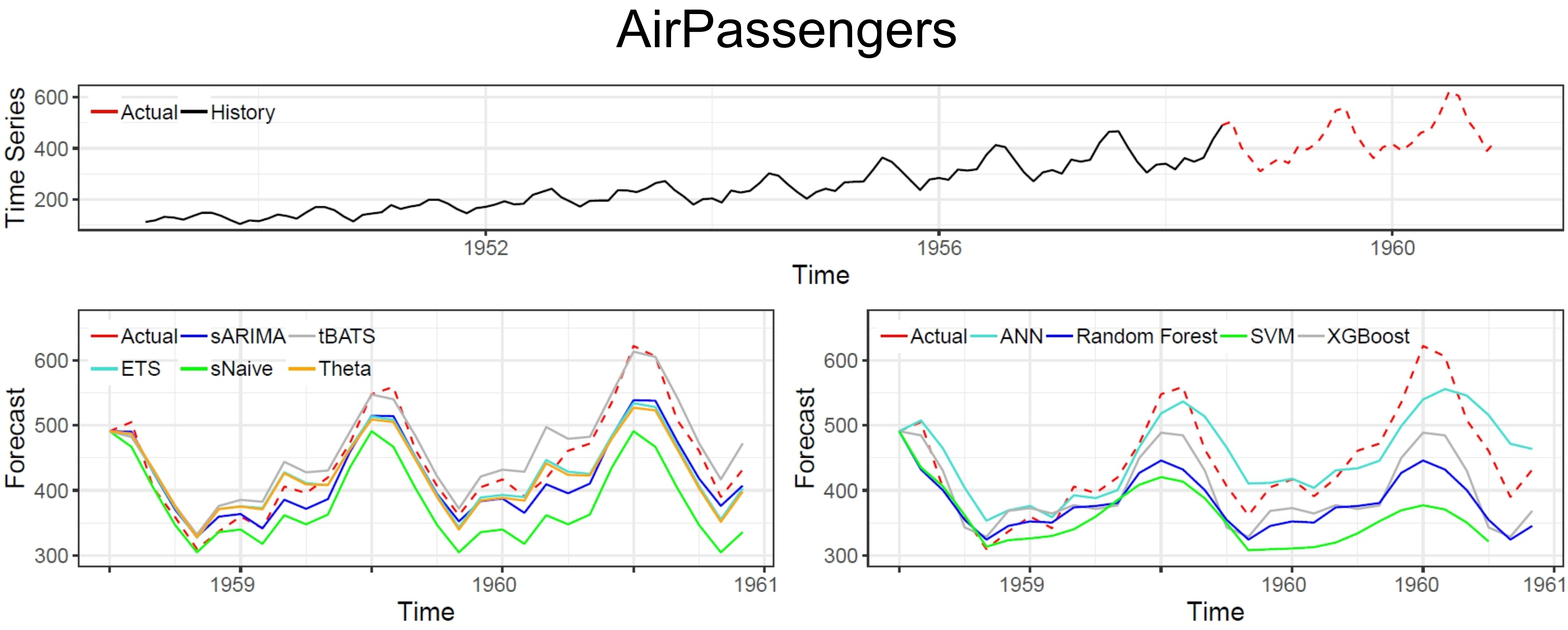
Model Fitting – Cont'd

```r
# used libraries
library(xgboost)
library(randomForest)
library(e1071)

# setting parameters
freq <- frequency(AirPassengers)
horizon <- 14
train <- ts(AirPassengers[1:130],frequency = freq)
len <- length(train)

# used for method training and prediction
ind <- seq(1,length(train))
period <- seq(1,length(train)) %% freq
covar <- as.matrix(cbind(ind, period))
```

Model Fitting – Cont’d

```r
ind <- seq(len+1,len+horizon)
period <- seq(len+1,len+horizon) %% freq
future <- as.matrix(cbind(ind, period))

# XGBoost
fit <- xgboost(label = train, data = covar, nround = 10, nthread = 2)
fc <- predict(fit, future)
# Random Forest
fit <- randomForest(y = train, x = covar)
fc <- predict(fit, future)
# SVM
fit <- svm(y = train, x = covar)
fc <- predict(fit, future)
```

Model Fitting



Evaluation

- Assessing forecast performance is a very important task
- Model error
  - Build model
  - Calculate residuals based on history
- Forecast error
  - A-posteriori
    - Comparison against the “future” values
    - Mostly not available
  - A-priori
    - Split time series into train and test set
    - Commonly 80% and 20%

Error Measure Categories

- Scale-dependent error measures
  - Intuitively while knowing the scale
  - Not suitable for different scales

- Percentage error measures
  - Easy to interpret
  - Scale has impact

$$\frac{12}{10} \gg \frac{10002}{10000}$$

- Scaled error measures
  - Normalization with baseline → scale independent
  - Less intuitive to understand

Error Measure Examples

- $MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |y_i - x_i|$
- $RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - x_i)^2}$

Scale-dependent error measure

- $MAPE = \frac{100\%}{n} \cdot \sum_{i=1}^{n} \left|\frac{y_i - x_i}{x_i}\right|$
- $sMAPE = \frac{200\%}{n} \cdot \sum_{i=1}^{n} \left|\frac{y_i - x_i}{y_i + x_i}\right|$

Percentage error measure

- $MASE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{\frac{n}{n-f} \cdot \sum_{i=f+1}^{n} |x_i - x_{i-f}|}$

Scaled error measure

- …

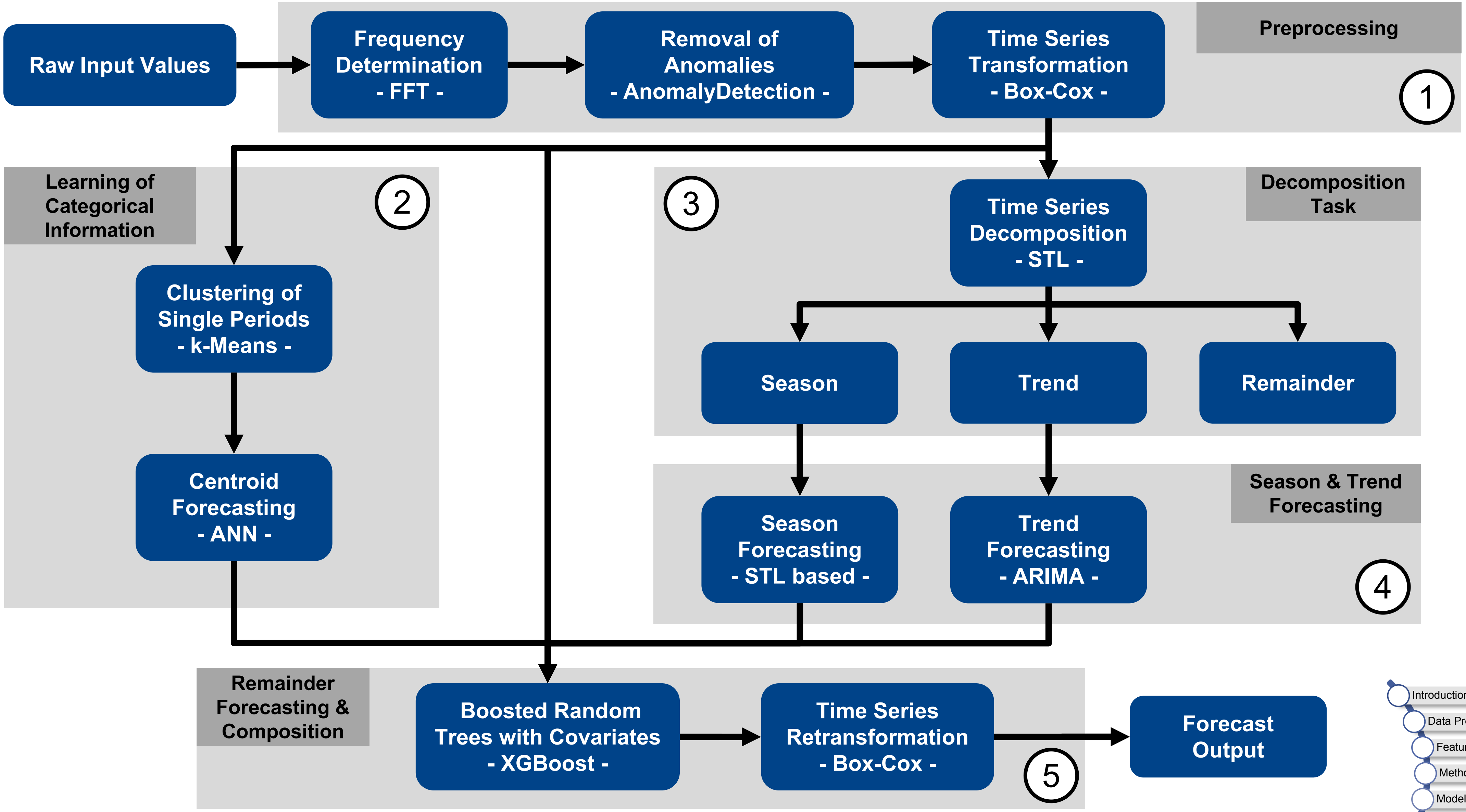
Evaluation

```
# used library
library(forecast)

model <- auto.arima(ts(AirPassengers[1:130],
                       frequency = 12))
fc <- forecast(m, h = 14)

accuracy(fc)
                    ME     RMSE      MAE     MPE    MAPE    MASE    ACF1
Training set 0.44932  9.87073  7.45597  0.0858 2.88924 0.24895 0.01638

accuracy(fc, AirPassengers[131:144])
                    ME     RMSE      MAE     MPE    MAPE    MASE    ACF1
Training set  0.44932  9.87073  7.45597  0.0858 2.88924 0.31360 0.01638
Test set      0.73502 15.17562 11.14010 -0.0154 2.45400 0.46856      NA
```

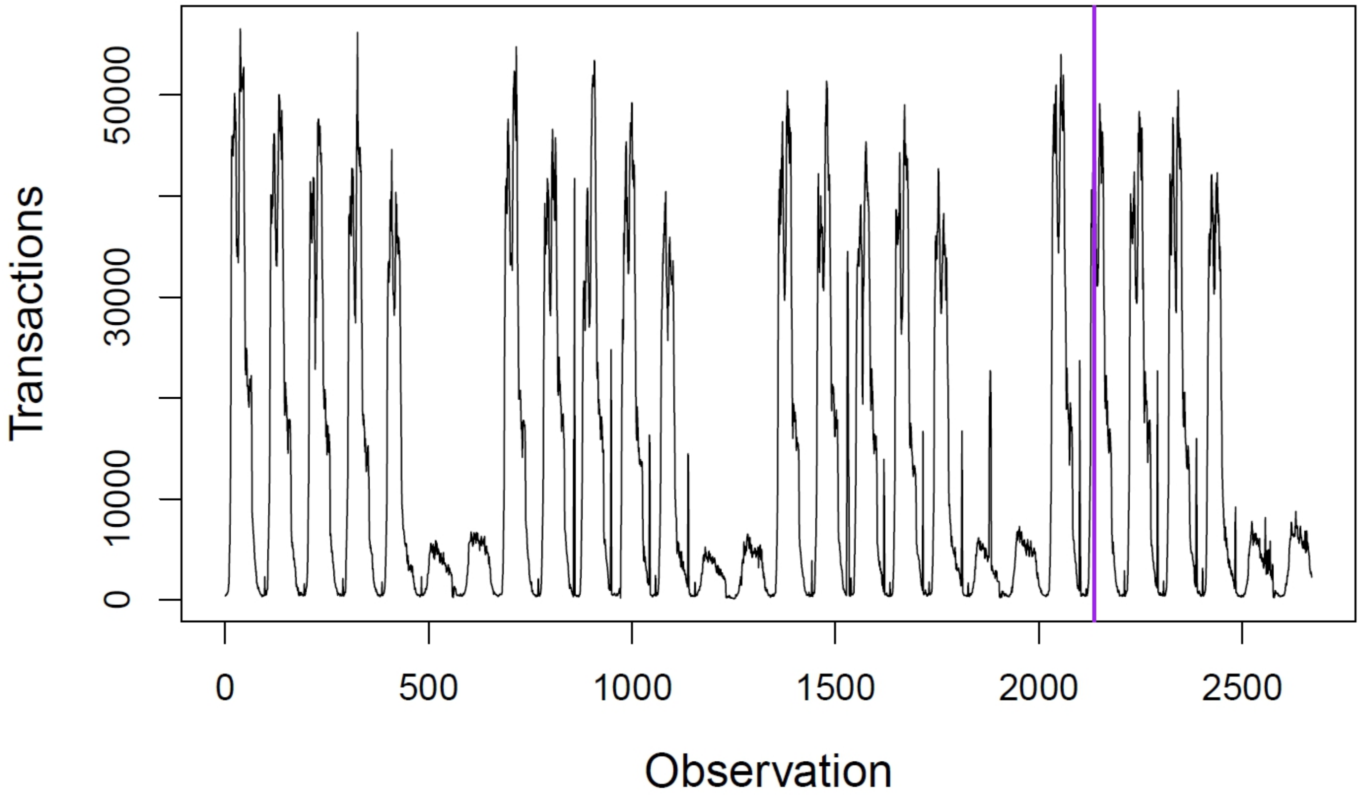
Comparing Forecasts

- Be careful when aggregating forecast error measures
  - Varying scales of different time series
  - Different treatment of positive and negative errors
- How to aggregate forecast error measures?
  - Keep the forecast horizon equally long
  - Use scaled error measures
  - Normalize the range of time series

Putting it together

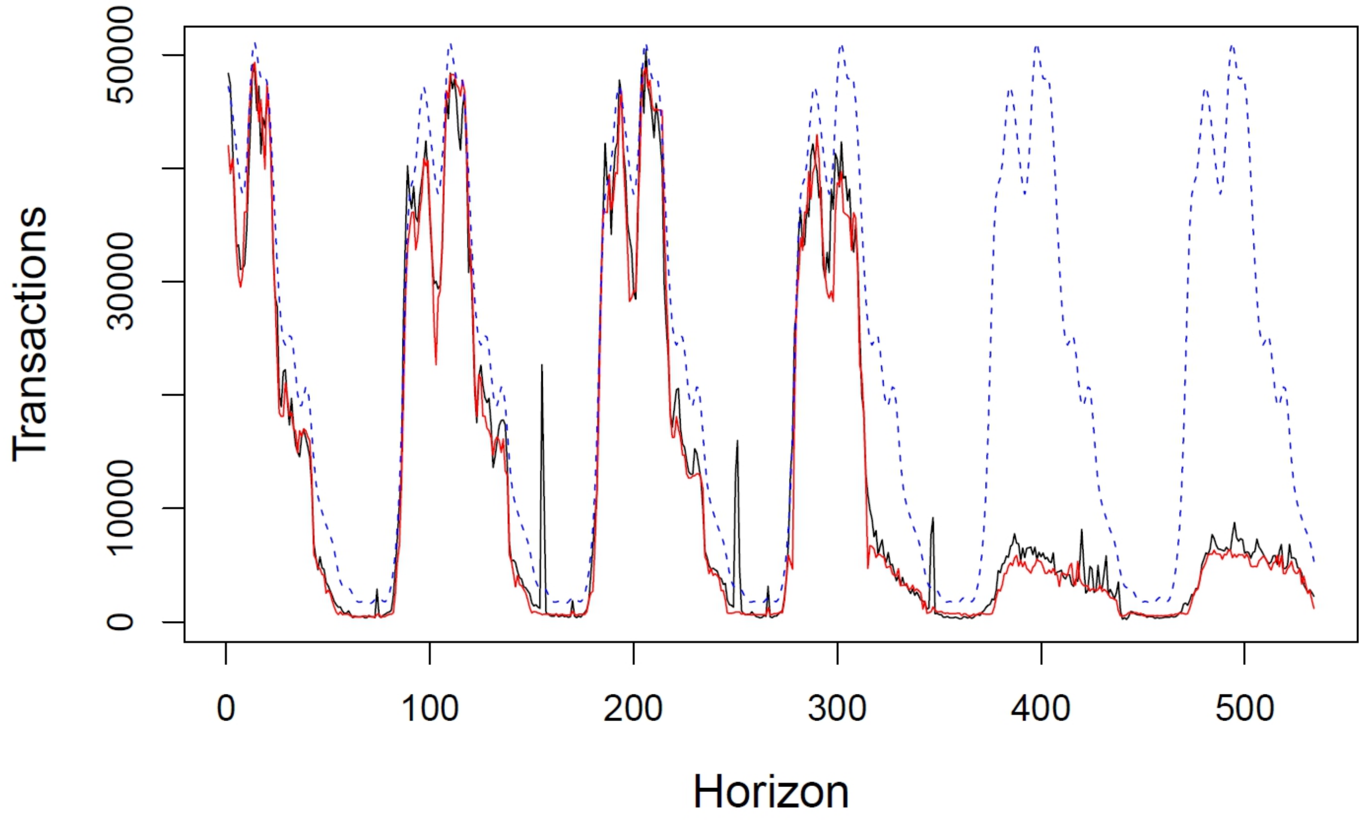


Telescope



Actual values
Left of purple line used for learning
right of purple line to be predicted

Telescope



Actual values
Telescope
tBATS

Telescope

```r
install.packages("devtools")
devtools::install_github("DescartesResearch/telescope")


# Alternative:
install.packages("remotes")
remotes::install_url(url="https://github.com/DescartesResearch/
                                        telescope/archive/master.zip",
                                        INSTALL_opt= "--no-multiarch")


# Loading the library
library(telescope)
# Example execution
forecast <- telescope.forecast(AirPassengers, horizon = 10)
```

Summary

- Forecasting is an important task for many autonomic systems
- Many existing libraries providing easy-to-use functions
- Preprocessing is always needed
- Feature engineering is essential for achieving accurate forecasts
- The error measure should be carefully selected, taking into account the properties of the aggregation