COLUMBIA UNIVERSITY
School of Professional Studies

Applied Analytics: Frameworks and Methods 2

Time Series Analysis

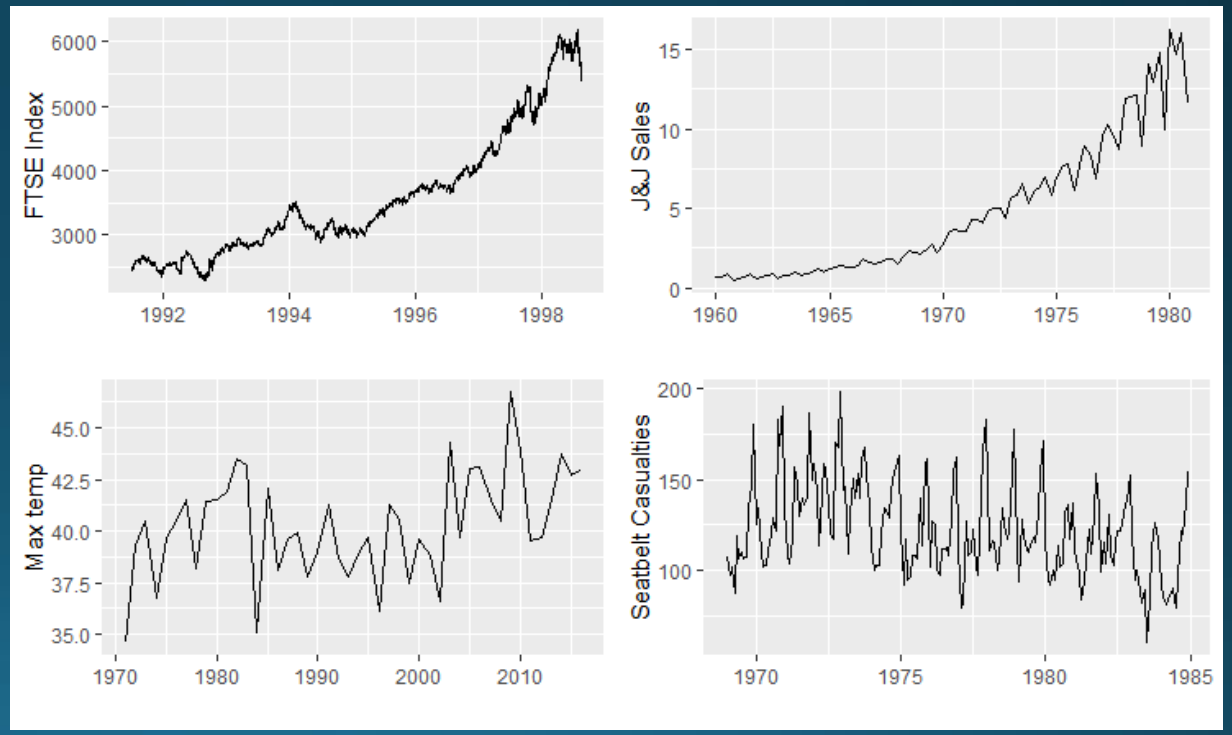
Outline

- Examine time-series data
- Explore and visualize time-series data
- Review basic/simple forecasting methods
- Explore exponential smoothing
- Examine ARIMA models
- Construct forecasts using simple methods, exponential smoothing, and ARIMA.

Time Series Data: Examine, Explore, and Visualize

Time Series

- A series of data in order of time
- A lot of data around us is measured over a span of time
  - Stock prices
  - Federal funds rate
  - Sales
  - Weather
- Time-series data is indexed on time.



Simplifying Assumptions

We make some simplifying assumptions about time-series

- Consecutive observations are equally spaced
- Apply a discrete-time index (even though this may only hold approximately)
  - E.g., daily stock returns are only available for weekdays, not weekends and holidays.

Time Series Data

- Indexed by time/date

- We use special data structures
  - ts
  - xts (stands for eXtensible time series): matrix with a time index
  - zoo

- Use of these data structures offer access to useful functions for manipulating, analyzing and visualizing time-series data.

Time Series Patterns

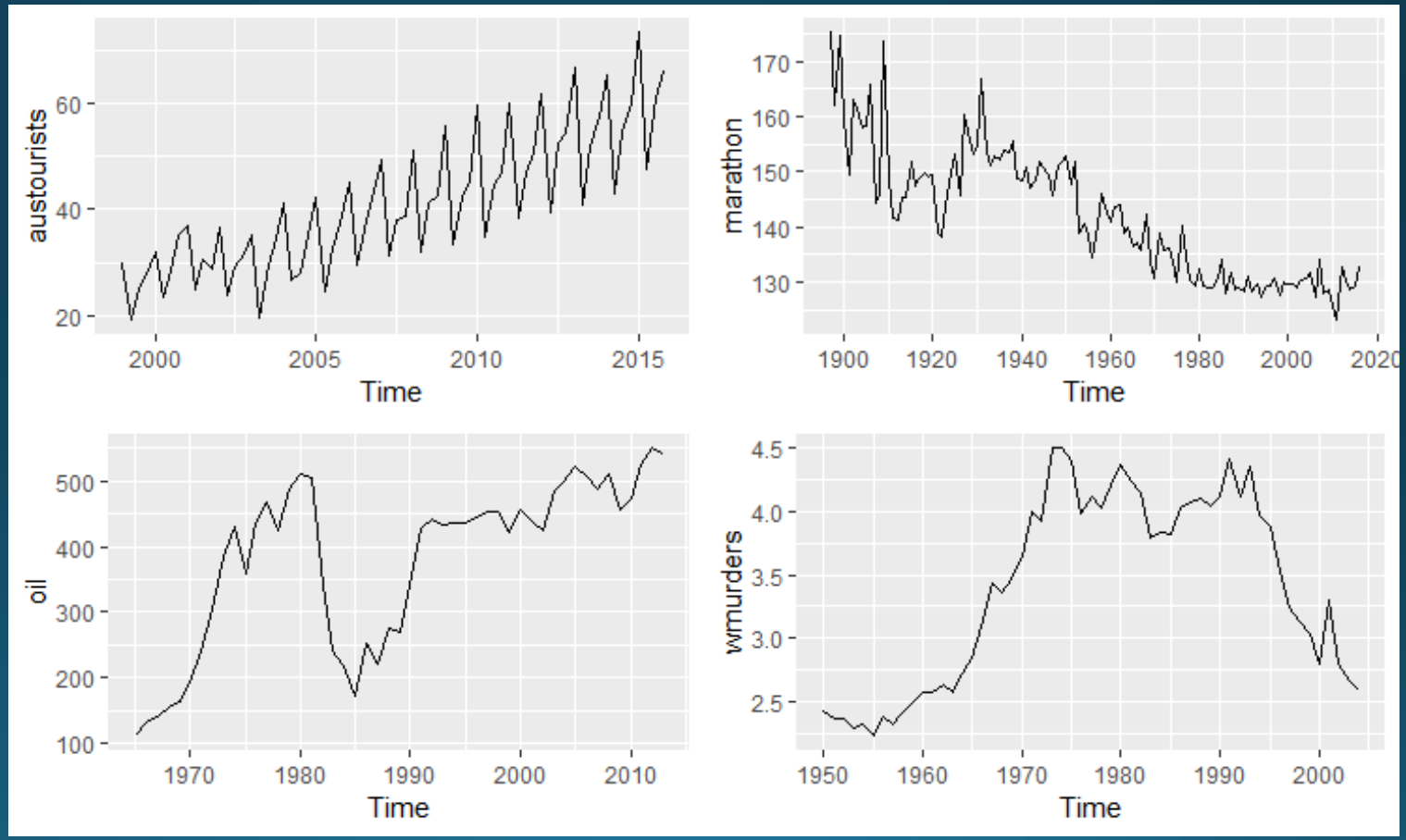
- A common observation across time-series is that prior observations influence later observations.
- The influence may be from the previous observation (trend), an observation that is a fixed number of periods ago (seasonality), or an observation that is not a fixed number of periods ago (cycle).
- Influence of prior observations can be examined by looking at lagged correlations or autocorrelations.

Time Series Patterns

- Trend
  - A long term increase or decrease in the data. Could be linear, non-linear, and may go from an increasing trend to a decreasing trend
- Seasonal
  - Pattern repeats itself every season. Here season may be period of the year or day of the week. Seasonality is always of a fixed or known frequency
- Cyclic
  - Data exhibit rises or falls that are not of a fixed frequency
- On the other hand, a time-series data that does not exhibit any pattern is a Stationary Process. White Noise is the simplest Stationary Process.

Time Series Patterns

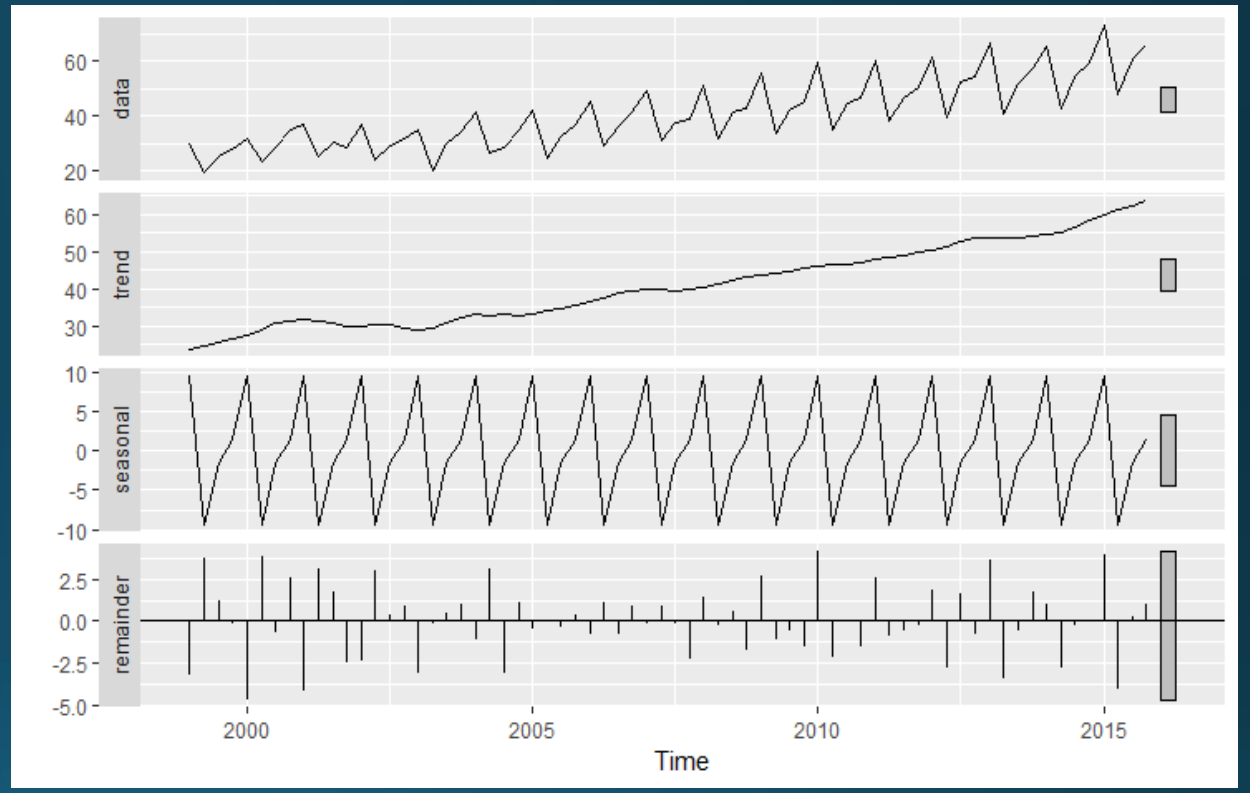
- Which is the *dominant* time series pattern in these charts?
  - Trend
  - Seasonal
  - Cyclic



Time Series Components

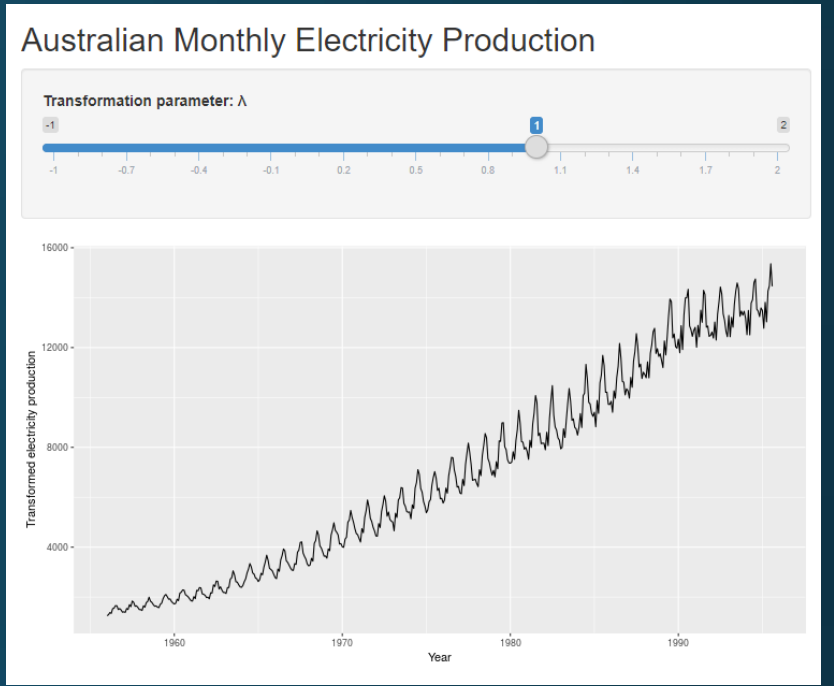
- Time-series may have multiple components in varying strengths. To better understand the series it is useful to divide it into its components: :
  - Trend components
  - Seasonal component
  - Remainder
- Assume an additive decomposition, then we can write

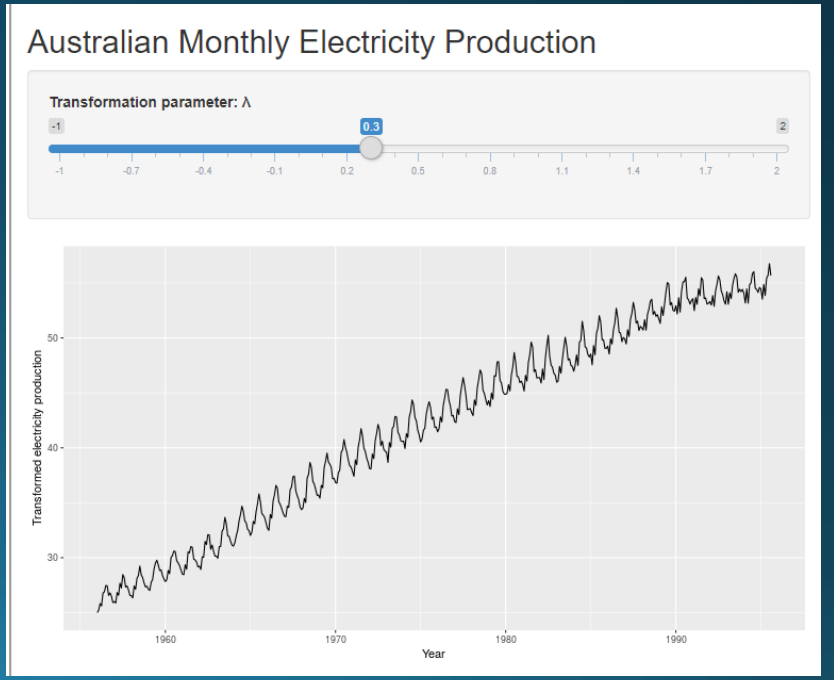
  $y_t = T_t + S_t + R_t$
- Seasonality and Trend Decomposition using Loess (STL) is a versatile and robust method for decomposing a time series
- Decomposition can be used to measure the strength of trend and seasonality



Transformations and Adjustments

- Calendar adjustments
  - Sometimes variation seen in seasonal data may be due to a simple calendar effect such as differences in number of business days in a month.
- Population adjustments
  - A simple adjustment for population is to look at per-capita measures
- Inflation adjustments
  - Data involving money could be adjusted for changes in the time value of money
  - Adjust for inflation, CPI, etc.
- Mathematical transformations
  - When data shows change in variance with time, the variance can be stabilized by a suitable transformation. A common transformation is to use log. A family of transformations that include both log and power transformations are Box-Cox transformations which depend on the parameter lambda.
  - A good value of λ is one which makes the size of the seasonal variation about the same across the whole series





Time Series Regression

- Forecast time series of interest with another time-series
- Important to check for assumptions
  - Model is a reasonable approximation of reality
  - The errors must
    - have a mean of 0
    - are not auto-correlated
    - unrelated to predictors
  - Useful properties of errors:
    - be normally distributed
    - have constant variance (no heteroscedasticity)

Forecasts

- Predictions with time-series data are called forecasts
- Of course, not all forecasts are derived from time-series data. Judgmental forecasts are often used when
  - There is no historical data
  - Data is incomplete or available after a delay
- Three approaches to incorporating judgmental forecasts
  - Rely entirely on judgment when there is no available data
  - When data are available, generate statistical forecasts and adjust using judgment
  - When data are available, generate statistical and judgmental forecasts independently and combine the two.

Judgmental Forecasts

- Delphi method
  - the aim of the Delphi method is to construct consensus forecasts from a group of experts in a structured iterative manner
- Forecasting by analogy
  - example is the pricing of a house through an appraisal process
- New product forecasting
  - Sales force composite, executive opinion, customer intentions

Basic Forecasting Methods

Simple/Basic Forecasting Methods

- Average Method
  - A very simple prediction, often the baseline in linear regression, is to use the average.
- Naïve Method
  - Future will be the same as the last observation
  - Since this is the best prediction for a random walk, these are also called random walk forecasts.
- Seasonal Naïve Method
  - Forecast is equal to the last observed value from the same season.
- Drift Method
  - Allow forecasts to increase or decrease over time, where the amount of change over time (called the drift) is set to be average change seen in historical data.

Exponential Smoothing Models

Exponential Smoothing

- Forecasts are weighted averages of past observations with the weights decaying exponentially such that recent observations get weighted more than distant observations.

Types

- Simple exponential smoothing
  - Forecasts are calculated using weighted averages, where the weights decrease exponentially. Most recent observations get the heaviest weight.
  - Simplest of exponential smoothing methods.
  - Suitable for forecasting data with no clear trend or seasonal pattern.

- Holt's Method
  - Extends simple exponential smoothing to allow the forecasting of data with a trend.

- Holt's Method with Damping
  - Forecasts generally display a constant trend (increasing or decreasing) indefinitely into the future. For this reason, a damping parameter is usually included.

Types

- Holt-Winter's seasonal method
  - Extends Holt's method to capture seasonality. Two types:
    - Additive method is used when seasonal variations are roughly constant
    - Multiplicative method is used when seasonal variations change in proportion to the level of the series

- But, exponential smoothing methods are not limited to these. By considering variations of trend (none, additive, additive damped) and seasonal components (non, additive, multiplicative), there are nine exponential smoothing methods

- For each of these methods, errors may be additive or multiplicative.

Exponential Smoothing Models: A Taxonomy

Trend Component	Seasonal Component		
	N	A	M
	(None)	(Additive)	(Multiplicative)
N (None)	(N,N)	(N,A)	(N,M)
A (Additive)	(A,N)	(A,A)	(A,M)
A_d (Additive damped)	(A_d,N)	(A_d,A)	(A_d,M)

Some of these methods we have already seen using other names:

Short hand	Method
(N,N)	Simple exponential smoothing
(A,N)	Holt's linear method
(A_d,N)	Additive damped trend method
(A,A)	Additive Holt-Winters' method
(A,M)	Multiplicative Holt-Winters' method
(A_d,M)	Holt-Winters' damped method

State Space Models

- Each model consists of
  - a measurement equation that describes the observed data and
  - some state equations that describe how the unobserved components or states (level, trend, seasonal) change over time.

- Hence, these are referred to as State Space Models

Table 7.7: State space equations for each of the models in the ETS framework.

ADDITIVE ERROR MODELS

Trend	Seasonal N	Seasonal A	Seasonal M
N	$y_t = \ell_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$	$y_t = \ell_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/\ell_{t-1}$
A	$y_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + b_{t-1})$
A_d	$y_t = \ell_{t-1} + \phi b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + \phi b_{t-1})$

MULTIPLICATIVE ERROR MODELS

Trend	Seasonal N	Seasonal A	Seasonal M
N	$y_t = \ell_{t-1}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$	$y_t = (\ell_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \alpha(\ell_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + s_{t-m})\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A	$y_t = (\ell_{t-1} + b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A_d	$y_t = (\ell_{t-1} + \phi b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$

ARIMA

ARIMA

- Exponential Smoothing and ARIMA are the two most widely used approaches to time-series forecasting and provide complementary approaches to the problem.

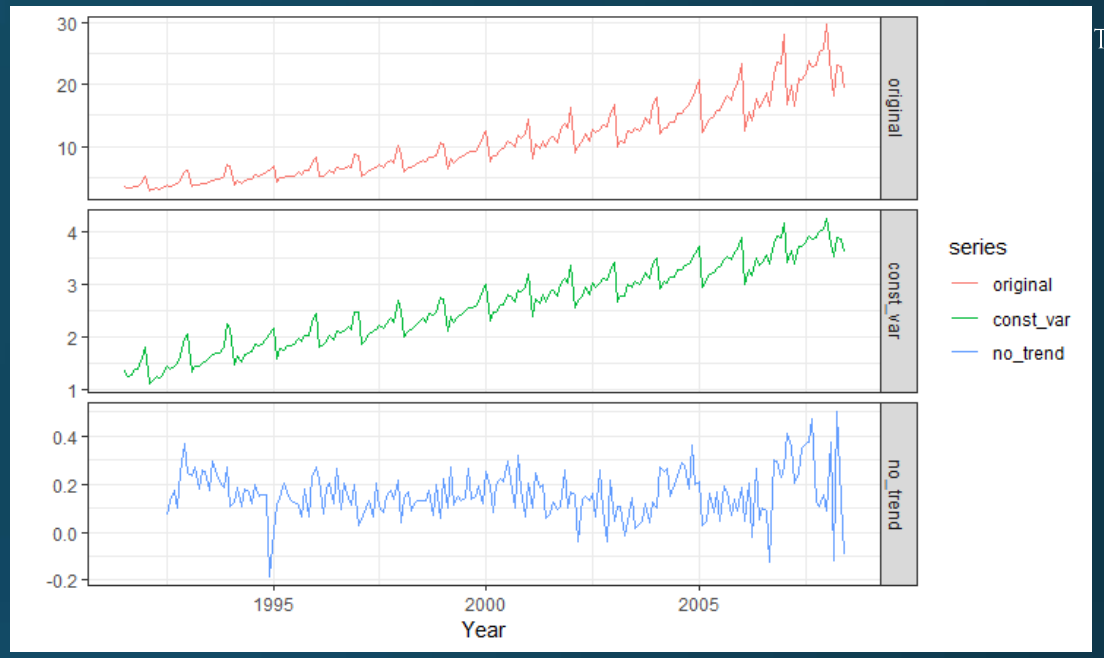
- While Exponential Smoothing models are based on a description of trend and seasonality in the data, ARIMA models aim to describe autocorrelations in the data.

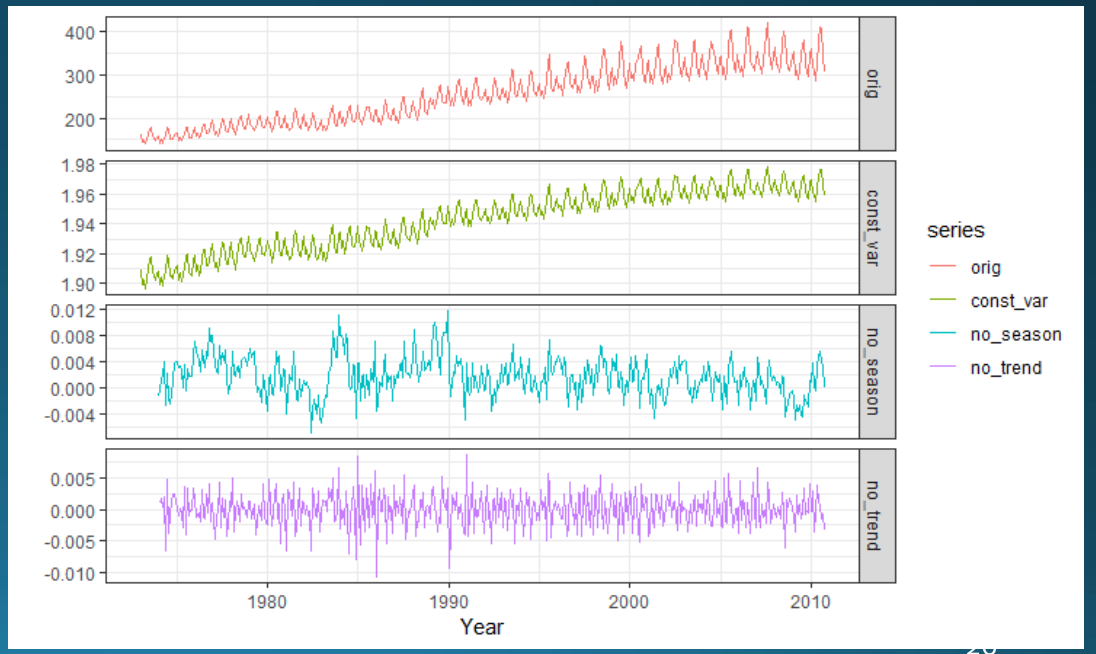
Stationary Process

- A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary
- Stationarity is an assumption of many time-series analysis procedures.
- However, most time-series are non-stationary.
- Unit Root Tests: Statistical Test of Stationarity designed to see if differencing is required. E.g., KPSS Test.

Stationary Process

- Fortunately, a non-stationary process can be transformed into a (weakly) stationary process through transformations that remove trend and stabilize variance.
  - Stabilize variance: Box-Cox Transformation
  - Remove seasonality and trend: Differencing





ARIMA

- AutoRegressive Integrated Moving Average
- ARIMA = AR + I + MA

Differencing (I)

- Differencing can help stabilize the mean of a time series by removing changes in the level of a time series, and so eliminating trend and seasonality.

Autoregressive Models (AR)

- Forecast a variable using a linear combination of past values of the variable
- $y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \varphi_3 y_{t-3} + \ldots. \varepsilon_t$
  - where ε_t is white noise

Moving Average (MA)

- A moving average model uses past forecast errors in a regression-like model
- $y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$
  - where ε_t is white noise

Non-seasonal ARIMA

- $y'_t = c + \varphi_1 y'_{t-1} + \cdots + \varphi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$
  - where y'_t is the differenced series

- ARIMA(p,d,q)
  - p = Number of ordinary Autoregressive lags
  - d = Number of ordinary differences
  - q = Number of ordinary Moving average lags

Seasonal ARIMA

- Non-Seasonal ARIMA + seasonal terms = Seasonal ARIMA

- ARIMA(p,d,q)(P,D,Q)m
  - (P,D,Q) is seasonal component
  - m is seasonal period

  where

  - p = Number of ordinary Autoregressive lags
  - d = Number of ordinary differences
  - q = Number of ordinary Moving average lags
  - P = Number of seasonal Autoregressive lags
  - D = Number of seasonal differences
  - Q = Number of seasonal Moving Average lags
  - m = Seasonal period or number of observations per year

Estimation and Order Selection

- ARIMA model is estimated using maximum likelihood estimation. Technique finds the values of the parameters which maximize the probability of obtaining the data that we have observed

- Information Criteria
  - AIC
  - BIC
  - AICc

- Information are useful for selecting the best values of p and q but not for selecting the appropriate order of differencing (d). Differencing changes the data on which likelihood is computed making models with different d not comparable.

ARIMA Modeling in R

- Automatic Process
  - auto.arima()
  - Uses computational shortcuts, so solution may not be optimal
  - Ensure more thorough examination by
    - stepwise=F
    - approximation=F

ARIMA Modeling in R

- Steps to select ARIMA model

1. Plot data to identify any unusual observations
2. If necessary, transform data to stabilize variance
3. If the data are non-stationary, take first differences until the data are stationary
4. Examine ACF and PACF to decide on AR (ARIMA(p,d,0) and/or MA terms (ARIMA(0,d,q))
5. Try chosen model and use AICc to search for a better model
6. Check residuals by plotting. If they do not look like white noise, try a modified model
7. Once the residuals look like white noise, calculate forecasts

Advanced Time Series Models

- Dynamic regression models
- Hierarchical and grouped time series
- TBATS model for complex seasonality
- Vector autoregressions
- Neural network autoregression

Illustration - R

Packages Used

- About Time Series
- Simplifying Assumptions
- Time Series Classes
- Nature of Time Series
- Time Series Patterns
- Time Series Components
- Simple Forecasting Methods
- Exponential Smoothing Models
- ETS Models
- ARIMA
- Comparing Forecasting Models

Simple Forecasting Methods

- Average Method
- Naive Method
- Seasonal Naive Method
- Drift Method

Exponential Smoothing Models

- Simple exponential smoothing
- Holt's Method
- Holt's Method with Damping
- Holt-Winter's seasonal method
  - Holt_Winter's Additive
  - Holt_Winter's Multiplicative

ETS Models

- ETS: AAA
- ETS: Automatic Selection

ARIMA

- Stationary Process
- ARIMA Model
- ARIMA - Automatic Model Selection

Time Series

Packages Used

```
library(ggplot2);library(ggthemes);library(gridExtra)  # For plots
library(quantmod);library(xts);library(zoo) # For using xts class objects
library(forecast) # Set of forecasting functions
library(fpp); library(fpp2) # Datasets from Forecasting text by Rob Hyndman
library(tseries) # for a statistical test
library(dplyr) # Data wrangling
```

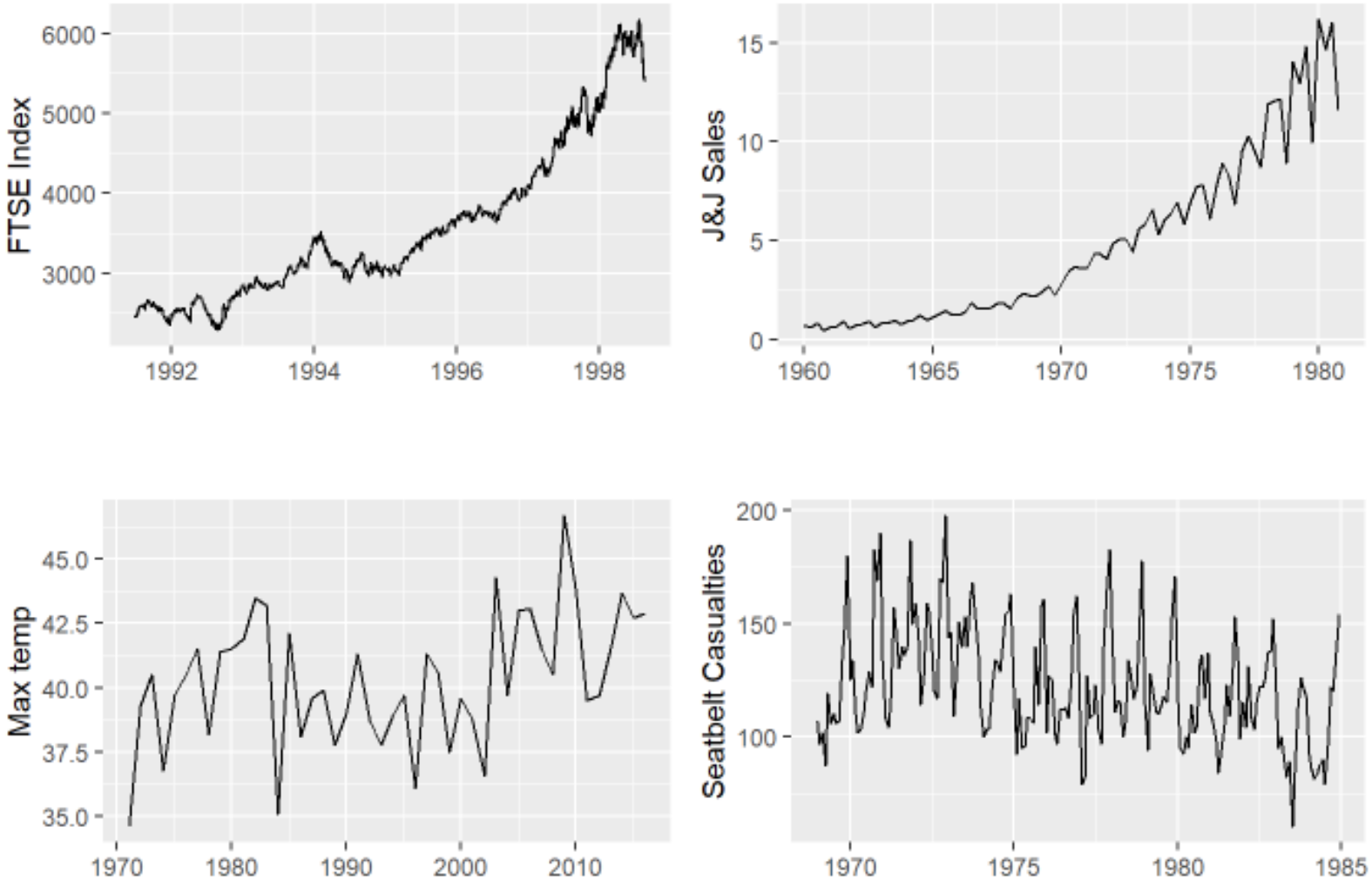
About Time Series

A lot of data around us is measured over a span of time

- Stock prices
- Federal funds rate
- Sales
- Weather

```
# ?EuStockMarkets
# ?JohnsonJohnson
# ?maxtemp
# ?Seatbelts
```

```
library(gridExtra)
grid.arrange(autoplot(EuStockMarkets[,'FTSE'])+xlab('')+ylab('FTSE Index'),
             autoplot(JohnsonJohnson)+xlab('')+ylab('J&J Sales'),
             autoplot(maxtemp)+xlab('')+ylab('Max temp'),
             autoplot(Seatbelts[,'DriversKilled'])+xlab('')+ylab('Seatbelt Casualties'))
```



Console | Terminal | Jobs

```
> View(EuStockMarkets)
> View(JohnsonJohnson)
```

EuStockMarkets | timeSeries Sp2021_kkc.R

Filter

	DAX	SMI	CAC	FTSE
1	1628.75	1678.1	1772.8	2443.6
2	1613.63	1688.5	1750.5	2460.2
3	1606.51	1678.6	1718.0	2448.2
4	1621.04	1684.1	1708.1	2470.4
5	1618.16	1686.6	1723.1	2484.7
6	1610.61	1671.6	1714.3	2466.8
7	1630.75	1682.9	1734.5	2487.9
8	1640.17	1703.6	1757.4	2508.4

JohnsonJohnson

	V1
1	0.71
2	0.63
3	0.85
4	0.44
5	0.61
6	0.69
7	0.92
8	0.55

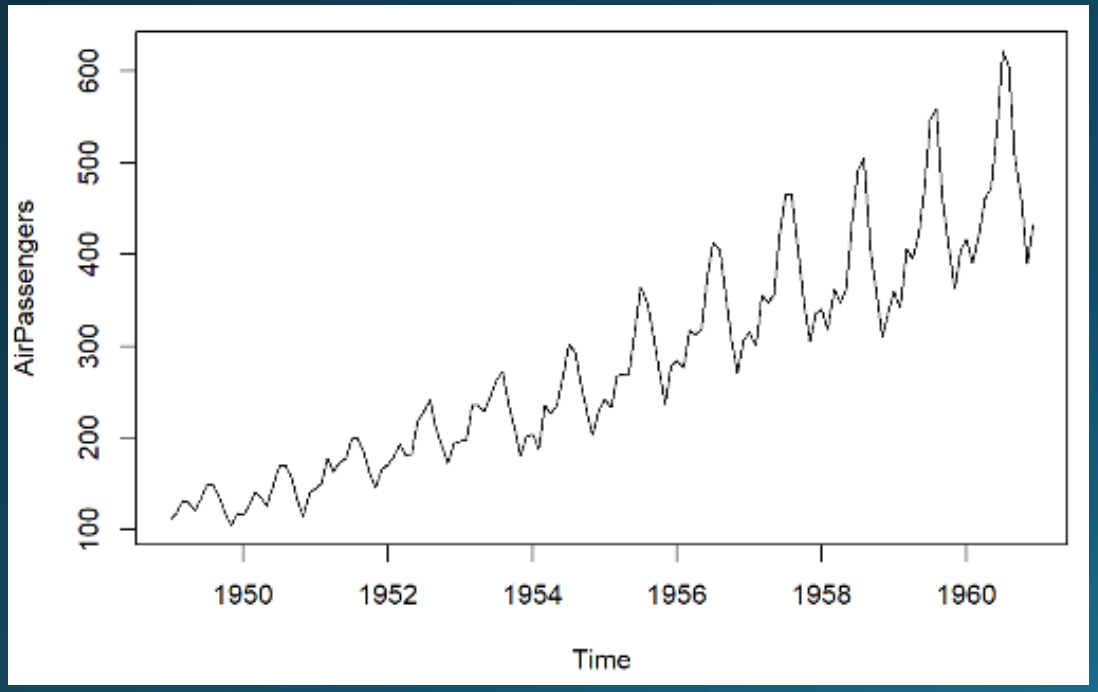
Simplifying Assumptions

We make some simplifying assumptions about time-series

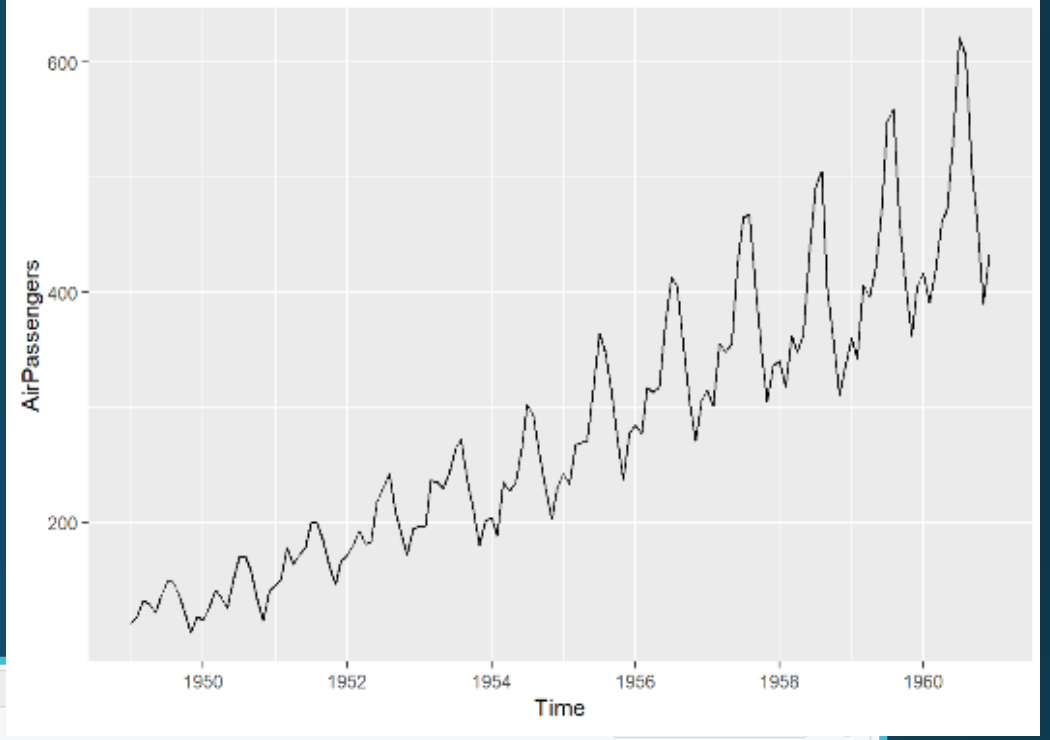
- Consecutive observations are equally spaced
- Apply a discrete-time index (even though this may only hold approximately). E.g., daily stock returns are only available for weekdays, not weekends and holidays.

Lets see if AirPassengers data meets these assumptions

```
plot(AirPassengers)
```




```
autoplot(AirPassengers)
```



Files | Plots | Packages | Help

R: Monthly Airline Passenger Numbers 1949-1960 — Find in Topic

AirPassengers {datasets} — R Documentation

Monthly Airline Passenger Numbers 1949-1960

Description

The classic Box & Jenkins airline data. Monthly totals of international airline passengers, 1949 to 1960.

Usage

```
AirPassengers
```

Format

A monthly time series, in thousands.

Time Series Classes

- Time series data needs a special class because it is indexed by time/date
- Data structures designed for time series data: ts, xts, zoo, ..
- Use of these data structures offers access to useful functions for manipulating, analyzing and visualizing time-series data.

ts methods/functions

Let us take a look at the ts structure first.

```
AirPassengers
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

What type of data structure is AirPassengers?

Class

```
class(AirPassengers)
```

```
## [1] "ts"
```

First period of the time series

```
start(AirPassengers)
```

```
## [1] 1949    1
```

Last Period of the time series

```
end(AirPassengers)
```

```
## [1] 1960   12
```

Period	
1	0.083
2	0.167
3	0.250
4	0.333
5	0.417
6	0.500
7	0.583
8	0.667
9	0.750
10	0.833
11	0.917
12	1.000

time - creates the vector of times at which a time series was sampled.

All time periods

```
time(AirPassengers)
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1949 1949.000 1949.083 1949.167 1949.250 1949.333 1949.417 1949.500 1949.583
## 1950 1950.000 1950.083 1950.167 1950.250 1950.333 1950.417 1950.500 1950.583
## 1951 1951.000 1951.083 1951.167 1951.250 1951.333 1951.417 1951.500 1951.583
## 1952 1952.000 1952.083 1952.167 1952.250 1952.333 1952.417 1952.500 1952.583
## 1953 1953.000 1953.083 1953.167 1953.250 1953.333 1953.417 1953.500 1953.583
## 1954 1954.000 1954.083 1954.167 1954.250 1954.333 1954.417 1954.500 1954.583
## 1955 1955.000 1955.083 1955.167 1955.250 1955.333 1955.417 1955.500 1955.583
## 1956 1956.000 1956.083 1956.167 1956.250 1956.333 1956.417 1956.500 1956.583
## 1957 1957.000 1957.083 1957.167 1957.250 1957.333 1957.417 1957.500 1957.583
## 1958 1958.000 1958.083 1958.167 1958.250 1958.333 1958.417 1958.500 1958.583
## 1959 1959.000 1959.083 1959.167 1959.250 1959.333 1959.417 1959.500 1959.583
## 1960 1960.000 1960.083 1960.167 1960.250 1960.333 1960.417 1960.500 1960.583
##           Sep      Oct      Nov      Dec
## 1949 1949.667 1949.750 1949.833 1949.917
## 1950 1950.667 1950.750 1950.833 1950.917
## 1951 1951.667 1951.750 1951.833 1951.917
## 1952 1952.667 1952.750 1952.833 1952.917
## 1953 1953.667 1953.750 1953.833 1953.917
## 1954 1954.667 1954.750 1954.833 1954.917
## 1955 1955.667 1955.750 1955.833 1955.917
## 1956 1956.667 1956.750 1956.833 1956.917
## 1957 1957.667 1957.750 1957.833 1957.917
## 1958 1958.667 1958.750 1958.833 1958.917
## 1959 1959.667 1959.750 1959.833 1959.917
## 1960 1960.667 1960.750 1960.833 1960.917
```

```
deltat(AirPassengers) # one month is 1/12 of a year
```

```
## [1] 0.08333333
```

deltat - the time interval between observations

```
cycle(AirPassengers)
```

cycle gives the positions in the cycle of each observation.

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949   1   2   3   4   5   6   7   8   9  10  11  12
## 1950   1   2   3   4   5   6   7   8   9  10  11  12
## 1951   1   2   3   4   5   6   7   8   9  10  11  12
## 1952   1   2   3   4   5   6   7   8   9  10  11  12
## 1953   1   2   3   4   5   6   7   8   9  10  11  12
## 1954   1   2   3   4   5   6   7   8   9  10  11  12
## 1955   1   2   3   4   5   6   7   8   9  10  11  12
## 1956   1   2   3   4   5   6   7   8   9  10  11  12
## 1957   1   2   3   4   5   6   7   8   9  10  11  12
## 1958   1   2   3   4   5   6   7   8   9  10  11  12
## 1959   1   2   3   4   5   6   7   8   9  10  11  12
## 1960   1   2   3   4   5   6   7   8   9  10  11  12
```

head and tail work in the same manner as on a data frame

```
head(AirPassengers)
```

```
##      Jan Feb Mar Apr May Jun
## 1949 112 118 132 129 121 135
```

```
tail(AirPassengers)
```

```
##      Jul Aug Sep Oct Nov Dec
## 1960 622 606 508 461 390 432
```

Explore data as an xts object

Convert AirPassengers to an xts object

```
AP_xts <- as.xts(AirPassengers)
```

Number of passengers for June, 1960

```
AP_xts['1960-06',]
```

```
##          [,1]
## Jun 1960  535
```

Average number of passengers for the year 1960

```
mean(AP_xts['1960',])
```

```
## [1] 476.1667
```

Number of months of data are included in this dataset?

```
nmonths(AP_xts)
```

```
## [1] 144
```

Calculate correlation between number of passengers and one-month lagged number of passengers

```
cor(AP_xts,lag.xts(AP_xts),use = 'complete.obs')
```

```
##           [,1]
## [1,] 0.9601946
```

Create a ts object

Lets create a ts object, even though in most cases we will work with data that is already a time series object. To define a ts object, we specify the data, start/end and frequency. Here, frequency is the number of observations per seasonal pattern

- annual is 1 (default)
- quarterly is 4
- monthly is 12
- weekly is 52
- other special ways to handle seasonality less than a week

```
sales_in_millions = sample(seq(1e2,1e4,1)/1e2,size = 96,replace = T)
sales_ts = ts(data=sales_in_millions,start = 2010,frequency = 12)
sales_ts
```

```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
## 2010 19.28  3.79 63.88 94.65 52.21 57.91 59.36 61.65 24.37 28.73 20.64 90.32
## 2011 99.24 28.63 47.02 64.01 87.75 86.45 75.24 47.76 59.31 88.15 74.86 19.15
## 2012 48.07 19.70 46.42 27.58 71.14 67.60 84.03 17.52 99.20 80.80 45.57 67.89
## 2013 45.95 58.94 81.49  4.76 97.67 74.95 60.34 24.48 50.62 31.84 54.23 86.61
## 2014 30.67 69.23 55.78 22.58 97.06 69.66 95.45  9.91 44.06 83.33 57.91 68.62
## 2015 19.67 33.71 74.64 20.33 85.50  9.82 41.50 89.00 19.69 23.14  5.04 43.74
## 2016 17.52 96.59 60.61 21.96 20.08 36.83 31.90 72.53 86.14 96.17 60.69 67.83
## 2017 18.85 10.37 72.92 71.72 82.14 67.02  6.67 11.15 36.68 51.35 49.33  8.92
```

Verify that sales_ts is a ts object

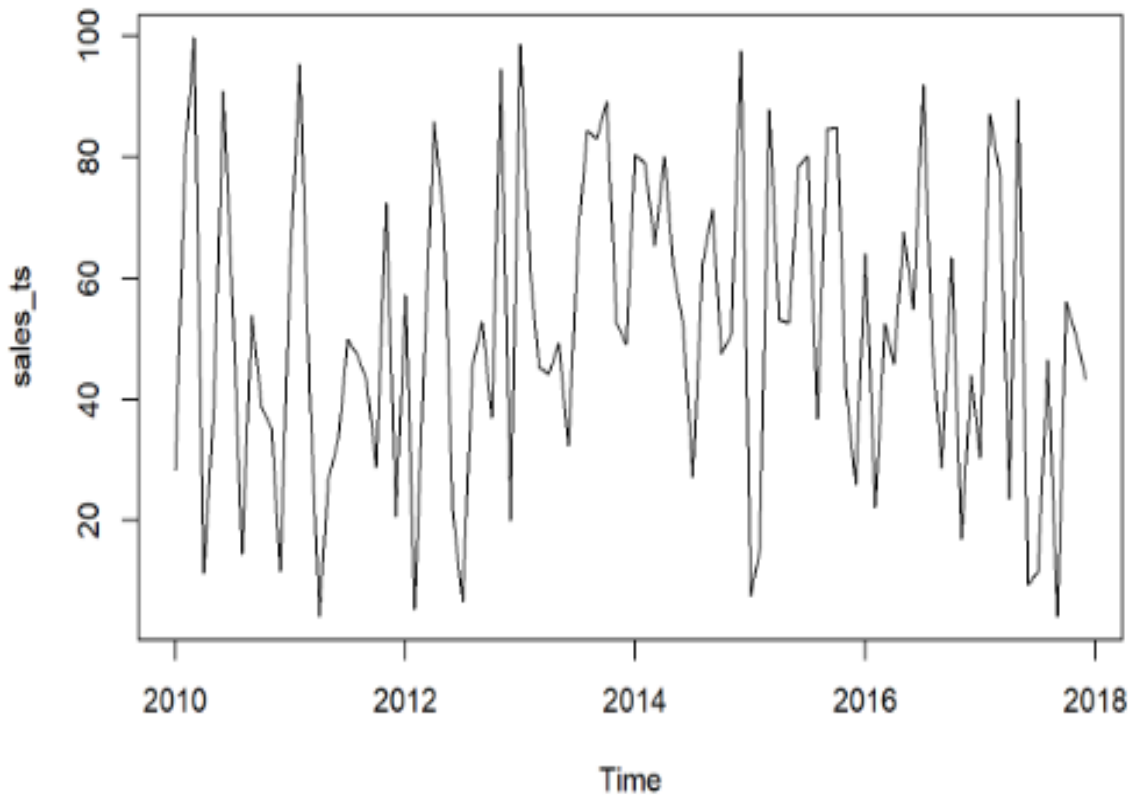
```
is.ts(sales_ts)
```

```
## [1] TRUE
```

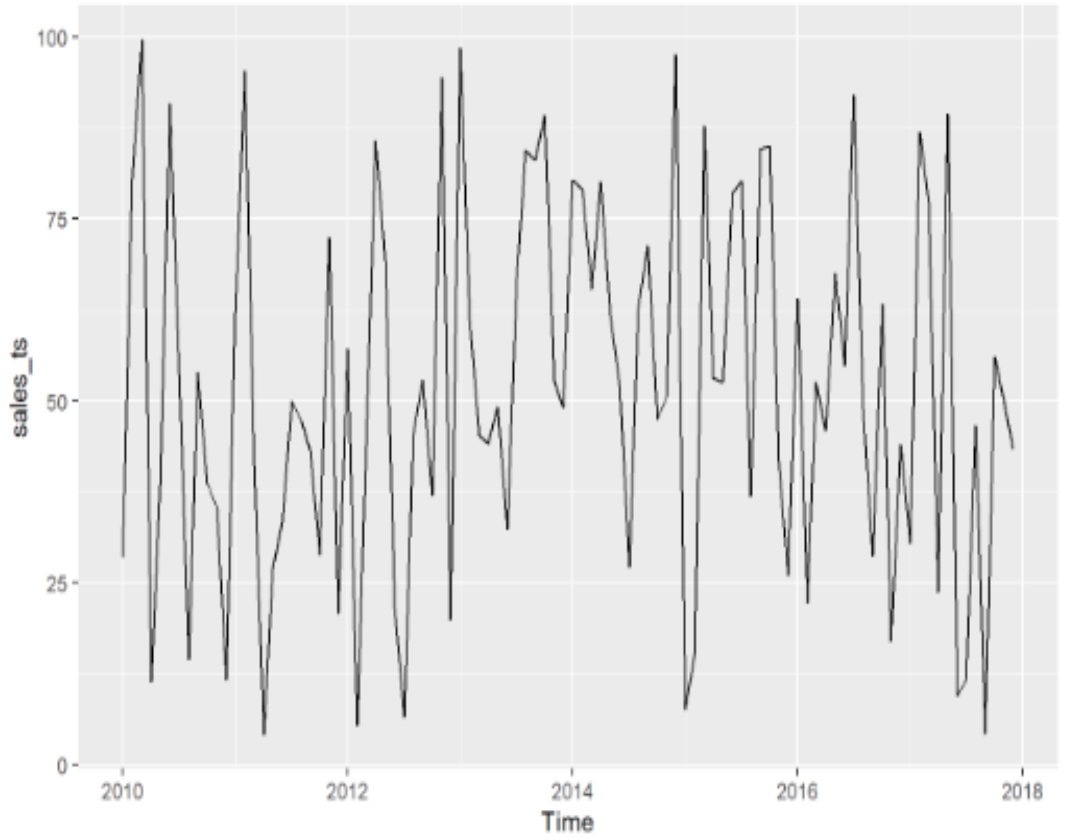
Plotting

ts objects can be plotted by using plot() or using autoplot() from library(ggplot2). The latter will generate a ggplot2 object that generally looks prettier.

```
plot(sales_ts)
```




```
library(ggplot2)
autoplot(sales_ts)
```



Nature of Time Series

A common observation across time-series is that prior observations influence later observations. The influence may be from the previous observation (trend), an observation that is a fixed number of periods ago (seasonality), or an observation that is not a fixed number of periods ago (cycle). Influence of prior observations can be examined by looking at lagged correlations or autocorrelations.

Lag 1 autocorrelation

Use AirPassengers data

```
acf(x = AirPassengers,lag.max = 1,plot=F)
```

1 lag

no lag

```
##
## Autocorrelations of series 'AirPassengers', by lag
##
## 0.0000 0.0833
##  1.000  0.948
```

This is the time stamp

Lag 1 and 2 Autocorrelation

```
acf(x=AirPassengers,lag.max=2,plot=F)
```

```
##
## Autocorrelations of series 'AirPassengers', by lag
##
## 0.0000 0.0833 0.1667
##  1.000  0.948  0.876
```

- Just as correlation measures the extent of a linear relationship between two variables, autocorrelation measures the linear relationship between lagged values of a time series.

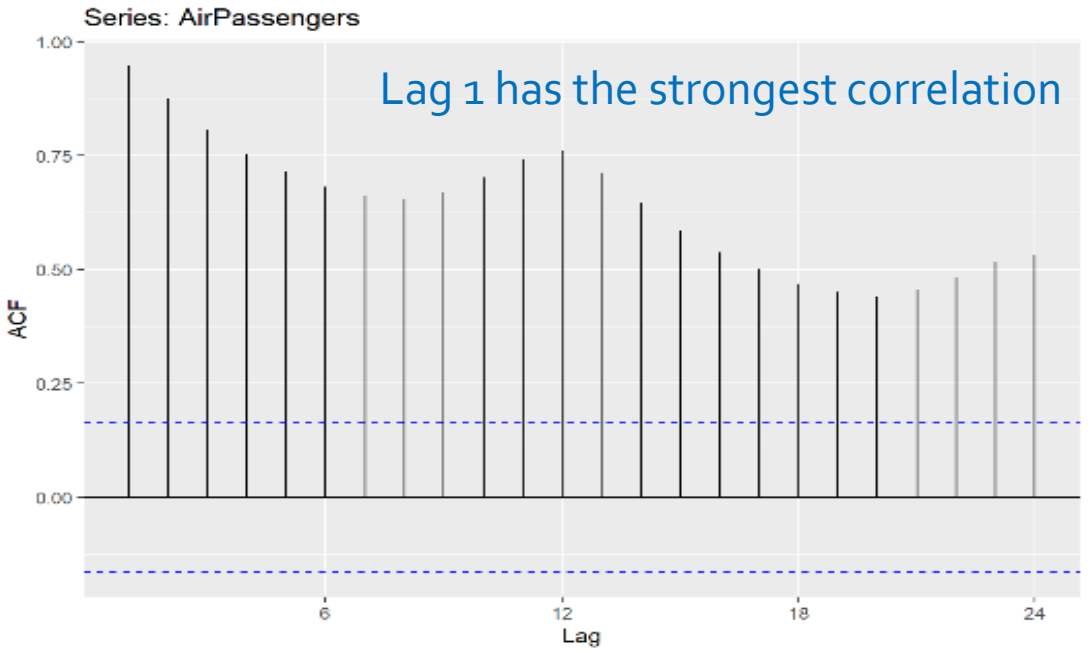
Examine the top few autocorrelations

```
acf(x = AirPassengers,plot=F)
```

```
##
## Autocorrelations of series 'AirPassengers', by lag
##
## 0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333
##  1.000  0.948  0.876  0.807  0.753  0.714  0.682  0.663  0.656  0.671  0.703
## 0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667 1.7500
##  0.743  0.760  0.713  0.646  0.586  0.538  0.500  0.469  0.450  0.442  0.457
```

Plot the top few autocorrelations but using ggAcf() instead of acf()

```
library(forecast)
ggAcf(x = AirPassengers)
```



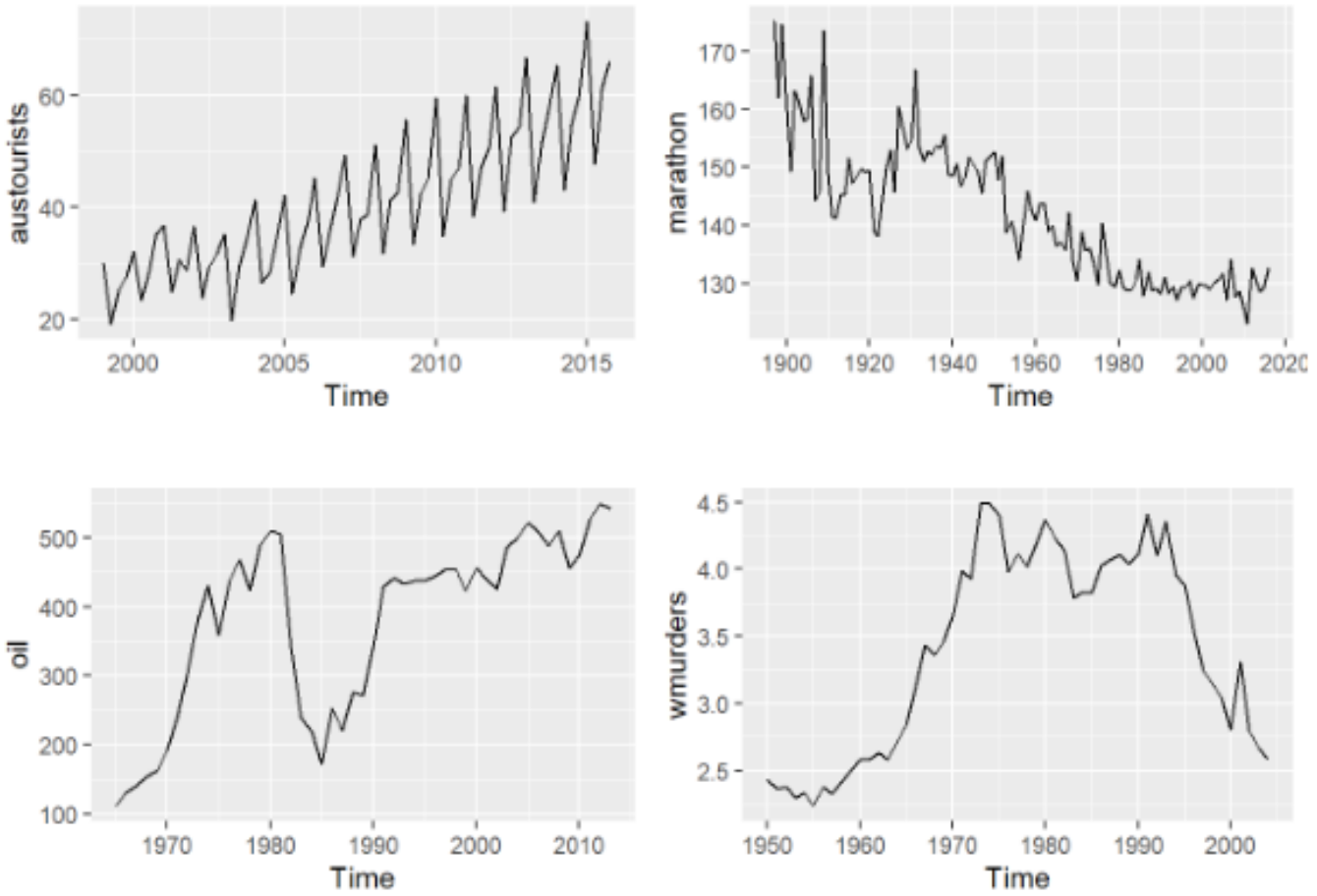
Time Series Patterns

- Trend: A long term increase or decrease in the data. Could be linear, non-linear, and may go from an increasing trend to a decreasing trend
- Seasonal: Pattern repeats itself every season. Here season may be period of the year or day of the week. Seasonality is always of a fixed or known frequency
- Cyclic: Data exhibit rises or falls that are not of a fixed frequency

On the other hand, a time-series data that does not exhibit any pattern is a Stationary Process. White Noise is the simplest Stationary Process

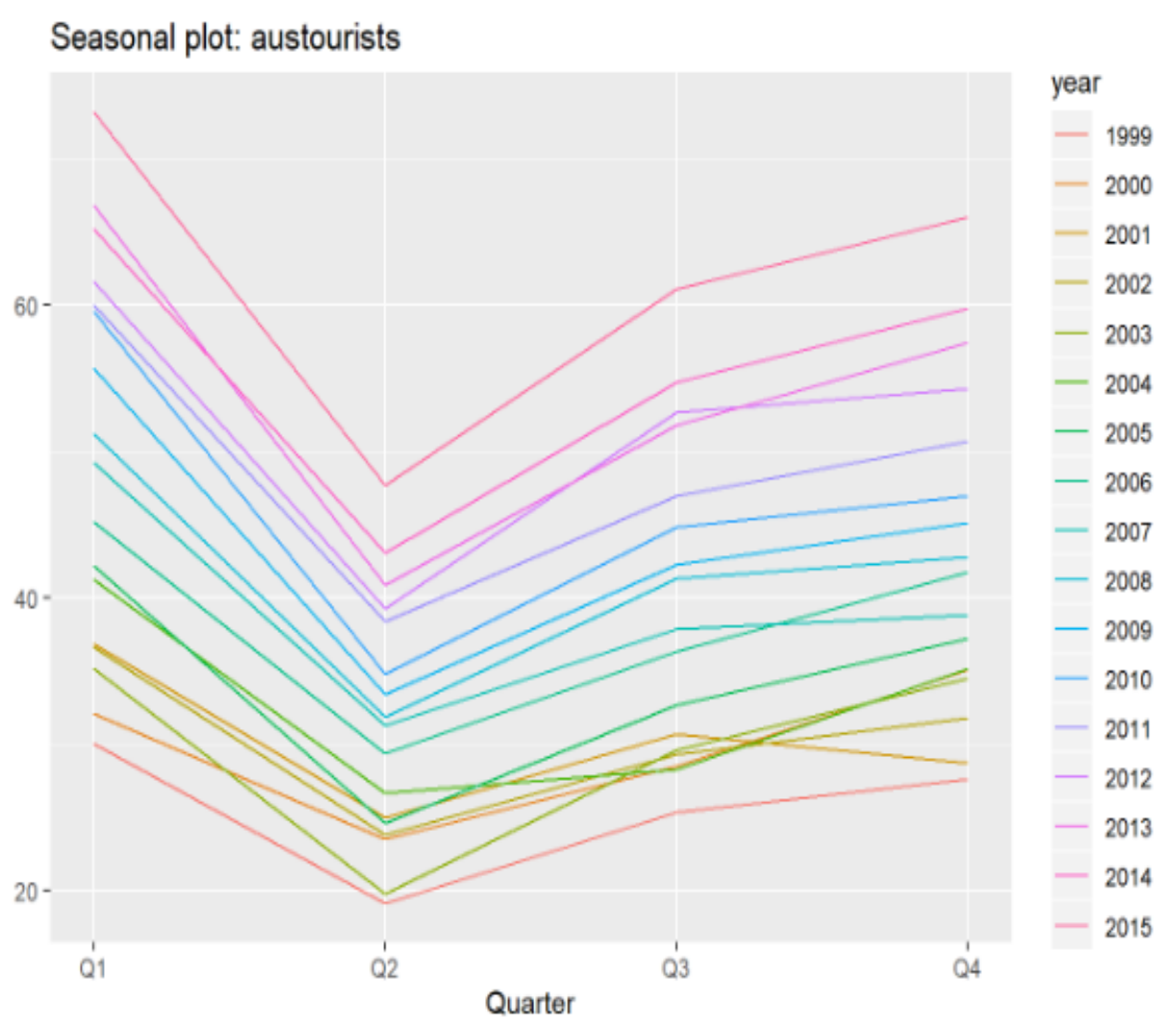
Which is the dominant time series pattern in these charts?

```
library(fpp)
library(gridExtra)
grid.arrange(autoplot(austourists), # International tourists to Australia
             autoplot(marathon),   # Marathon times
             autoplot(oil),        # Saudi Arabia oil production
             autoplot(wmurders))   # female murder rate
```

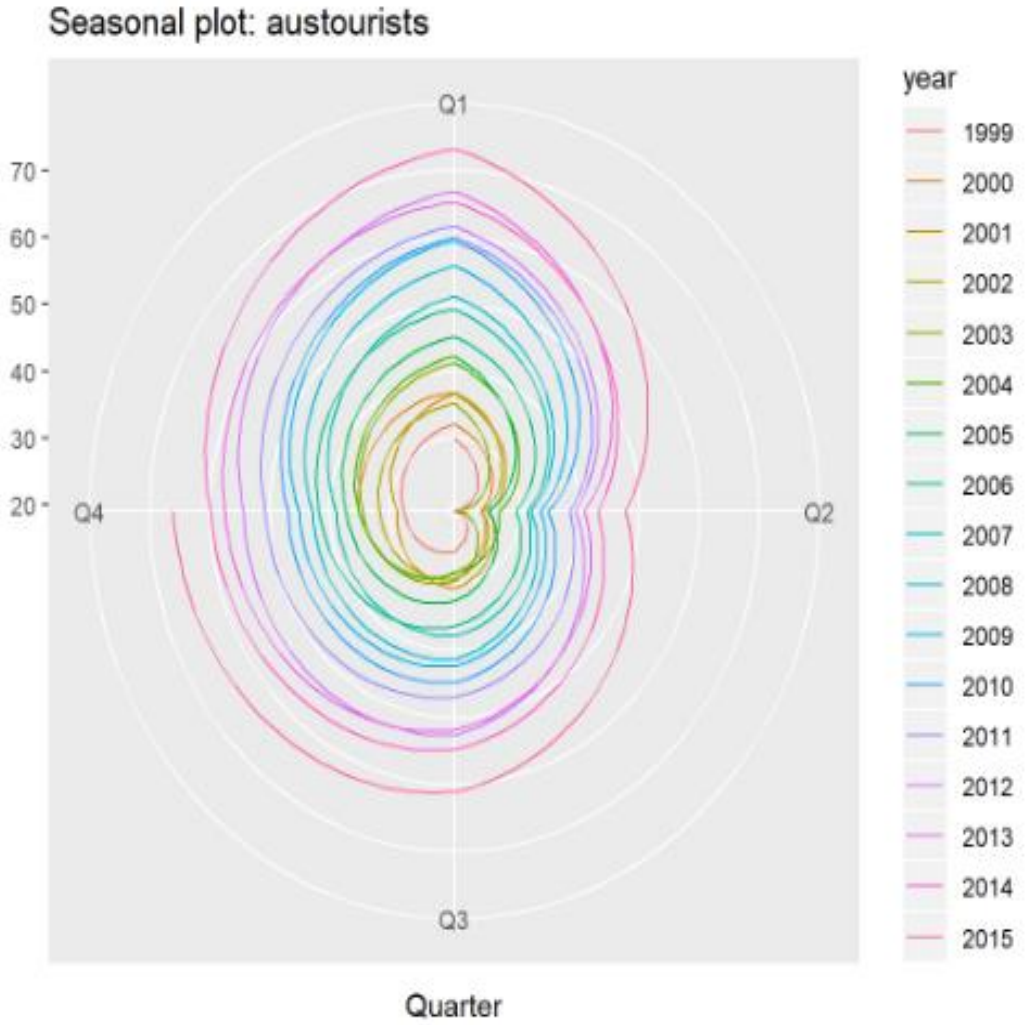


To more clearly spot seasonality, one can examine a seasonal plot.

```
ggseasonplot(austourists)
```




```
ggseasonplot(austourists,polar=T)
```



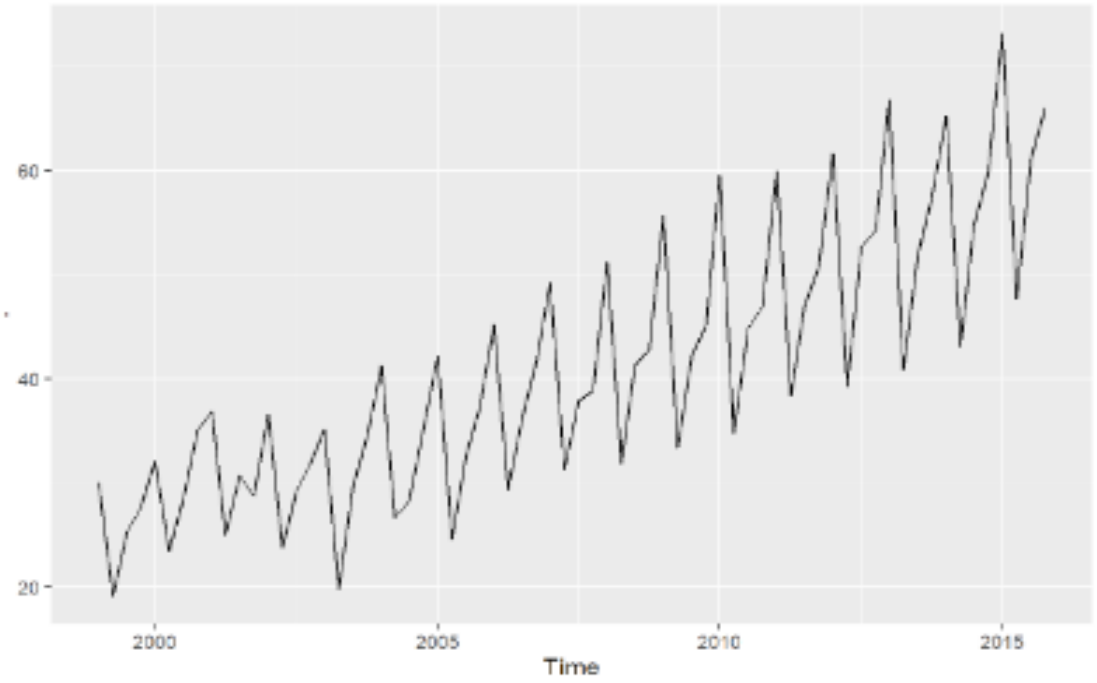
Time Series Components

Time-series may have multiple components in varying strengths. To better understand the series it is useful to divide it into its components:

* Trend components
* Seasonal component
* Remainder

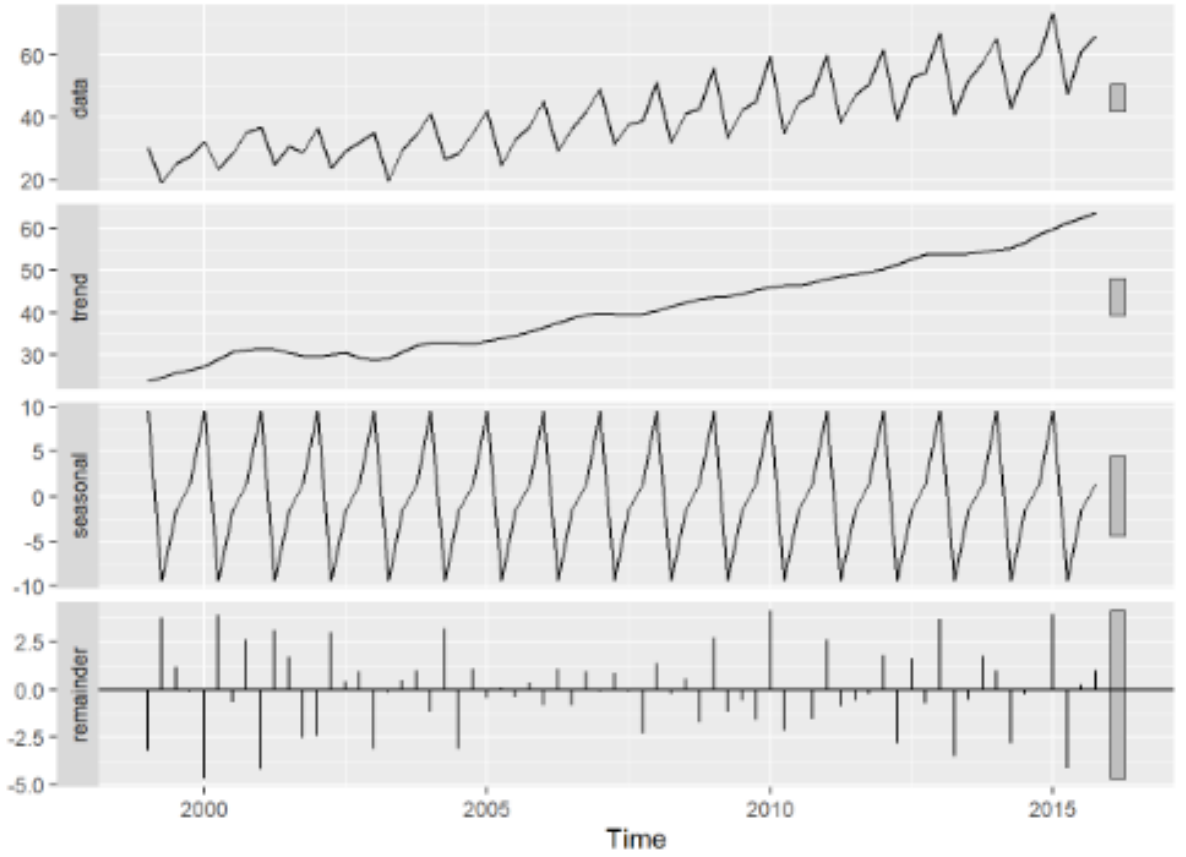
Here is a plot of quarterly visitor nights (in millions) spent by international tourists to Australia. 1999-2015

```
austourists%>%
  autoplot()
```



Now, let us decompose this time series using STL (Seasonality and Trend Decomposition using Loess), a versatile and robust method for decomposing a time series. In fact, decomposition can be used to measure the strength of trend and seasonality.

```
austourists%>%
  stl(s.window = 'periodic')%>%
  autoplot()
```



Simple Forecasting Methods

Let us forecast quarterly australian beer production.

```
library(fpp2)
ausbeer
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1956  284  213  227  308
## 1957  262  228  236  320
## 1958  272  233  237  313
## 1959  261  227  250  314
## 1960  286  227  260  311
## 1961  295  233  257  339
## 1962  279  250  270  346
## 1963  294  255  278  363
## 1964  313  273  300  370
## 1965  331  288  306  386
## 1966  335  288  308  402
## 1967  353  316  325  405
## 1968  393  319  327  442
## 1969  383  332  361  446
## 1970  387  357  374  466
## 1971  410  370  379  487
## 1972  419  378  393  506
## 1973  458  387  427  565
## 1974  465  445  450  556
## 1975  500  452  435  554
## 1991  464  424  436  574
## 1992  443  410  420  532
## 1993  433  421  410  512
## 1994  449  381  423  531
## 1995  426  408  416  520
## 1996  409  398  398  507
## 1997  432  398  406  526
## 1998  428  397  403  517
## 1999  435  383  424  521
## 2000  421  402  414  500
## 2001  451  380  416  492
## 2002  428  408  406  506
## 2003  435  380  421  490
## 2004  435  390  412  454
## 2005  416  403  408  482
## 2006  438  386  405  491
## 2007  427  383  394  473
## 2008  420  390  410  488
## 2009  415  398  419  488
## 2010  414  374
```

- Use the ausbeer data set

Split the data into train and test. Since the test data contains 42 quarters, we will be constructing forecasts for 42 periods.

```
train = window(ausbeer,end=c(1999,04))
test = window(ausbeer, start=c(2000,01))
length(test)
```

```
## [1] 42
```

Average Method

A very simple prediction, often the baseline in linear regression, is to use the average.

Model and Forecast

```
average_model = meanf(train,h = 42)
average_model
```

80% and 95% confidence bands

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2000 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2000 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2000 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2000 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2001 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2001 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2001 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2001 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2002 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2002 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2002 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2002 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2003 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2003 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2003 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2003 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
```

- forecast for beer production over the next 42 months of the test sample.
- the point forecast of Australian beer production for Q2 2003 is 412.4318


```
## 2004 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2004 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2004 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2004 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2005 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2005 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2005 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2005 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2006 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2006 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2006 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2006 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2007 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2007 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2007 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2007 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2008 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2008 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2008 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2008 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2009 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2009 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
## 2009 Q3       412.4318 291.6172 533.2464 227.0775 597.7861
## 2009 Q4       412.4318 291.6172 533.2464 227.0775 597.7861
## 2010 Q1       412.4318 291.6172 533.2464 227.0775 597.7861
## 2010 Q2       412.4318 291.6172 533.2464 227.0775 597.7861
```

The model object contains a point forecast, and 80% and 95% confidence bands for the estimate. Let us extract the estimate for the last observation, 2010, Q2

```
average_model$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 412.4318 412.4318 412.4318 412.4318
## 2001 412.4318 412.4318 412.4318 412.4318
## 2002 412.4318 412.4318 412.4318 412.4318
## 2003 412.4318 412.4318 412.4318 412.4318
## 2004 412.4318 412.4318 412.4318 412.4318
## 2005 412.4318 412.4318 412.4318 412.4318
## 2006 412.4318 412.4318 412.4318 412.4318
## 2007 412.4318 412.4318 412.4318 412.4318
## 2008 412.4318 412.4318 412.4318 412.4318
## 2009 412.4318 412.4318 412.4318 412.4318
## 2010 412.4318 412.4318
```

```
window(average_model$mean,c(2010,02))
```

```
##          Qtr2
## 2010 412.4318
```

Accuracy

Let us examine the accuracy of the above prediction from average_model on the train sample. accuracy() from library(forecast) is a handy function that automatically computes a set of error indices.

```
accuracy(average_model)
```

```
##                         ME     RMSE      MAE       MPE     MAPE     MASE
## Training set -5.118499e-15 93.38414 74.56612 -6.210103 20.74216 4.668865
##                    ACF1
## Training set 0.7089738
```

Let us examine the accuracy on the test sample. To get test-set performance, set the x argument as the entire dataset rather than just the test set.

```
accuracy(average_model,x = ausbeer)
```

```
##                         ME     RMSE      MAE       MPE      MAPE     MASE
## Training set -5.118499e-15 93.38414 74.56612 -6.210103 20.742160 4.668865
## Test set      1.525866e+01 40.10097 28.70346  2.876448  6.348359 1.797232
##                    ACF1 Theil's U
## Training set  0.70897377        NA
## Test set     -0.05298108 0.7832398
```

Now, let's set x argument as the test set and compare the results.

```
accuracy(average_model,x = test)
```

the RMSE of the prediction in the train sample

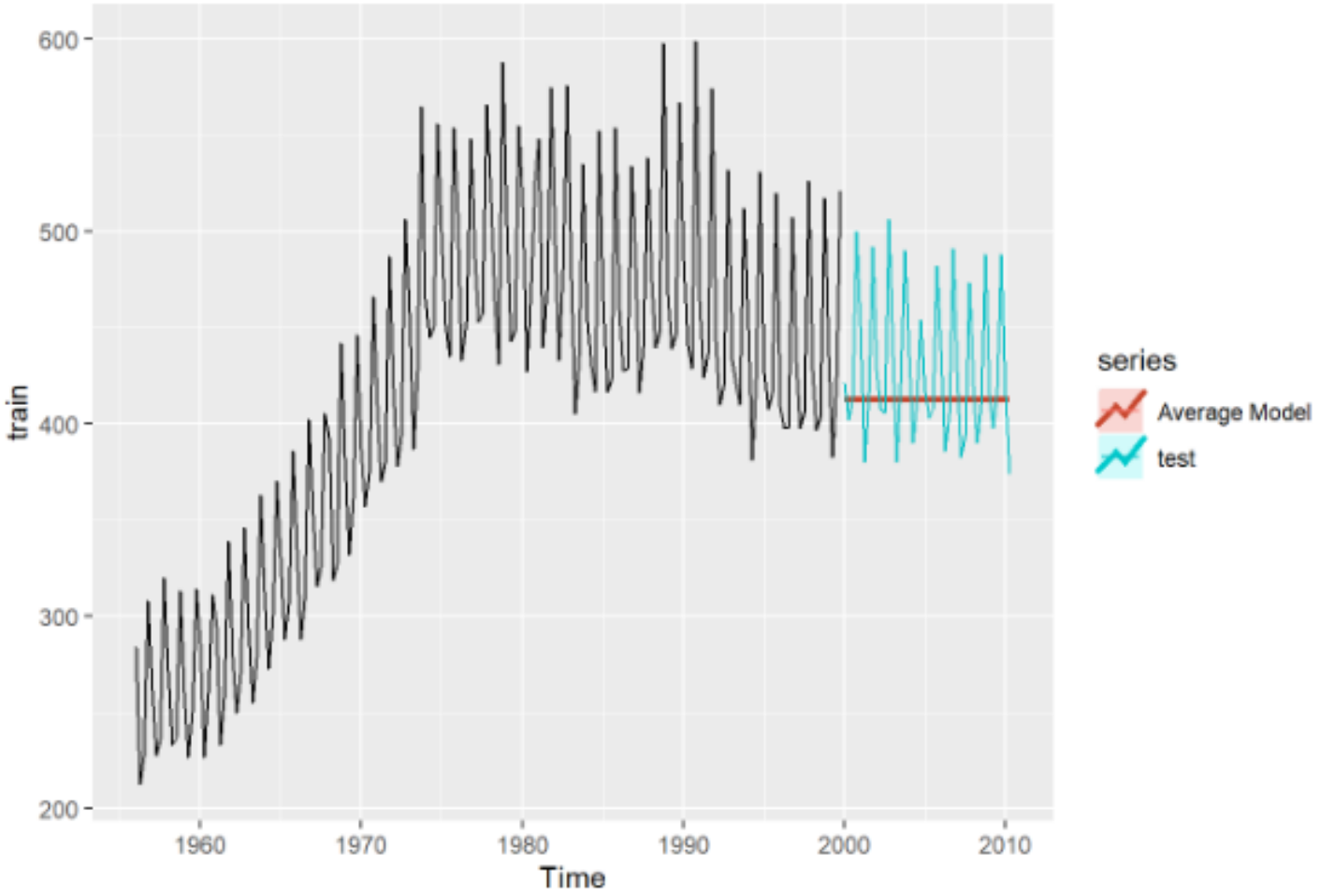
```
##                         ME     RMSE      MAE       MPE      MAPE     MASE
## Training set -5.118499e-15 93.38414 74.56612 -6.210103 20.742160 4.668865
## Test set      1.525866e+01 40.10097 28.70346  2.876448  6.348359 1.797232
##                    ACF1 Theil's U
## Training set  0.70897377        NA
## Test set     -0.05298108 0.7832398
```

the RMSE of the average_model on the test sample

Visualize Forecast

autoplot() uses a layered approach to plotting like ggplot2 and plot().

```
autoplot(train)+
  autolayer(average_model,PI = F,size=1.1,series = 'Average Model')+
  autolayer(test)
```



Naive Method

- Future will be the same as the last observation
- Since this is the best prediction for a random walk, these are also called random walk forecasts

Model and Forecast

```
naive_model = naive(train,h=42)
```

Let's see the point forecast for 2010 Q2.

```
naive_model$mean
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 2000  521  521  521  521
## 2001  521  521  521  521
## 2002  521  521  521  521
## 2003  521  521  521  521
## 2004  521  521  521  521
## 2005  521  521  521  521
## 2006  521  521  521  521
## 2007  521  521  521  521
## 2008  521  521  521  521
## 2009  521  521  521  521
## 2010  521  521
```

- the point forecast of beer production for Q2, 2010 is 521

Note, for a naive model, the forecast is the same as the last observation in the train sample

```
last(train)
```

```
## [1] 521
```


Use naive_model to construct a forecast for beer production over the next 42 months of the test sample.

Accuracy

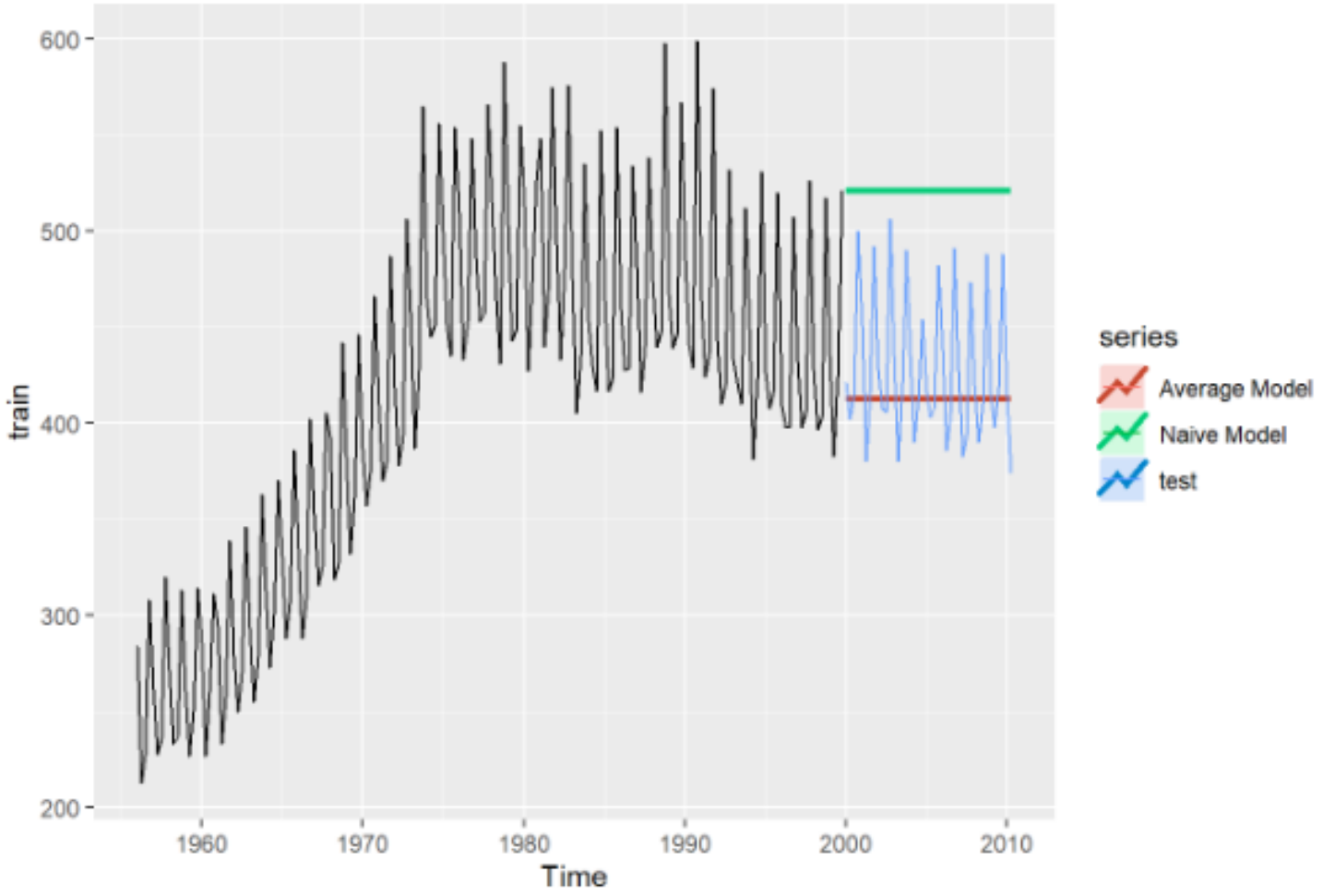
```
accuracy(naive_model,x=ausbeer)
```

```
##                     ME      RMSE      MAE        MPE     MAPE     MASE
## Training set   1.354286  70.30818 58.73714  -0.967862 13.94342 3.677753
## Test set     -93.309524 100.40881 93.30952 -22.690269 22.69027 5.842460
##                    ACF1 Theil's U
## Training set -0.22478344        NA
## Test set     -0.05298108  1.898409
```

RMSE of the naive_model on the test sample

Visualize Forecast


```
autoplot(train)+
  autolayer(average_model,PI = F,size=1.1,series = 'Average Model')+
  autolayer(naive_model,PI=F,size=1.1, series='Naive Model')+
  autolayer(test)
```



Seasonal Naive Method

Forecast is equal to the last observed value from the same season

Model and Forecast

```
seasonal_naive_model = snaive(train,h=42)
seasonal_naive_model
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2000 Q1             435 409.3535 460.6465 395.7771 474.2229
## 2000 Q2             383 357.3535 408.6465 343.7771 422.2229
## 2000 Q3             424 398.3535 449.6465 384.7771 463.2229
## 2000 Q4             521 495.3535 546.6465 481.7771 560.2229
## 2001 Q1             435 398.7304 471.2696 379.5304 490.4696
## 2001 Q2             383 346.7304 419.2696 327.5304 438.4696
## 2001 Q3             424 387.7304 460.2696 368.5304 479.4696
## 2001 Q4             521 484.7304 557.2696 465.5304 576.4696
## 2002 Q1             435 390.5790 479.4210 367.0639 502.9361
## 2002 Q2             383 338.5790 427.4210 315.0639 450.9361
## 2002 Q3             424 379.5790 468.4210 356.0639 491.9361
## 2002 Q4             521 476.5790 565.4210 453.0639 588.9361
## 2003 Q1             435 383.7070 486.2930 356.5542 513.4458
## 2003 Q2             383 331.7070 434.2930 304.5542 461.4458
## 2003 Q3             424 372.7070 475.2930 345.5542 502.4458
## 2003 Q4             521 469.7070 572.2930 442.5542 599.4458
## 2004 Q1             435 377.6527 492.3473 347.2949 522.7051
## 2004 Q2             383 325.6527 440.3473 295.2949 470.7051
## 2004 Q3             424 366.6527 481.3473 336.2949 511.7051
## 2004 Q4             521 463.6527 578.3473 433.2949 608.7051
## 2005 Q1             435 372.1792 497.8208 338.9239 531.0761
## 2005 Q2             383 320.1792 445.8208 286.9239 479.0761
## 2005 Q3             424 361.1792 486.8208 327.9239 520.0761
## 2005 Q4             521 458.1792 583.8208 424.9239 617.0761
## 2006 Q1             435 367.1458 502.8542 331.2259 538.7741
## 2006 Q2             383 315.1458 450.8542 279.2259 486.7741
## 2006 Q3             424 356.1458 491.8542 320.2259 527.7741
## 2006 Q4             521 453.1458 588.8542 417.2259 624.7741
## 2007 Q1             435 362.4608 507.5392 324.0608 545.9392
## 2007 Q2             383 310.4608 455.5392 272.0608 493.9392
## 2007 Q3             424 351.4608 496.5392 313.0608 534.9392
## 2007 Q4             521 448.4608 593.5392 410.0608 631.9392
## 2008 Q1             435 358.0605 511.9395 317.3312 552.6688
## 2008 Q2             383 306.0605 459.9395 265.3312 500.6688
## 2008 Q3             424 347.0605 500.9395 306.3312 541.6688
## 2008 Q4             521 444.0605 597.9395 403.3312 638.6688
## 2009 Q1             435 353.8987 516.1013 310.9662 559.0338
## 2009 Q2             383 301.8987 464.1013 258.9662 507.0338
## 2009 Q3             424 342.8987 505.1013 299.9662 548.0338
## 2009 Q4             521 439.8987 602.1013 396.9662 645.0338
## 2010 Q1             435 349.9402 520.0598 304.9123 565.0877
## 2010 Q2             383 297.9402 468.0598 252.9123 513.0877
```

Point forecast for 2010 Q2. Note the similarity in predictions for any given season.

```
seasonal_naive_model$mean
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 2000  435  383  424  521
## 2001  435  383  424  521
## 2002  435  383  424  521
## 2003  435  383  424  521
## 2004  435  383  424  521
## 2005  435  383  424  521
## 2006  435  383  424  521
## 2007  435  383  424  521
## 2008  435  383  424  521
## 2009  435  383  424  521
## 2010  435  383
```

COLUMBIA UNIVERSITY
School of Professional Studies

- Use the ausbeer data set
  - Recall:
    train = window(ausbeer, end=c(1999,04))
    test = window(ausbeer, start=c(2000,01))

```
library(fpp2)
ausbeer
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1991  464  424  436  574
## 1992  443  410  420  532
## 1993  433  421  410  512
## 1994  449  381  423  531
## 1995  426  408  416  520
## 1996  409  398  398  507
## 1997  432  398  406  526
## 1998  428  397  403  517
## 1999  435  383  424  521
## 2000  421  402  414  500
## 2001  451  380  416  492
## 2002  428  408  406  506
## 2003  435  380  421  490
## 2004  435  390  412  454
## 2005  416  403  408  482
## 2006  438  386  405  491
## 2007  427  383  394  473
## 2008  420  390  410  488
## 2009  415  398  419  488
## 2010  414  374
```

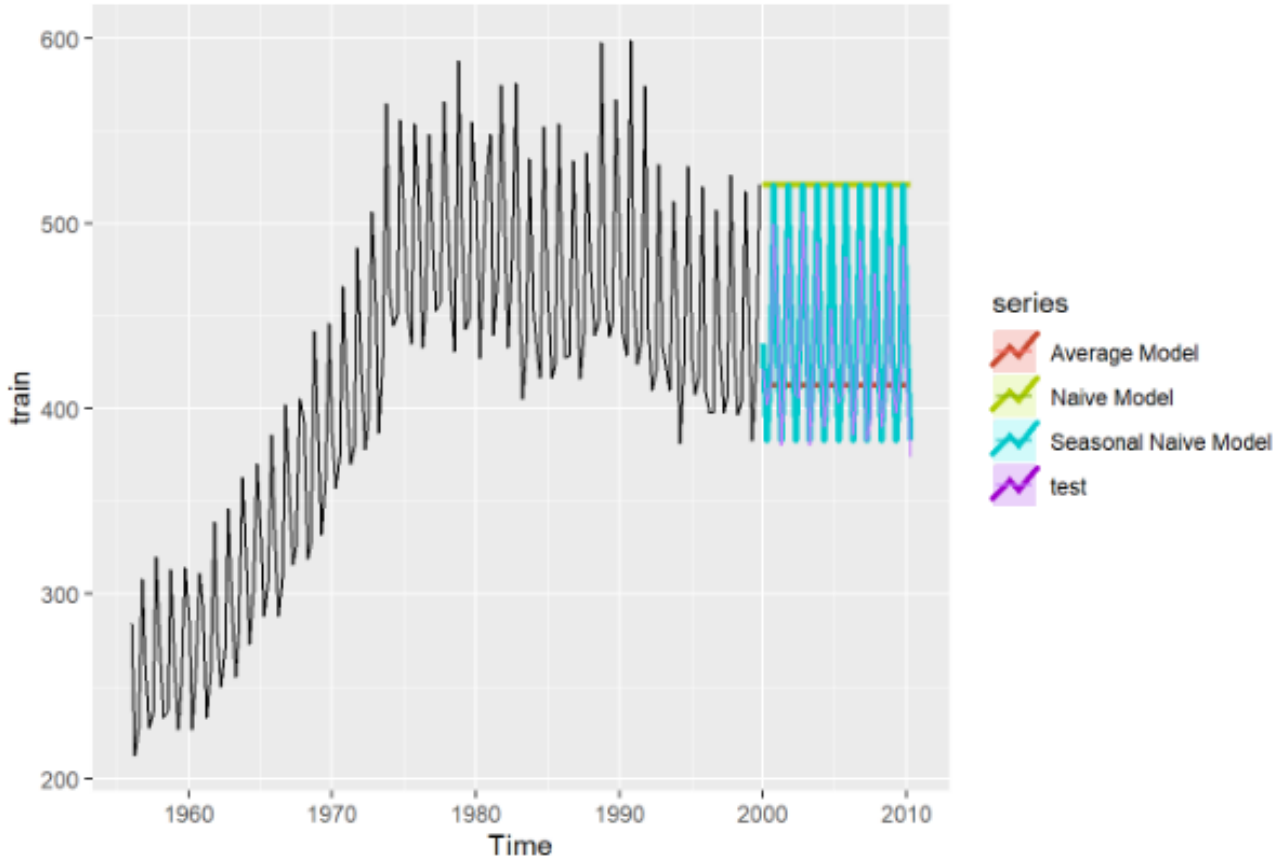
Accuracy

```
accuracy(seasonal_naive_model,x = ausbeer)
```

```
##                     ME     RMSE      MAE       MPE     MAPE     MASE       ACF1
## Training set   4.25000 20.01206 15.97093  1.148642 3.818071 1.000000 0.02734555
## Test set     -11.54762 21.85286 17.02381 -2.521773 3.863916 1.065925 0.02909800
##              Theil's U
## Training set        NA
## Test set     0.4278746
```

Visualize Forecast

```
autoplot(train)+
  autolayer(average_model,PI = F,size=1.1,series = 'Average Model')+
  autolayer(naive_model,PI=F,size=1.1, series='Naive Model')+
  autolayer(seasonal_naive_model,PI=F,size=1.1,series='Seasonal Naive Model')+
  autolayer(test)
```



Drift Method

Allow forecasts to increase or decrease over time, where the amount of change over time (called the drift) is set to be average change seen in historical data

Model and Forecast

```
drift_model = rwf(train,h=42,drift = T)
drift_model$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 522.3543 523.7086 525.0629 526.4171
## 2001 527.7714 529.1257 530.4800 531.8343
## 2002 533.1886 534.5429 535.8971 537.2514
## 2003 538.6057 539.9600 541.3143 542.6686
## 2004 544.0229 545.3771 546.7314 548.0857
## 2005 549.4400 550.7943 552.1486 553.5029
## 2006 554.8571 556.2114 557.5657 558.9200
## 2007 560.2743 561.6286 562.9829 564.3371
## 2008 565.6914 567.0457 568.4000 569.7543
## 2009 571.1086 572.4629 573.8171 575.1714
## 2010 576.5257 577.8800
```

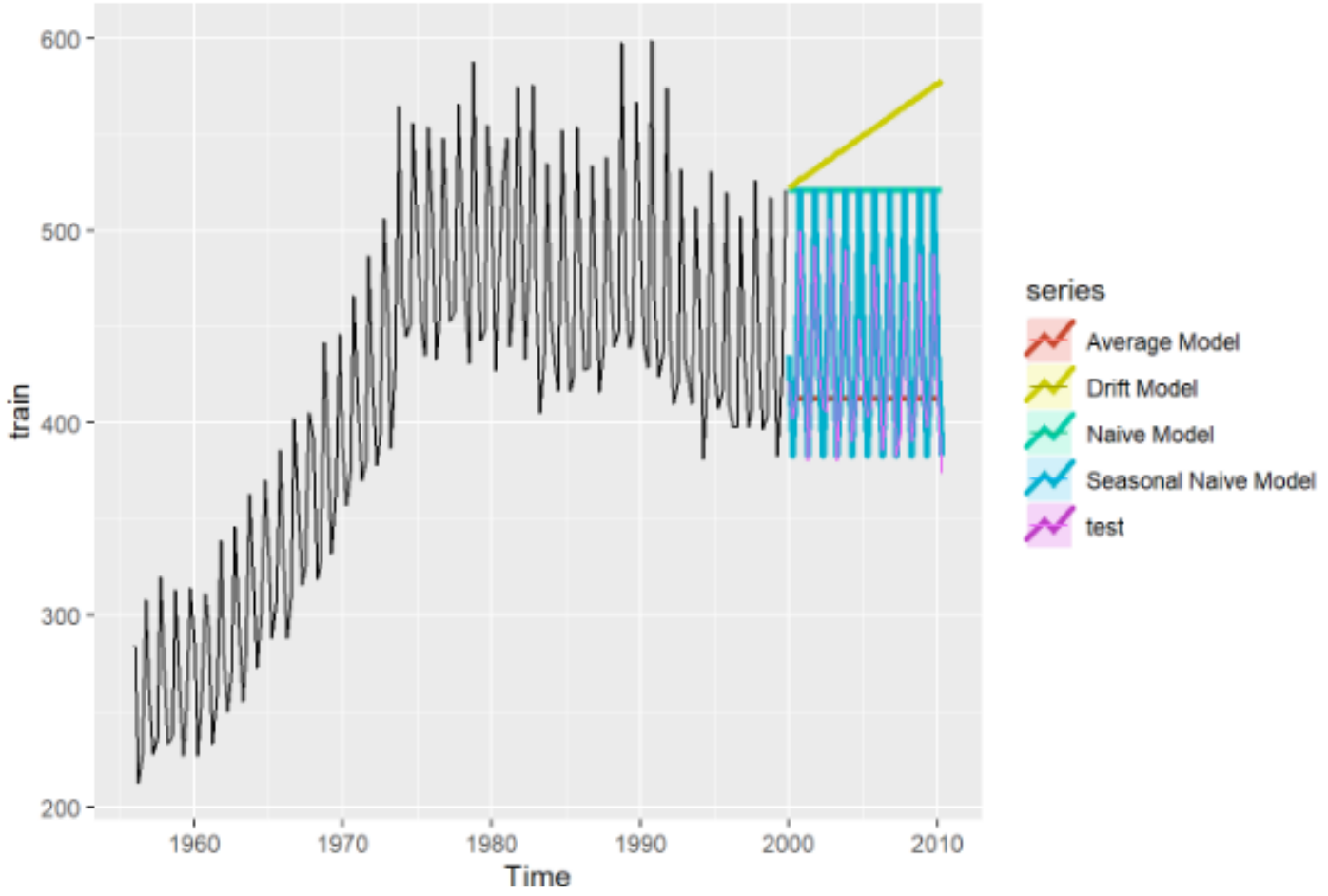
Accuracy

```
accuracy(drift_model,x = ausbeer)
```

```
##                        ME      RMSE       MAE         MPE     MAPE     MASE
## Training set -3.572936e-15  70.29513  58.81084  -1.315888 13.97802 3.682368
## Test set     -1.224267e+02 129.44954 122.42667 -29.581531 29.58153 7.665594
##                    ACF1 Theil's U
## Training set -0.2247834        NA
## Test set      0.1520385  2.458927
```

Visualize Forecast

```
autoplot(train)+
  autolayer(average_model,PI = F,size=1.1,series = 'Average Model')+
  autolayer(naive_model,PI=F,size=1.1, series='Naive Model')+
  autolayer(seasonal_naive_model,PI=F,size=1.1,series='Seasonal Naive Model')+
  autolayer(drift_model,PI=F,size=1.1,series='Drift Model')+
  autolayer(test)
```



Exponential Smoothing Models

Forecasts are weighted averages of past observations with the weights decaying exponentially such that recent observations get weighted more than distant observations.

Simple exponential smoothing

- Forecasts are calculated using weighted averages, where the weights decrease exponentially. Most recent observations get the heaviest weight.
- Simplest of exponential smoothing methods
- Suitable for forecasting data with no clear trend or seasonal pattern.

Model and Forecast

```
ses_model = ses(train,h = 42)
ses_model$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 445.7878 445.7878 445.7878 445.7878
## 2001 445.7878 445.7878 445.7878 445.7878
## 2002 445.7878 445.7878 445.7878 445.7878
## 2003 445.7878 445.7878 445.7878 445.7878
## 2004 445.7878 445.7878 445.7878 445.7878
## 2005 445.7878 445.7878 445.7878 445.7878
## 2006 445.7878 445.7878 445.7878 445.7878
## 2007 445.7878 445.7878 445.7878 445.7878
## 2008 445.7878 445.7878 445.7878 445.7878
## 2009 445.7878 445.7878 445.7878 445.7878
## 2010 445.7878 445.7878
```

- Exponential smoothing method has the same value for all forecast
- It is the estimated mean of the future possible sample path.

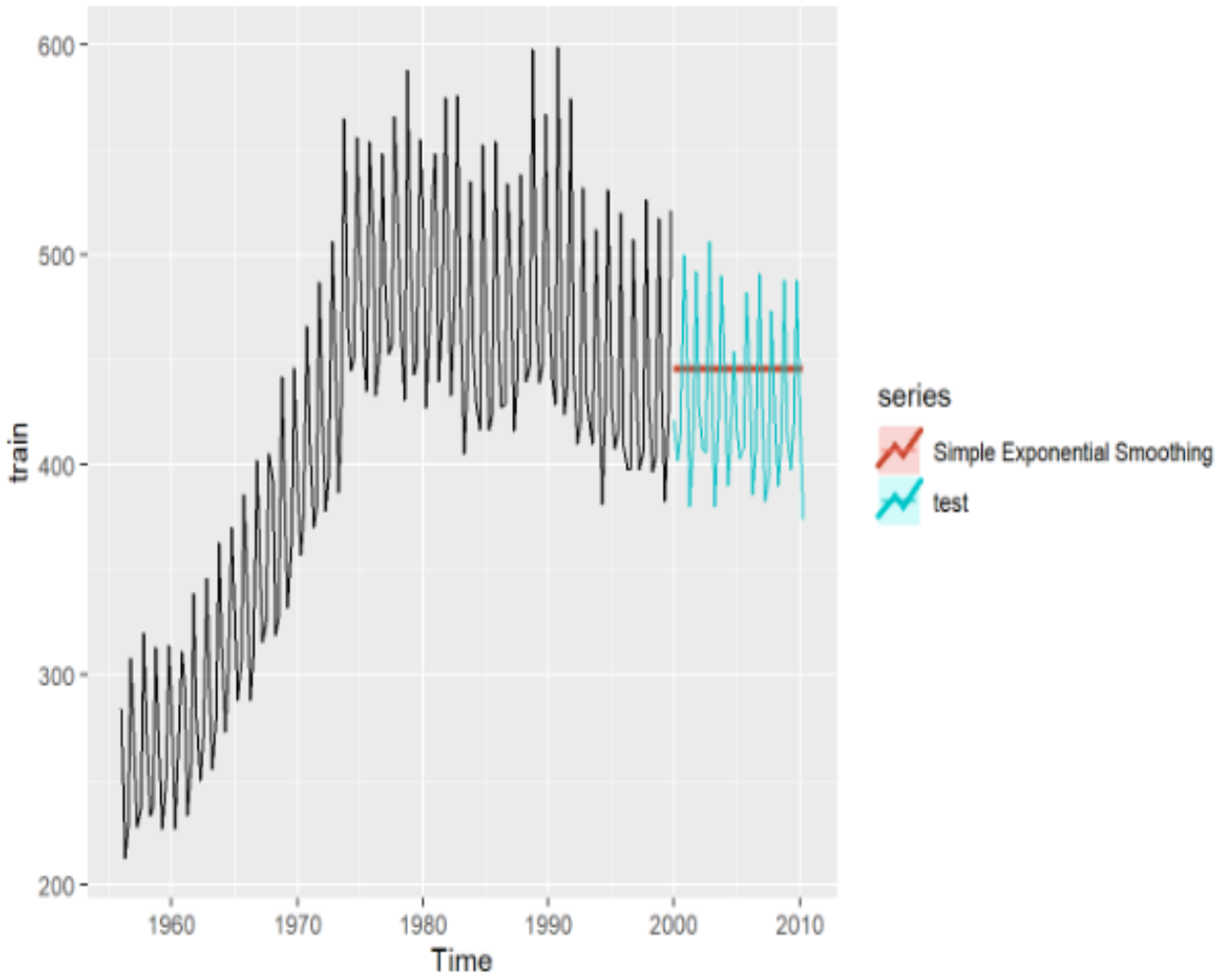
Accuracy

```
accuracy(ses_model,x = ausbeer)
```

```
##                     ME     RMSE      MAE        MPE     MAPE     MASE
## Training set   6.78376 52.86562 42.65547  0.5391999 10.07522 2.670819
## Test set     -18.09737 41.26471 37.68469 -4.9785614  8.97363 2.359580
##                    ACF1 Theil's U
## Training set -0.05109356        NA
## Test set     -0.05298108 0.7972014
```

Visualize Forecasts

```
autoplot(train)+
  autolayer(ses_model,series = "Simple Exponential Smoothing",PI = F, size=1.1)+
  autolayer(test)
```



- Exponential smoothing method with trend

Holt's Method

Extends simple exponential smoothing to allow the forecasting of data with a trend

Model and Forecast

```
holt_model = holt(train,h=42)
holt_model$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 443.8257 448.1301 452.4345 456.7389
## 2001 461.0433 465.3477 469.6522 473.9566
## 2002 478.2610 482.5654 486.8698 491.1742
## 2003 495.4786 499.7830 504.0874 508.3918
## 2004 512.6962 517.0006 521.3051 525.6095
## 2005 529.9139 534.2183 538.5227 542.8271
## 2006 547.1315 551.4359 555.7403 560.0447
## 2007 564.3491 568.6535 572.9579 577.2624
## 2008 581.5668 585.8712 590.1756 594.4800
## 2009 598.7844 603.0888 607.3932 611.6976
## 2010 616.0020 620.3064
```

Accuracy

```
accuracy(holt_model,x=ausbeer)
```

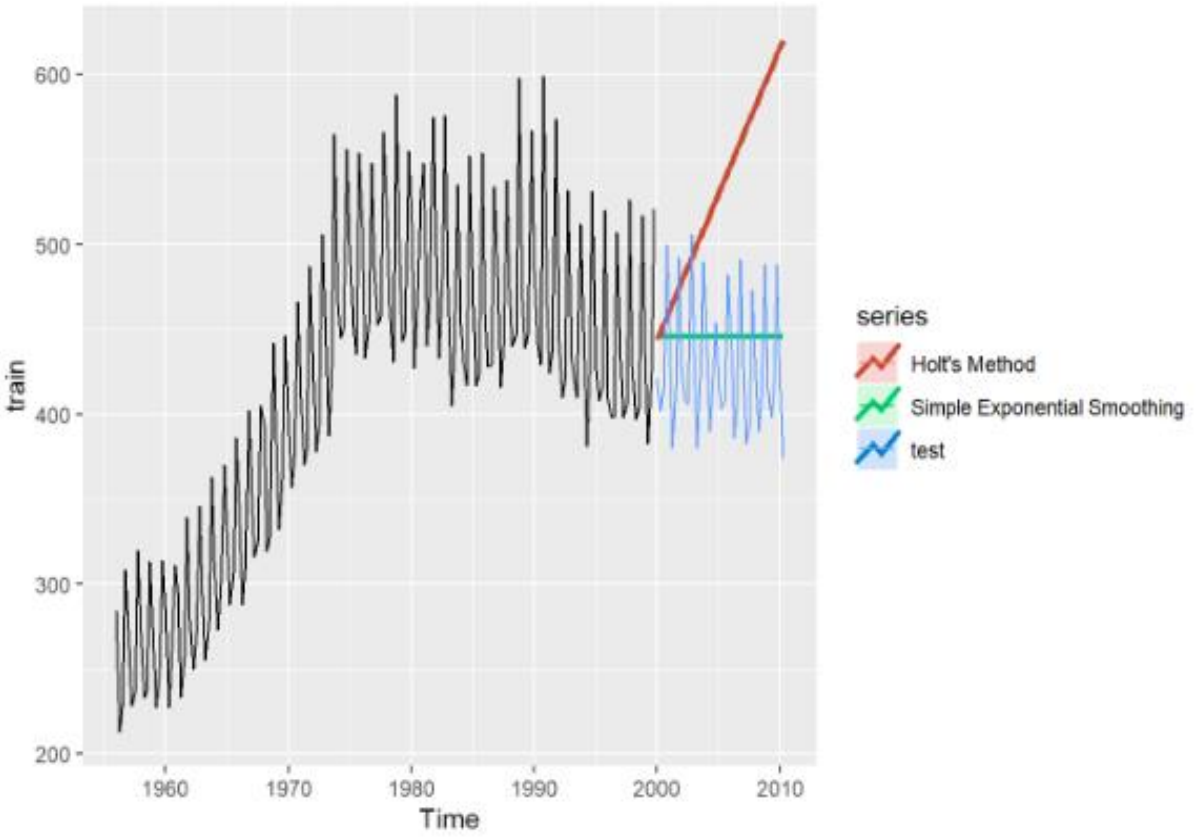
```
##                      ME      RMSE       MAE         MPE      MAPE     MASE
## Training set    1.048297  51.06394  41.34716  -0.8878183  9.914797 2.588901
## Test set     -104.375595 124.04137 108.00085 -25.4057657 26.131936 6.762339
##                    ACF1 Theil's U
## Training set -0.03671365        NA
## Test set      0.61128212  2.375372
```

- Holt's method will produce forecasts where the trend continues at the same slope indefinitely into the future.

COLUMBIA UNIVERSITY
School of Professional Studies

Visualize Forecast

```
autoplot(train)+
  autolayer(ses_model,series = "Simple Exponential Smoothing",PI = F, size=1.1)+
  autolayer(holt_model,series="Holt's Method",PI=F,size=1.1)+
  autolayer(test)
```



Holt's Method with Damping

Forecasts generally display a constant trend (increasing or decreasing) indefinitely into the future. For this reason, a damping parameter is usually included

Model and Forecast

```
holt_damped_model = holt(train,h=42,damped = T)
holt_damped_model$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 441.6555 443.9208 445.9752 447.8384
## 2001 449.5282 451.0607 452.4506 453.7112
## 2002 454.8544 455.8912 456.8315 457.6844
## 2003 458.4578 459.1593 459.7954 460.3724
## 2004 460.8957 461.3702 461.8006 462.1910
## 2005 462.5450 462.8660 463.1572 463.4213
## 2006 463.6608 463.8780 464.0750 464.2537
## 2007 464.4157 464.5627 464.6960 464.8168
## 2008 464.9265 465.0259 465.1161 465.1978
## 2009 465.2720 465.3393 465.4003 465.4556
## 2010 465.5058 465.5513
```


Accuracy

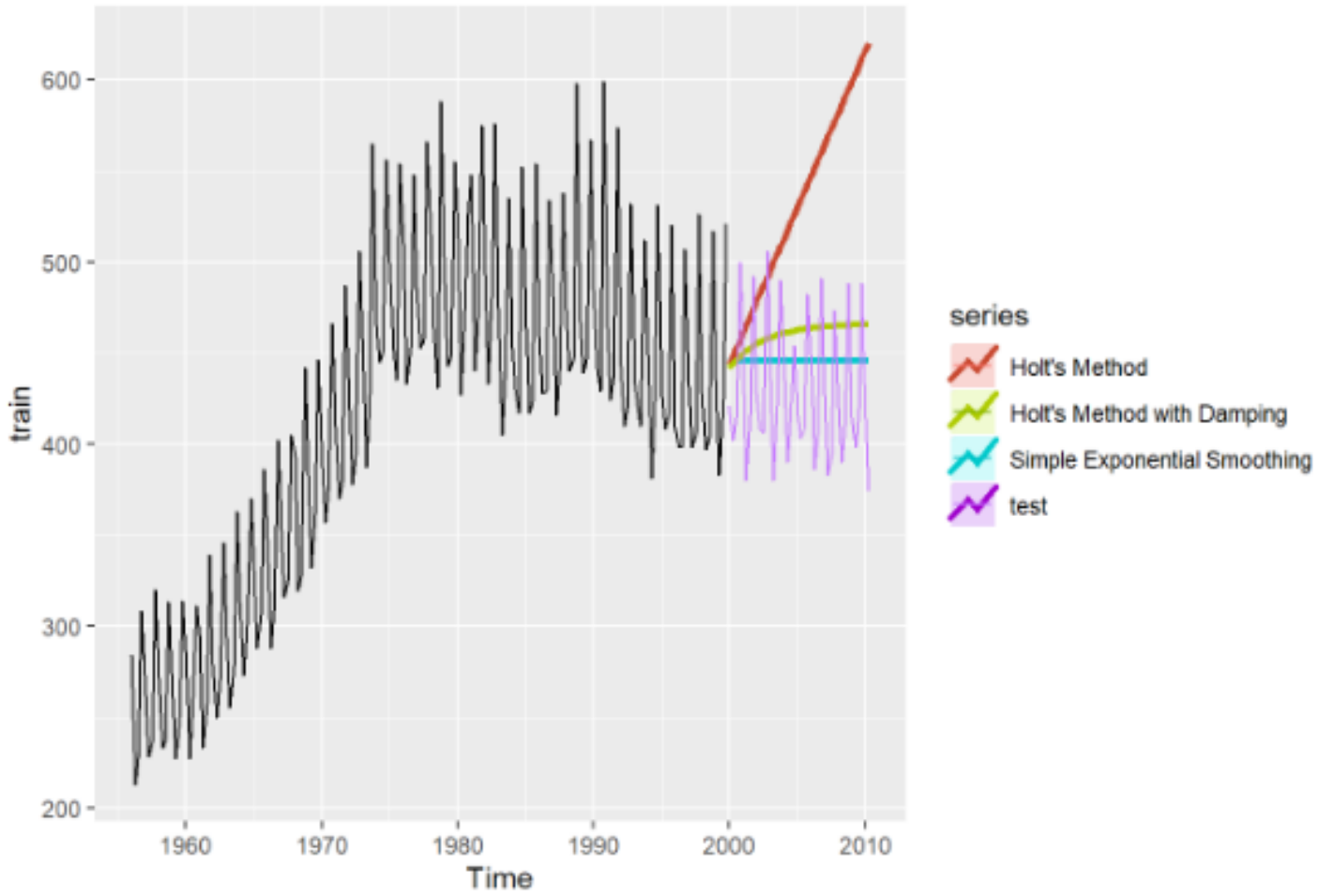
```
accuracy(holt_damped_model,x=ausbeer)
```

```
##                      ME     RMSE      MAE        MPE      MAPE      MASE
## Training set   3.113352 50.01115 40.99473 -0.3297766  9.791311 2.566834
## Test set     -32.180549 49.84832 44.97675 -8.3039388 10.897279 2.816164
##                     ACF1 Theil's U
## Training set -0.059527825        NA
## Test set     -0.005103579 0.9593471
```

- Dampen the trend over time so as to level off to a constant value.
- Under this method, the short run forecast is trended but the long run forecast is constant

Visualize Forecast

```
autoplot(train)+
  autolayer(ses_model,series = "Simple Exponential Smoothing",PI = F, size=1.1)+
  autolayer(holt_model,series="Holt's Method",PI=F,size=1.1)+
  autolayer(holt_damped_model,series="Holt's Method with Damping",PI=F,size=1.1)+
  autolayer(test)
```



Holt-Winter's seasonal method

Extends Holt's method to capture seasonality. Two types:

- Additive method is used when seasonal variations are roughly constant
- Multiplicative method is used when seasonal variations change in proportion to the level of the series

Holt_Winter's Additive

Model and Forecast

```
hw_additive = hw(train,h=42,seasonal = 'additive', damped=T)
hw_additive$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 431.7122 396.8446 412.7531 524.0245
## 2001 433.3171 398.1286 413.7803 524.8462
## 2002 433.9745 398.6545 414.2010 525.1828
## 2003 434.2438 398.8699 414.3733 525.3207
## 2004 434.3541 398.9581 414.4439 525.3771
## 2005 434.3992 398.9943 414.4728 525.4003
## 2006 434.4177 399.0091 414.4847 525.4097
## 2007 434.4253 399.0151 414.4895 525.4136
## 2008 434.4284 399.0176 414.4915 525.4152
## 2009 434.4297 399.0186 414.4923 525.4159
## 2010 434.4302 399.0190
```

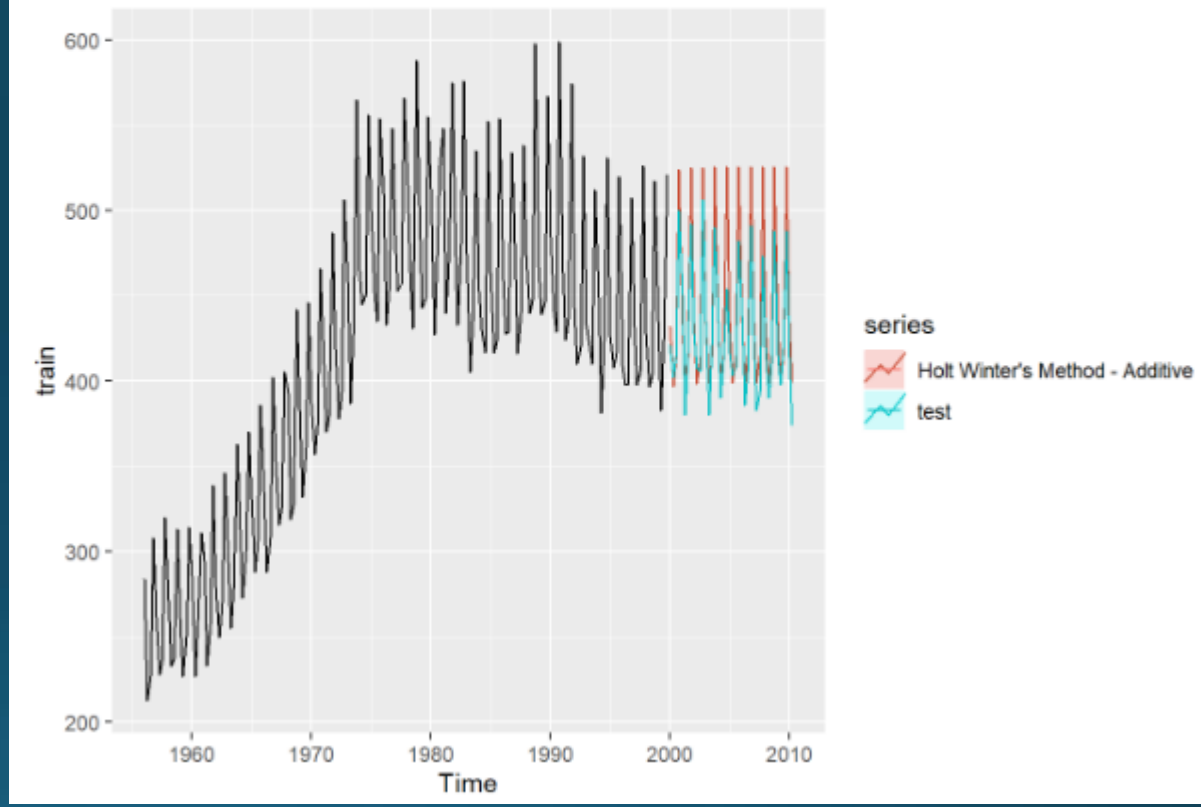
Accuracy

```
accuracy(hw_additive,x = ausbeer)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set   1.339248 16.02895 12.41005  0.2859756 3.005731 0.7770402
## Test set     -14.058340 22.67754 16.71429 -3.1512303 3.775641 1.0465444
##                    ACF1 Theil's U
## Training set -0.1816207        NA
## Test set     -0.1857105 0.4430322
```

Visualize Forecasts

```
autoplot(train)+
  autolayer(hw_additive,series="Holt Winter's Method - Additive",PI=F)+
  autolayer(test)
```



- Exponential smoothing method with trend and seasonality (Additive version)
- When seasonal variation increases with the level of the series, multiplicative method should be use instead of additive method.

- Exponential smoothing method with trend and seasonality (Multiplicative version)

Holt_Winter's Multiplicative

Model and Forecast

```
hw_multiplicative = hw(train,h=42,seasonal = 'multiplicative', damped=T)
hw_multiplicative$mean
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000 432.4894 398.2671 413.6034 524.9590
## 2001 435.1293 400.2185 415.2329 526.6252
## 2002 436.2614 401.0644 415.9487 527.3690
## 2003 436.7758 401.4576 416.2903 527.7351
## 2004 437.0374 401.6654 416.4787 527.9464
## 2005 437.1954 401.7973 416.6044 528.0945
## 2006 437.3110 401.8981 416.7043 528.2166
## 2007 437.4093 401.9861 416.7938 528.3281
## 2008 437.5005 402.0690 416.8789 528.4353
## 2009 437.5887 402.1497 416.9623 528.5407
## 2010 437.6758 402.2296
```

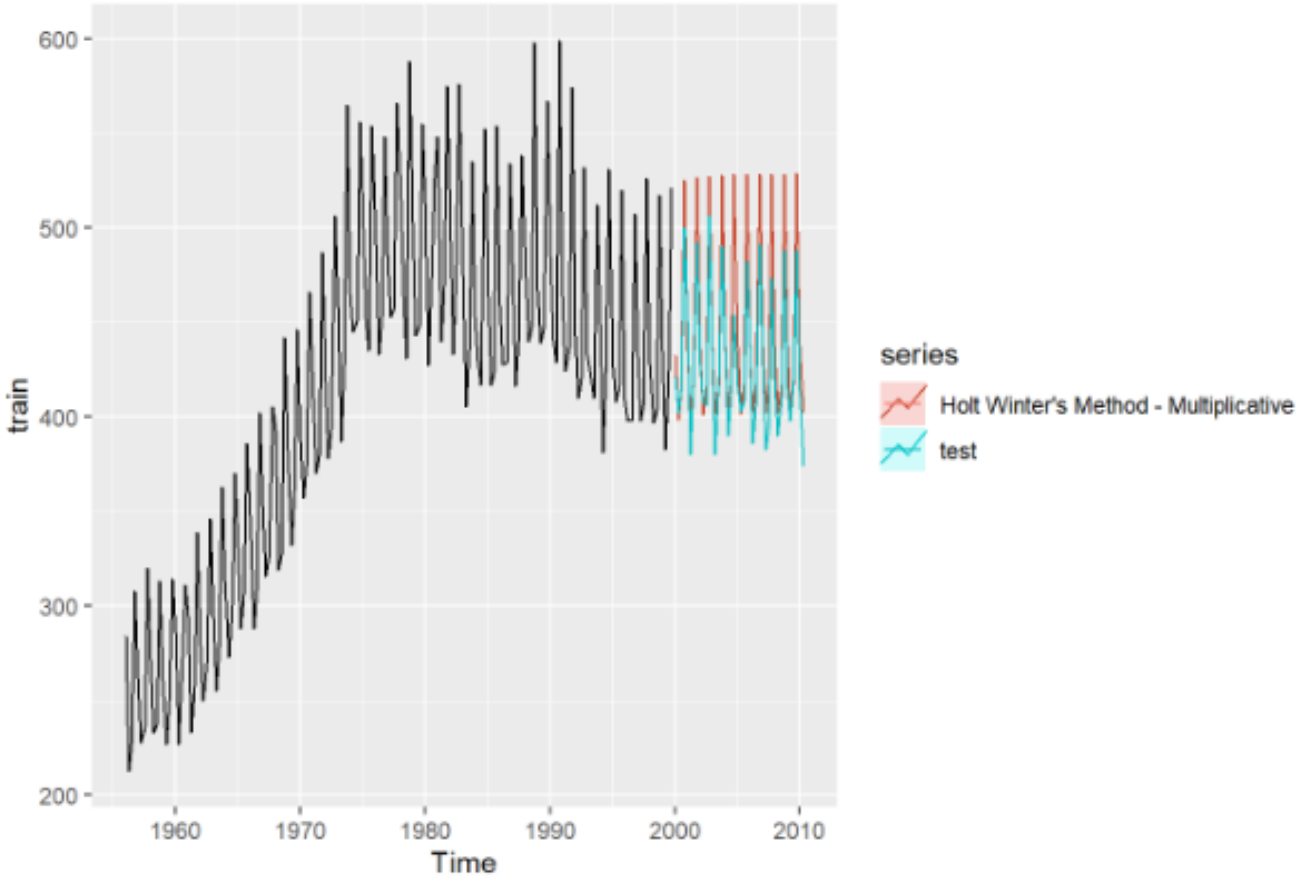
Accuracy

```
accuracy(hw_multiplicative,x=ausbeer)
```

```
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   1.259767 15.55545 11.94622  0.314631 2.887841 0.7479979
## Test set     -16.474394 24.38794 18.20496 -3.721500 4.125710 1.1398809
##                    ACF1 Theil's U
## Training set -0.2232171        NA
## Test set     -0.1694875  0.475703
```

Visualize Forecast

```
autoplot(train)+
  autolayer(hw_multiplicative,series="Holt Winter's Method - Multiplicative",PI=F)+
  autolayer(test)
```



ETS Models

But, exponential smoothing methods are not limited to the ones examined so far. By considering variations of trend (none, additive, additive damped) and seasonal components (non, additive, multiplicative), there are nine exponential smoothing methods. And, for each of these methods, errors may be additive or multiplicative.

ETS models in R are handled by ets() from library(forecast). Unlike functions such as naive(), ses(), hw() functions, the ets() function does not produce forecasts. Rather, it estimates the model parameters and returns information on the fitted model.

ETS: AAA

Let us fit an exponential smoothing model where the Errors are Additive, Trend is Additive and Seasonal component is Additive. Note the model argument in ets().

Model

```
ets_aaa = ets(train,model = 'AAA')
```

The ETS model object has methods like other predictive modelling objects such as lm(). Let us examine the coefficients of the model

```
coef(ets_aaa)
```

```
##       alpha        beta       gamma           l           b          s0
##   0.2284115   0.0374727   0.2029944 259.2792901   0.2011150  52.7137253
##          s1          s2
## -20.0075675 -41.6237261
```

ETS Model: AAA

```
> ?ets
>
```

Files | Plots | Packages | Help | Viewer

Refresh Help Topic

R: Exponential smoothing state space model — Find in Topic

y	a numeric vector or time series of class ts
model	Usually a three-character string identifying method using the framework terminology of Hyndman et al. (2002) and Hyndman et al. (2008). The first letter denotes the error type ("A", "M" or "Z"); the second letter denotes the trend type ("N","A","M" or "Z"); and the third letter denotes the season type ("N","A","M" or "Z"). In all cases, "N"=none, "A"=additive, "M"=multiplicative and "Z"=automatically selected. So, for example, "ANN" is simple exponential smoothing with additive errors, "MAM" is multiplicative Holt-Winters' method with multiplicative errors, and so on.

- Each exponential smoothing method described previously could be written as in "innovations state space model.
- ets() provides a way of selecting the best model for a particular time series. It search for model with minimum AICc
- ets() does not provide forecast, it only estimate the parameters, we will need to pass the estimated results to the forecast function

ETS Model: AAA

Table 7.7: State space equations for each of the models in the ETS framework.

ADDITIVE ERROR MODELS

Trend	Seasonal: N	Seasonal: A	Seasonal: M
N	$y_t = \ell_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$	$y_t = \ell_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/\ell_{t-1}$
A	$y_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + b_{t-1})$
A_d	$y_t = \ell_{t-1} + \phi b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + \phi b_{t-1})$

MULTIPLICATIVE ERROR MODELS

Trend	Seasonal: N	Seasonal: A	Seasonal: M
N	$y_t = \ell_{t-1}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$	$y_t = (\ell_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \alpha(\ell_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + s_{t-m})\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A	$y_t = (\ell_{t-1} + b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A_d	$y_t = (\ell_{t-1} + \phi b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$

Summary of the model. Note, since we did not specify the damping term, ets() tried both an Additive-Damped trend and just an Additived trend and went with the option that led to a lower AICc.

```
summary(ets_aaa)
```

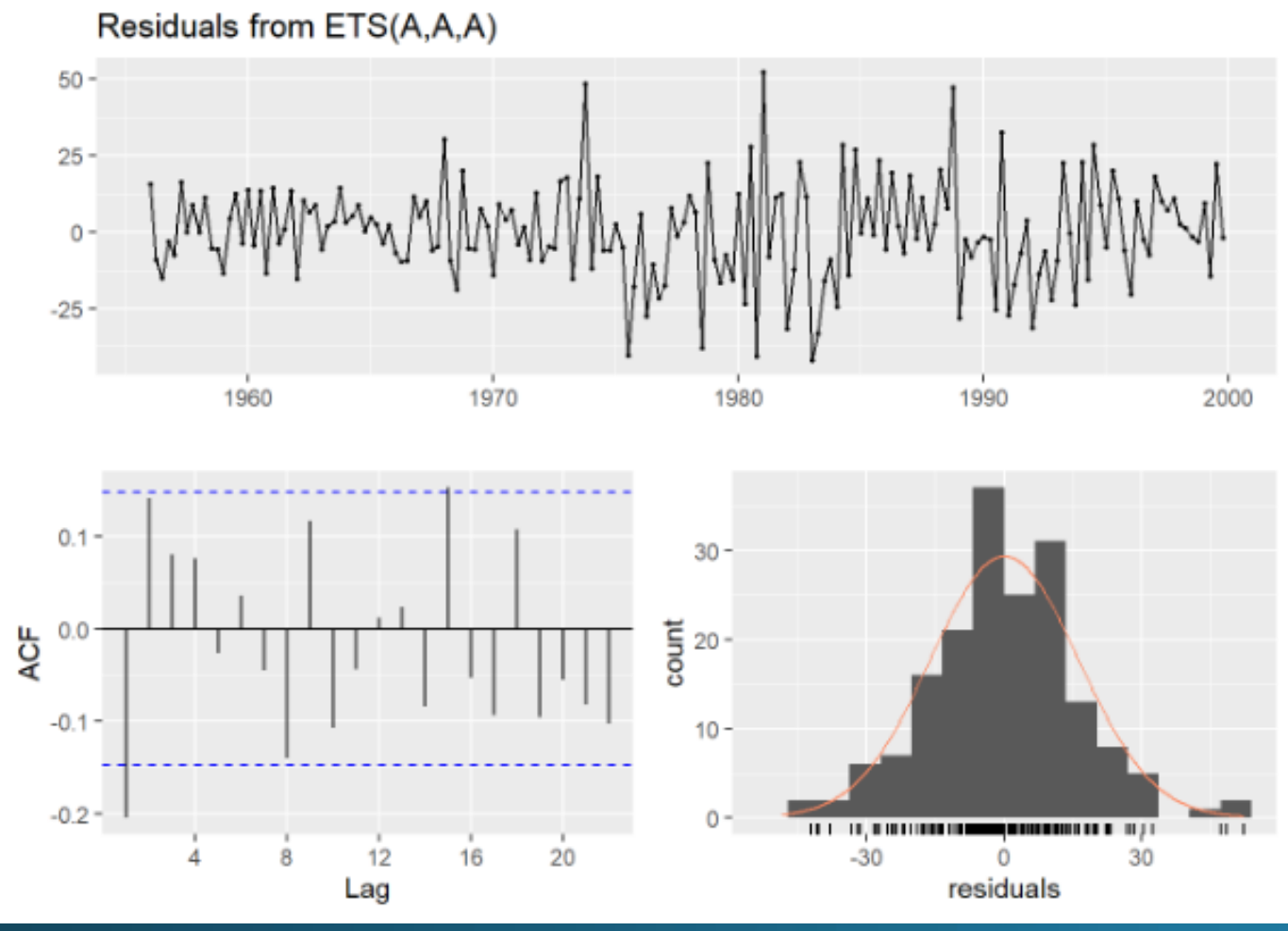
```
## ETS(A,A,A)
##
## Call:
##  ets(y = train, model = "AAA")
##
##   Smoothing parameters:
##     alpha = 0.2284
##     beta  = 0.0375
##     gamma = 0.203
##
##   Initial states:
##     l = 259.2793
##     b = 0.2011
##     s = 52.7137 -20.0076 -41.6237 8.9176
##
##   sigma:  16.5039
##
##      AIC     AICc      BIC
## 1906.683 1907.768 1935.218
##
## Training set error measures:
##                     ME     RMSE      MAE          MPE     MAPE      MASE
## Training set 0.0339053 16.12443 12.43517 -0.02612254 2.989648 0.7786129
##                    ACF1
## Training set -0.2047428
```

- The model with the minimum value of the AIC (AICc) is often the best model for forecasting.
- The AICc corrected for small sample bias

ETS Model: AAA

Examine the residuals.
Based on the ACF plot and a significant Ljung-Box test, residuals are not like white noise.

```
checkresiduals(ets_aaa)
```



- All of these methods for checking residuals are conveniently packaged into one R function checkresiduals(), which will produce a time plot, ACF plot and histogram of the residuals (with an overlaid normal distribution for comparison), and do a Ljung-Box test with the correct degrees of freedom.
- A small p-value (for instance, p-value < .05) indicates the possibility of non-zero autocorrelation within the first m lags. That is, a small p-value indicates that the data are probably not white noise.

H_0 : the autocorrelations up to lag k are all 0
H_A : the autocorrelations of one or more lags differ from 0.

```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,A,A)
## Q* = 22.82, df = 3, p-value = 4.403e-05
##
## Model df: 8.   Total lags used: 11
```

Residuals are not white noise

ETS Model: AAA

Forecast

```
ets_aaa_forecast = forecast(ets_aaa,h=42)
ets_aaa_forecast
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2000 Q1       431.7860 410.6354 452.9365 399.4390 464.1330
## 2000 Q2       396.8128 374.9274 418.6982 363.3419 430.2836
## 2000 Q3       412.7249 389.9184 435.5315 377.8453 447.6045
## 2000 Q4       523.7759 499.8572 547.6946 487.1954 560.3564
## 2001 Q1       433.4849 406.5913 460.3785 392.3547 474.6151
## 2001 Q2       398.5117 370.2168 426.8065 355.2385 441.7849
## 2001 Q3       414.4238 384.5491 444.2985 368.7344 460.1133
## 2001 Q4       525.4748 493.8485 557.1011 477.1066 573.8431
## 2002 Q1       435.1838 399.9788 470.3888 381.3424 489.0251
## 2002 Q2       400.2106 363.0282 437.3930 343.3450 457.0762
## 2002 Q3       416.1227 376.8129 455.4326 356.0035 476.2420
## 2002 Q4       527.1737 485.5943 568.7531 463.5835 590.7640
## 2003 Q1       436.8827 391.3182 482.4472 367.1978 506.5676
## 2003 Q2       401.9095 353.8973 449.9217 328.4811 475.3379
## 2003 Q3       417.8217 367.2366 468.4067 340.4586 495.1848
## 2003 Q4       528.8726 475.5961 582.1492 447.3932 610.3520
## 2004 Q1       438.5816 381.0154 496.1478 350.5416 526.6216
## 2004 Q2       403.6084 343.2020 464.0148 311.2248 495.9920
## 2004 Q3       419.5206 356.1673 482.8739 322.6301 516.4111
## 2004 Q4       530.5715 464.1694 596.9737 429.0182 632.1249
```

the point forecast of the stock price for Q4, 2004

```
## 2005 Q1       440.2805 369.3366 511.2244 331.7812 548.7799
## 2005 Q2       405.3073 331.1844 479.4303 291.9460 518.6686
## 2005 Q3       421.2195 343.8240 498.6149 302.8533 539.5856
## 2005 Q4       532.2705 451.5126 613.0283 408.7620 655.7789
## 2006 Q1       441.9794 356.4573 527.5015 311.1846 572.7742
## 2006 Q2       407.0062 318.0039 496.0086 270.8888 543.1236
## 2006 Q3       422.9184 330.3522 515.4846 281.3506 564.4862
## 2006 Q4       533.9694 437.7585 630.1803 386.8275 681.1112
## 2007 Q1       443.6783 342.4990 544.8577 288.9379 598.4188
## 2007 Q2       408.7051 303.7720 513.6383 248.2238 569.1865
## 2007 Q3       424.6173 315.8543 533.3803 258.2787 590.9559
## 2007 Q4       535.6683 423.0016 648.3350 363.3594 707.9771
## 2008 Q1       445.3772 327.5506 563.2039 265.1769 625.5775
## 2008 Q2       410.4040 288.5712 532.2369 224.0768 596.7313
## 2008 Q3       426.3162 300.4071 552.2253 233.7548 618.8776
## 2008 Q4       537.3672 407.3135 667.4209 338.4672 736.2672
## 2009 Q1       447.0761 311.6804 582.4719 240.0062 654.1461
## 2009 Q2       412.1030 272.4654 551.7405 198.5457 625.6602
## 2009 Q3       428.0151 284.0704 571.9598 207.8707 648.1595
## 2009 Q4       539.0661 390.7505 687.3816 312.2370 765.8951
## 2010 Q1       448.7751 294.9428 602.6073 213.5089 684.0412
## 2010 Q2       413.8019 255.5059 572.0979 171.7090 655.8947
```

- RMSE of the ets_aaa model on the test sample

Accuracy

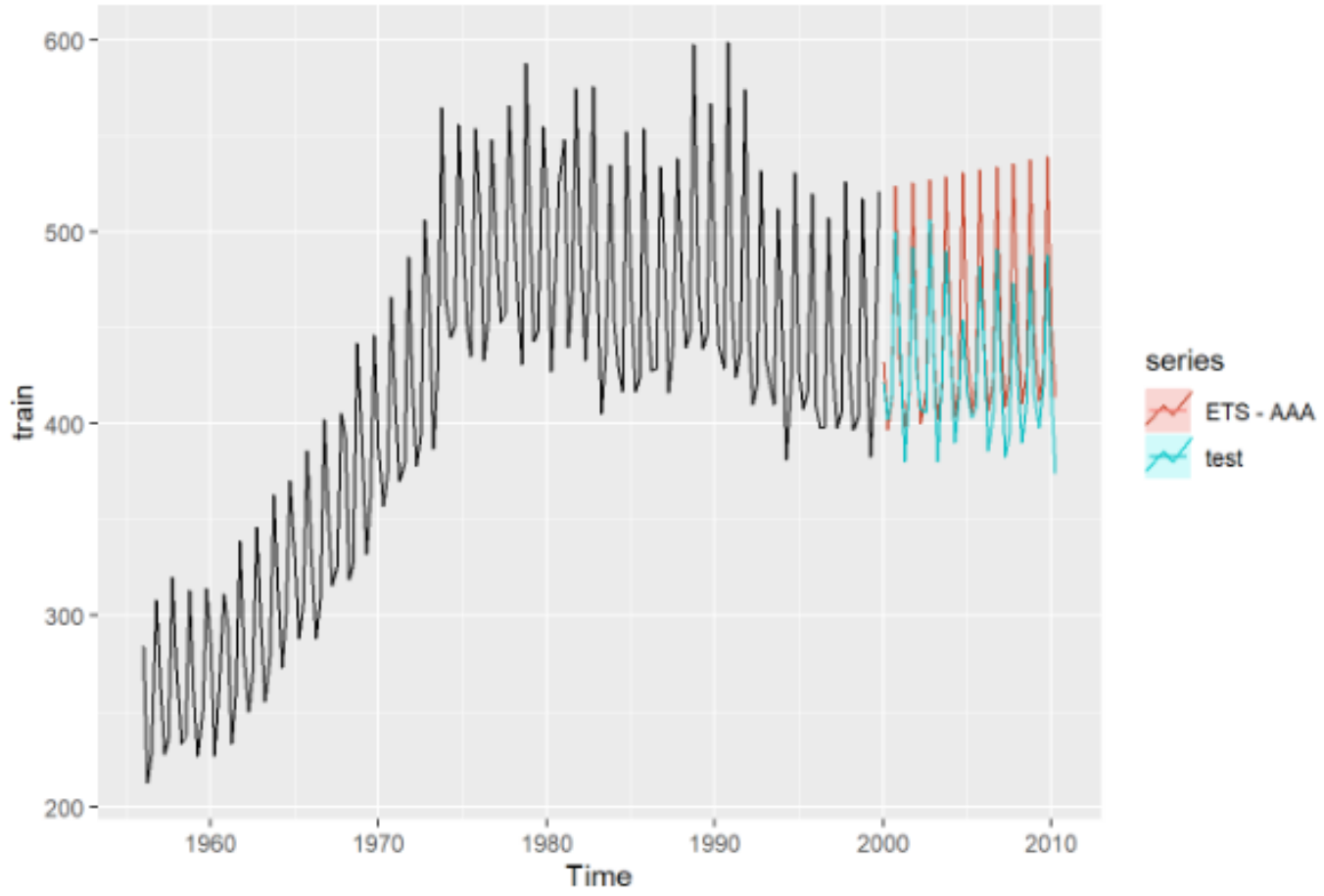
```
accuracy(ets_aaa_forecast,x = ausbeer)
```

```
##                      ME     RMSE      MAE          MPE     MAPE      MASE
## Training set   0.0339053 16.12443 12.43517 -0.02612254 2.989648 0.7786129
## Test set     -20.3899116 28.22830 22.12902 -4.65481860 5.060769 1.3855813
##                    ACF1 Theil's U
## Training set -0.20474276        NA
## Test set      0.01545558 0.5492527
```

ETS Model: AAA

Visualize Forecast

```
autoplot(train)+
  autolayer(ets_aaa_forecast,series="ETS - AAA",PI=F)+
  autolayer(test)
```



ETS Model: Automatic Selection

ETS: Automatic Selection

When only the time-series is specified, and all other arguments are left at their default values, then ets() will automatically select the best model based on AICc. Compare the AICc for this model to ETS(A,A,A) above. Note that the chosen model has Multiplicative Errors, Additive Trend, and Multiplicative Seasonal.

Model

```
ets_auto = ets(train)
summary(ets_auto)
```

```
## ETS(M,A,M)
##
## Call:
##  ets(y = train)
##
##   Smoothing parameters:
##     alpha = 0.2263
##     beta  = 0.035
##     gamma = 0.1766
##
##   Initial states:
##     l = 255.5524
##     b = 0.8861
##     s = 1.1819 0.9125 0.8628 1.0428
##
##   sigma:  0.0371
##
##      AIC     AICc      BIC
## 1869.636 1870.720 1898.170
##
## Training set error measures:
##                      ME    RMSE      MAE         MPE    MAPE      MASE
## Training set -0.2090563 15.8621 12.12096 -0.04973312 2.91519 0.758939
##                    ACF1
## Training set -0.1999151
```

ETS Model: Automatic Selection

Forecast

```
ets_auto_forecast = forecast(ets_auto,h=42)
```

Accuracy

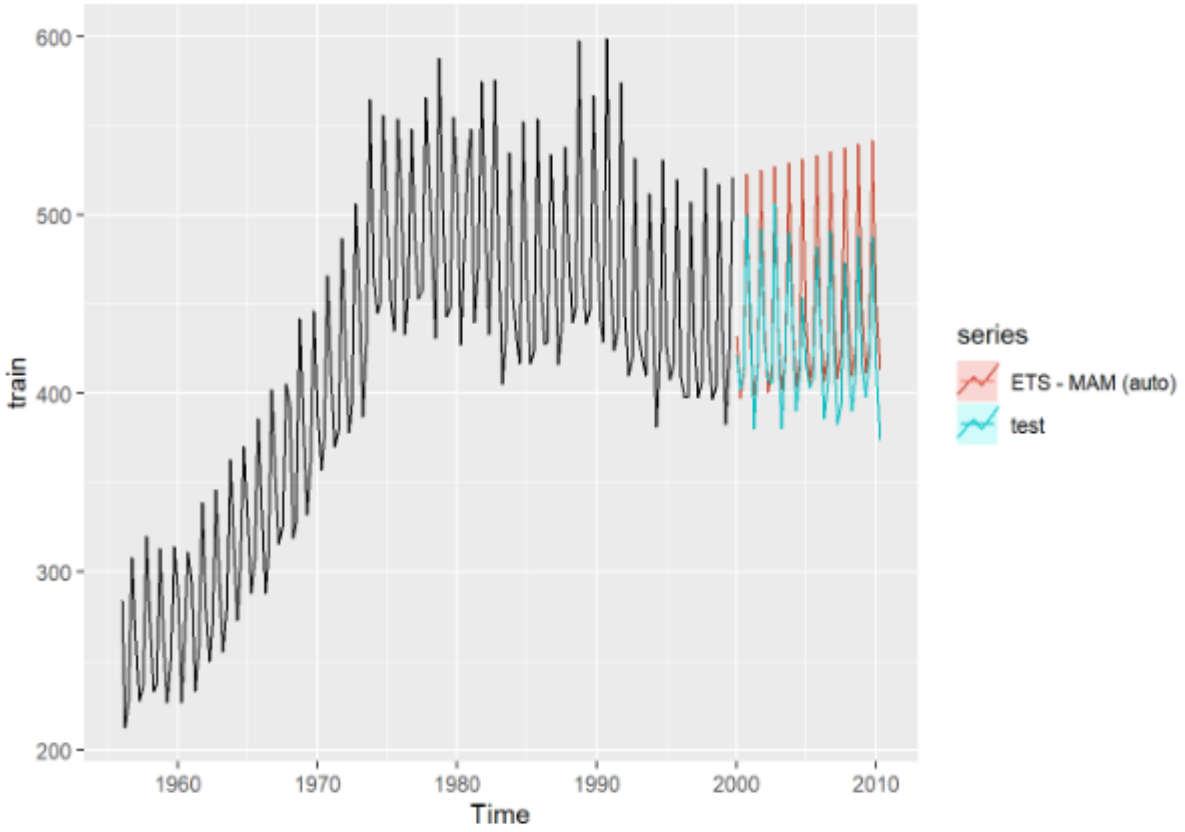
Even though ets_auto has a lower AICc, its RMSE is worse than that of ETS(A,A,A)

```
accuracy(ets_auto_forecast,x = ausbeer)
```

```
##                      ME     RMSE      MAE         MPE     MAPE     MASE
## Training set  -0.2090563 15.86210 12.12096 -0.04973312 2.915190 0.758939
## Test set     -20.4860097 28.61288 22.23155 -4.66935195 5.076505 1.392001
##                    ACF1 Theil's U
## Training set -0.1999151        NA
## Test set      0.0102163  0.556942
```

Visualize Forecast

```
autoplot(train)+
  autolayer(ets_auto_forecast,series="ETS - MAM (auto)",PI=F)+
  autolayer(test)
```



ARIMA

- Exponential Smoothing and ARIMA are the two most widely used approaches to time-series forecasting and provide complementary approaches to the problem.
- While Exponential Smoothing models are based on a description of trend and seasonality in the data, ARIMA models aim to describe autocorrelations in the data

Stationary Process

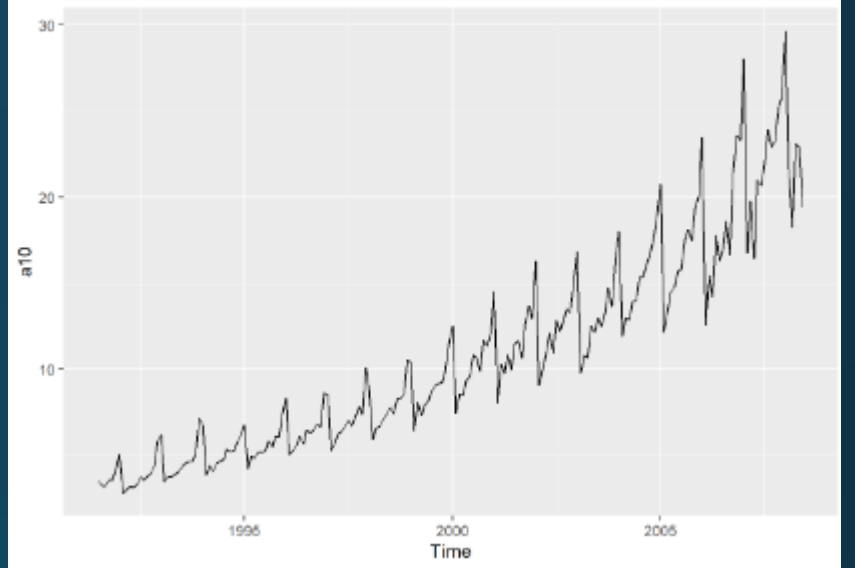
Since an ARMA process assumes the data is stationary, let us examine the assumption of stationarity and how to satisfy it.

- Stationary Process has constant mean, variance and covariance (between identically spaced data points) over time. Stationarity is an assumption of many time-series analysis procedures. However, most time-series are non-stationary
- Fortunately, a non-stationary process can be transformed into a (weakly) stationary process through transformations that remove trend and stabilize variance.
- We will stabilize variance using a Box-Cox Transformation and remove seasonality and trend using differencing

ARIMA: Stationary Process

- Here, you will use a Box-Cox transformation to stabilize the variance of the pre-loaded a10 series, which contains monthly anti-diabetic drug sales in Australia from 1991-2008.

```
autoplot(a10)
```



Making a Series Stationary: Example 1

Original Series

```
a10
```

```
##          Jan       Feb       Mar       Apr       May       Jun       Jul
## 1991                                                             3.526591
## 1992  5.088335  2.814520  2.985811  3.204780  3.127578  3.270523  3.737851
## 1993  6.192068  3.450857  3.772307  3.734303  3.905399  4.049687  4.315566
## 1994  6.731473  3.841278  4.394076  4.075341  4.540645  4.645615  4.752607
## 1995  6.749484  4.216067  4.949349  4.823045  5.194754  5.170787  5.256742
## 1996  8.329452  5.069796  5.262557  5.597126  6.110296  5.689161  6.486849
## 1997  8.524471  5.277918  5.714303  6.214529  6.411929  6.667716  7.050831
## 1998  8.798513  5.918261  6.534493  6.675736  7.064201  7.383381  7.813496
## 1999 10.391416  6.421535  8.062619  7.297739  7.936916  8.165323  8.717420
## 2000 12.511462  7.457199  8.591191  8.474000  9.386803  9.560399 10.834295
## 2001 14.497581  8.049275 10.312891  9.753358 10.850382  9.961719 11.443601
## 2002 16.300269  9.053485 10.002449 10.788750 12.106705 10.954101 12.844566
## 2003 16.828350  9.800215 10.816994 10.654223 12.512323 12.161210 12.998046
## 2004 18.003768 11.938030 12.997900 12.882645 13.943447 13.989472 15.339097
## 2005 20.778723 12.154552 13.402392 14.459239 14.795102 15.705248 15.829550
## 2006 23.486694 12.536987 15.467018 14.233539 17.783058 16.291602 16.980282
## 2007 28.038383 16.763869 19.792754 16.427305 21.000742 20.681002 21.834890
## 2008 29.665356 21.654285 18.264945 23.107677 22.912510 19.431740
```

```
##          Aug       Sep       Oct       Nov       Dec
## 1991  3.180891  3.252221  3.611003  3.565869  4.306371
## 1992  3.558776  3.777202  3.924490  4.386531  5.810549
## 1993  4.562185  4.608662  4.667851  5.093841  7.179962
## 1994  5.350605  5.204455  5.301651  5.773742  6.204593
## 1995  5.855277  5.490729  6.115293  6.088473  7.416598
## 1996  6.300569  6.467476  6.828629  6.649078  8.606937
## 1997  6.704919  7.250988  7.819733  7.398101 10.096233
## 1998  7.431892  8.275117  8.260441  8.596156 10.558939
## 1999  9.070964  9.177113  9.251887  9.933136 11.532974
## 2000 10.643751  9.908162 11.710041 11.340151 12.079132
## 2001 11.659239 10.647060 12.652134 13.674466 12.965735
## 2002 12.196500 12.854748 13.542004 13.287640 15.134918
## 2003 12.517276 13.268658 14.733622 13.669382 16.503966
## 2004 15.370764 16.142005 16.685754 17.636728 18.869325
## 2005 17.554701 18.100864 17.496668 19.347265 20.031291
## 2006 18.612189 16.623343 21.430241 23.575517 23.334206
## 2007 23.930204 22.930357 23.263340 25.250030 25.806090
## 2008
```

ARIMA: Stationary Process

Stabilize Variance

```
a10_const_var = BoxCox(a10,lambda = BoxCox.lambda(a10))
a10_const_var
```

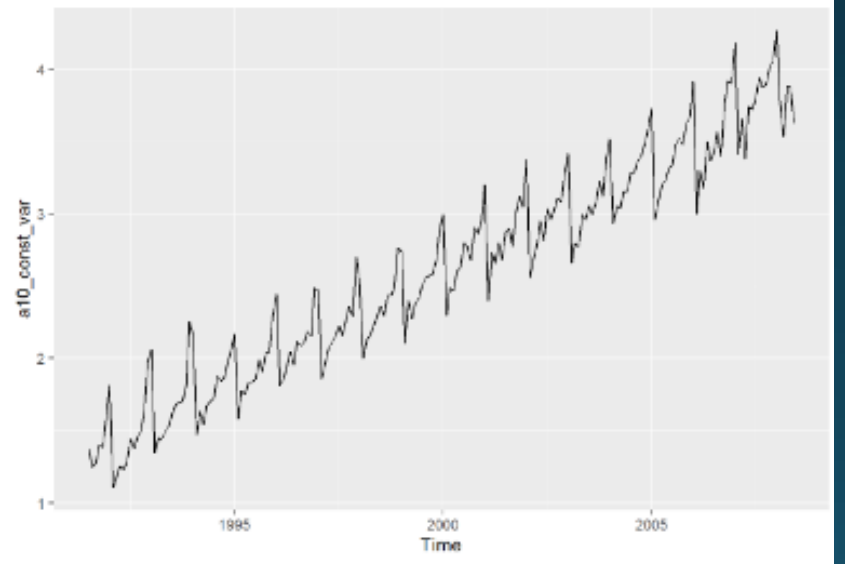
```
##           Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1991                                                      1.370640 1.249719
## 1992 1.813838 1.108403 1.176346 1.258433 1.230064 1.282127 1.439555 1.381367
## 1993 2.060083 1.345059 1.450472 1.438425 1.491844 1.535336 1.612066 1.679652
## 1994 2.166791 1.472067 1.633938 1.542927 1.673877 1.701798 1.729698 1.876275
## 1995 2.170224 1.583846 1.779608 1.747770 1.839502 1.833762 1.854241 1.989287
## 1996 2.444272 1.809319 1.855616 1.932585 2.043207 1.953056 2.119355 2.082179
## 1997 2.474891 1.859242 1.958598 2.064685 2.104516 2.154573 2.226514 2.161715
## 1998 2.516905 2.002791 2.128714 2.156116 2.228963 2.286257 2.360153 2.294776
## 1999 2.740742 2.106427 2.401353 2.271098 2.380705 2.418016 2.504592 2.557562
## 2000 2.996340 2.299201 2.485227 2.467026 2.603388 2.628007 2.797657 2.773422
## 2001 3.203656 2.399174 2.730431 2.654922 2.799686 2.683470 2.872744 2.898487
## 2002 3.371455 2.554986 2.688989 2.791898 2.950612 2.812706 3.033019 2.960871
## 2003 3.417552 2.661387 2.795472 2.774764 2.996436 2.956847 3.049642 2.996988
## 2004 3.515807 2.931163 3.049626 3.037159 3.148428 3.153087 3.284117 3.287069
## 2005 3.727330 2.956087 3.092632 3.199894 3.232556 3.317933 3.329258 3.478946
## 2006 3.911279 2.999181 3.296010 3.177573 3.497791 3.370688 3.430582 3.564495
## 2007 4.182556 3.411991 3.655150 3.382661 3.743172 3.720310 3.801421 3.939630
## 2008 4.270269 3.788975 3.536880 3.886679 3.873874 3.627938
```

```
##           Sep      Oct      Nov      Dec
## 1991 1.275573 1.398595 1.383719 1.609482
## 1992 1.452016 1.497678 1.631851 1.979620
## 1993 1.692032 1.707642 1.815177 2.249997
## 1994 1.841819 1.864826 1.971617 2.062651
## 1995 1.908552 2.044244 2.038670 2.292095
## 1996 2.115532 2.185216 2.150983 2.487656
## 1997 2.262758 2.361199 2.288847 2.701625
## 1998 2.435630 2.433288 2.485993 2.762514
## 1999 2.573116 2.583979 2.679584 2.883464
## 2000 2.676181 2.904492 2.860244 2.947449
## 2001 2.773846 3.011932 3.120929 3.046157
## 2002 3.034127 3.107214 3.080547 3.264953
## 2003 3.078540 3.226626 3.120405 3.389388
## 2004 3.357387 3.405230 3.485739 3.584657
## 2005 3.523672 3.474122 3.621507 3.672894
## 2006 3.399808 3.773410 3.916994 3.901423
## 2007 3.875049 3.896824 4.021381 4.054717
## 2008
```

- BoxCox.lambda() will give best value of lambda

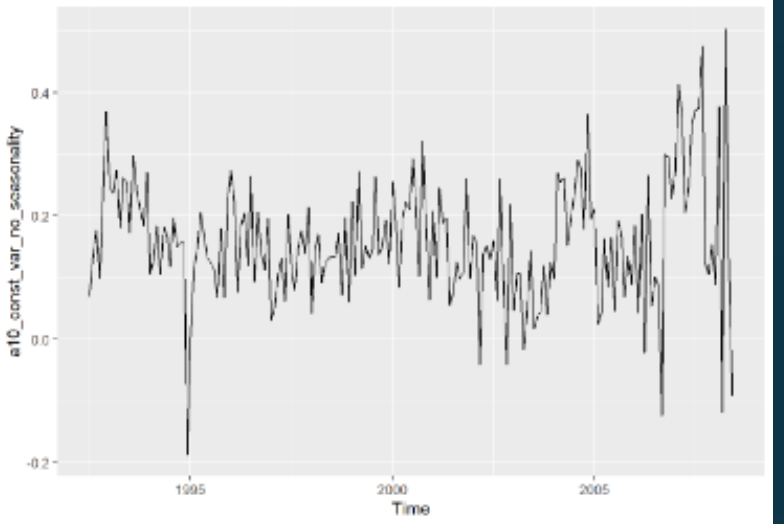
```
> BoxCox.lambda(a10)
[1] 0.1313326
```

```
autoplot(a10_const_var)
```



ARIMA: Stationary Process

```
autoplot(a10_const_var_no_seasonality)
```



Remove Seasonality

```
a10_const_var_no_seasonality = diff(x = a10_const_var,lag = 12)
a10_const_var_no_seasonality
```

```
##              Jan          Feb          Mar          Apr          May
## 1992
## 1993  0.246245444  0.236655903  0.274125392  0.179992291  0.261780298
## 1994  0.106707317  0.127008025  0.183466224  0.104501404  0.182032776
## 1995  0.003433011  0.111778361  0.145670228  0.204842989  0.165625047
## 1996  0.274048279  0.225473639  0.076007964  0.184814821  0.203705337
## 1997  0.030619086  0.049922844  0.102981627  0.132100224  0.061308706
## 1998  0.042013647  0.143548717  0.170116222  0.091430829  0.124446414
## 1999  0.223837120  0.103636140  0.272638470  0.114982618  0.151741933
## 2000  0.255598717  0.192773618  0.083874135  0.195927270  0.222683128
## 2001  0.207315424  0.099973328  0.245204433  0.187895960  0.196298560
## 2002  0.167799083  0.155812045 -0.041441862  0.136976673  0.150926216
## 2003  0.046097124  0.106401350  0.106482654 -0.017134319  0.045823888
## 2004  0.098254769  0.269775707  0.254154218  0.262395495  0.151991548
## 2005  0.211523067  0.024923542  0.043005821  0.162734647  0.084128115
## 2006  0.183948732  0.043094279  0.203377923 -0.022321033  0.265235151
## 2007  0.271276841  0.412810508  0.359139697  0.205088323  0.245380862
## 2008  0.087713881  0.376983677 -0.118270012  0.504017315  0.130702472
```

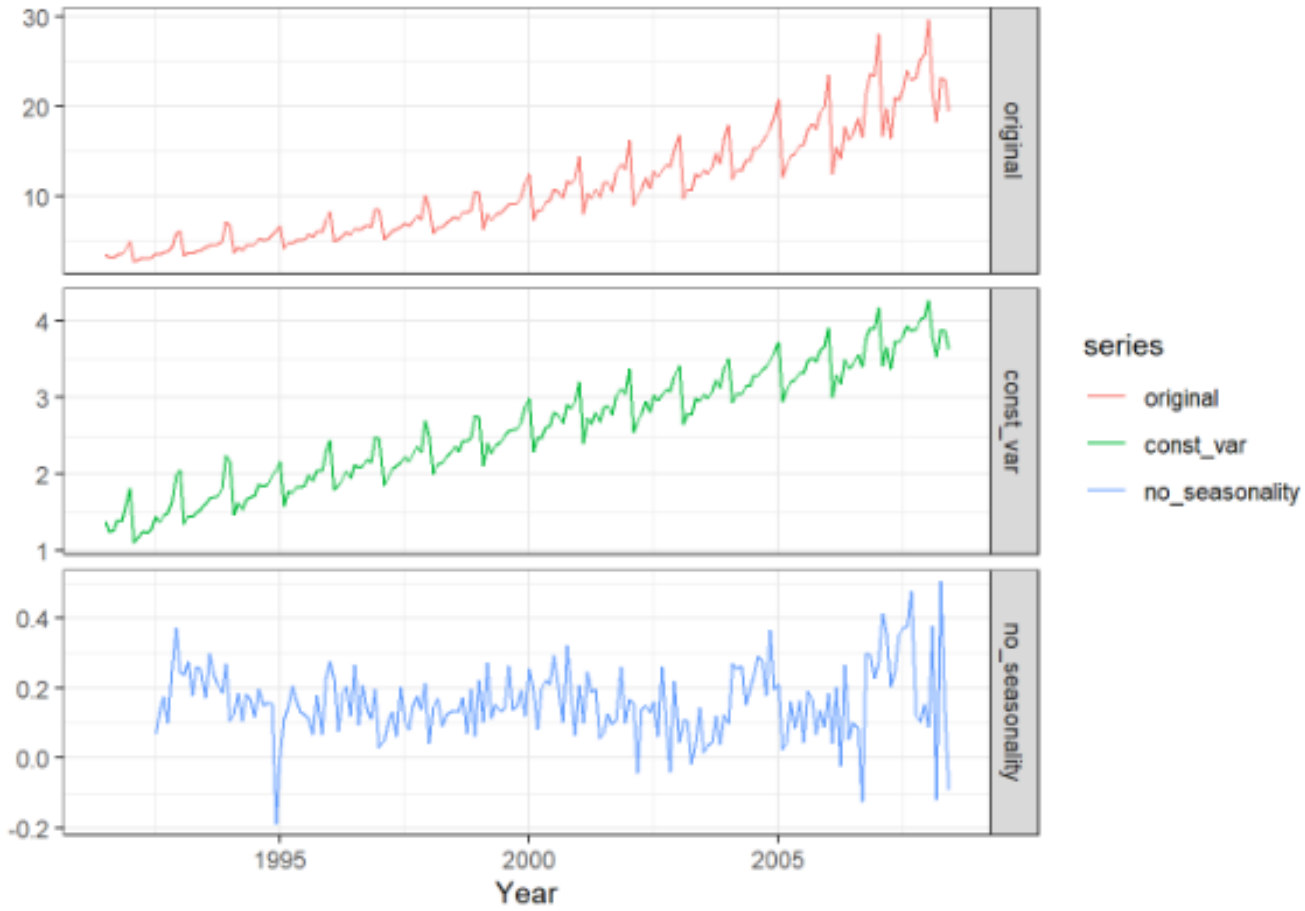
```
##              Jun          Jul          Aug          Sep          Oct
## 1992                0.068914872  0.131647730  0.176442705  0.099083250
## 1993  0.253208354  0.172511836  0.298285240  0.240016156  0.209963892
## 1994  0.166462225  0.117631639  0.196623325  0.149786828  0.157183802
## 1995  0.131964341  0.124543290  0.113011767  0.066732755  0.179418490
## 1996  0.119293319  0.265113644  0.092892385  0.206980636  0.140971432
## 1997  0.201517801  0.107158949  0.079535272  0.147225882  0.175982731
## 1998  0.131683621  0.133639241  0.133061334  0.172872087  0.072088989
## 1999  0.131759208  0.144438879  0.262786380  0.137485989  0.150691433
## 2000  0.209991223  0.293065044  0.215860117  0.103065352  0.320512593
## 2001  0.055462118  0.075086949  0.125064642  0.097664934  0.107440890
## 2002  0.129236467  0.160275193  0.062383518  0.260280947  0.095281699
## 2003  0.144140785  0.016622688  0.036117330  0.044412227  0.119411497
## 2004  0.196240086  0.234474574  0.290080889  0.278847563  0.178604272
## 2005  0.164846565  0.045141607  0.191876814  0.166284639  0.068892416
## 2006  0.052754167  0.101323581  0.085549371 -0.123863817  0.299287705
## 2007  0.349622833  0.370839029  0.375135514  0.475241292  0.123414028
## 2008 -0.092372655
```

```
##              Nov          Dec
## 1992  0.248131960  0.370137637
## 1993  0.183326086  0.270377420
## 1994  0.156439404 -0.187346253
## 1995  0.067053891  0.229444081
## 1996  0.112312454  0.195560601
## 1997  0.137864109  0.213969434
## 1998  0.197146054  0.060888612
## 1999  0.193591168  0.120950198
## 2000  0.180659912  0.063985039
## 2001  0.260685014  0.098707526
## 2002 -0.040381829  0.218796465
## 2003  0.039857509  0.124434429
## 2004  0.365334465  0.195269583
## 2005  0.135767731  0.088236836
## 2006  0.295486720  0.228529224
## 2007  0.104387656  0.153293541
## 2008
```

ARIMA: Stationary Process

See the changes

```
dat = cbind(original = a10,
      const_var = a10_const_var,
      no_seasonality = a10_const_var_no_seasonality)
library(ggthemes)
autoplot(dat,facets=T,colour = T)+ylab('')+xlab('Year')+theme_bw()
```



There are automatic functions to estimate the level of differencing required to make a series non-stationary

```
nsdiffs(a10_const_var)
```

```
## [1] 1
```

```
ndiffs(a10_const_var_no_seasonality)
```

```
## [1] 0
```

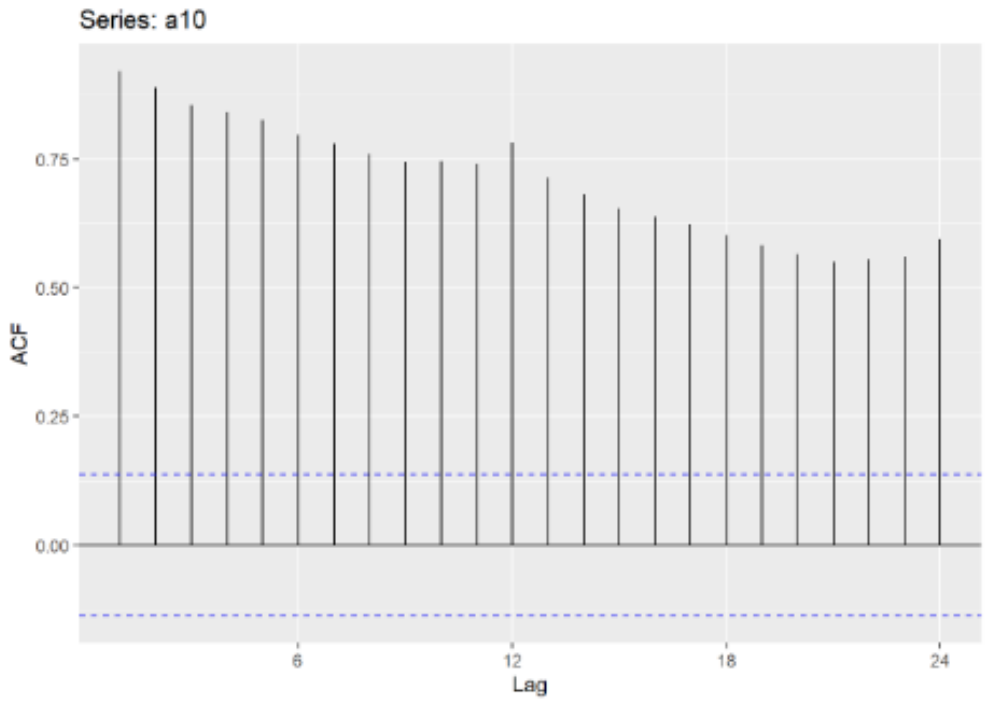
nsdiffs()
- uses to determine the appropriate number of seasonal differences required to make a given time series stationary

ndiffs()
– use to determine the appropriate number of first differences required to make a given time series stationary

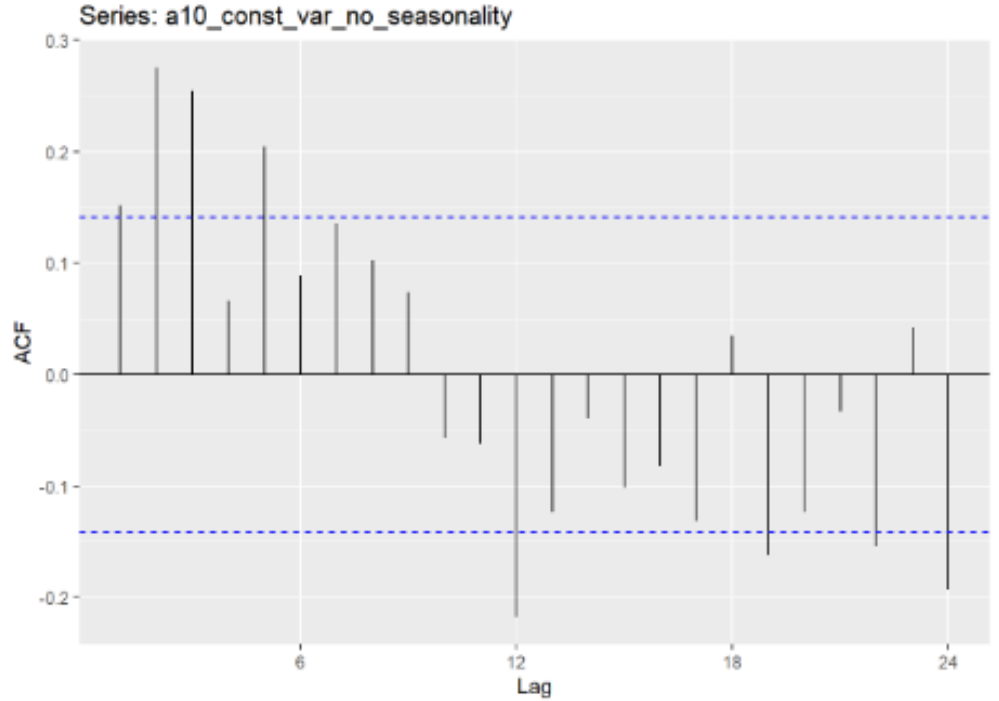
ARIMA: Stationary Process: ACF Plot and KPSS Test

Or, one can examine ACF plots to see if the data is stationary

```
ggAcf(a10)
```




```
ggAcf(a10_const_var_no_seasonality)
```



KPSS Test for Stationarity Non-significant test statistic indicates the differenced data is stationary.

```
kpss.test(a10_const_var_no_seasonality)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  a10_const_var_no_seasonality
## KPSS Level = 0.14853, Truncation lag parameter = 4, p-value = 0.1
```

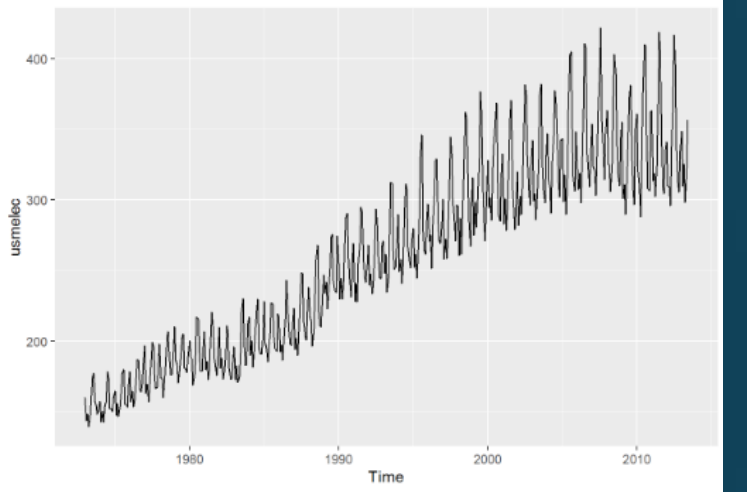
H_0: data is stationary

Making a Series Stationary: Example 2

Let us use another example to illustrate.

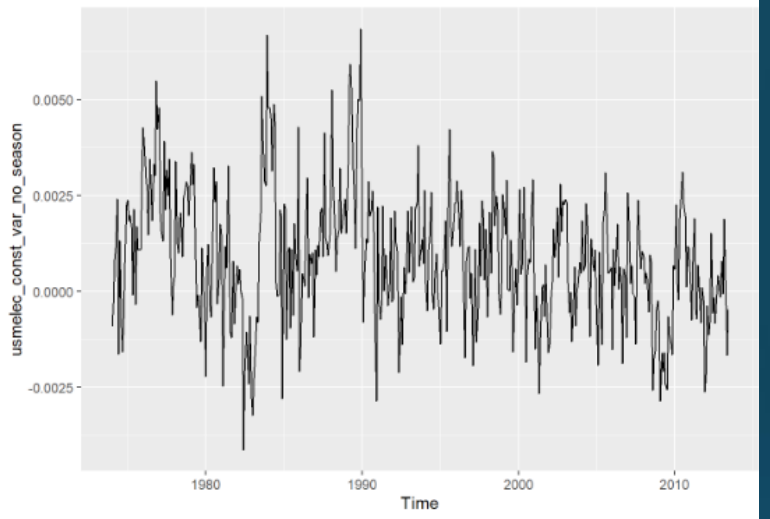
Original Series

```
autoplot(usmelec)
```



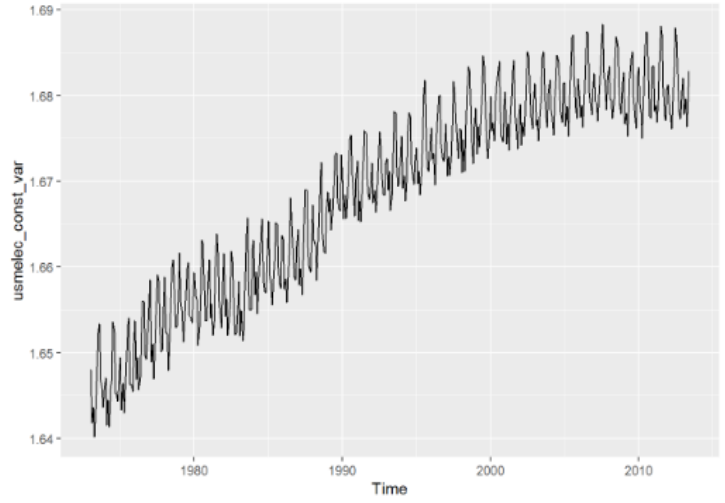
Seasonal Differencing

```
usmelec_const_var_no_season = diff(usmelec_const_var,lag = 12)
autoplot(usmelec_const_var_no_season)
```



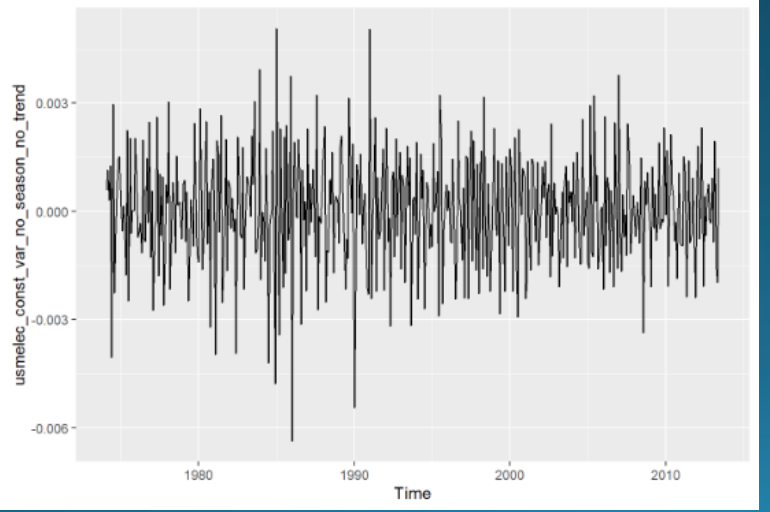
Stabilize Variance

```
usmelec_const_var = BoxCox(usmelec,lambda = BoxCox.lambda(usmelec))
autoplot(usmelec_const_var)
```



Lag-1 Differencing

```
usmelec_const_var_no_season_no_trend = diff(usmelec_const_var_no_season,1)
autoplot(usmelec_const_var_no_season_no_trend)
```



usmelec {fpp2}

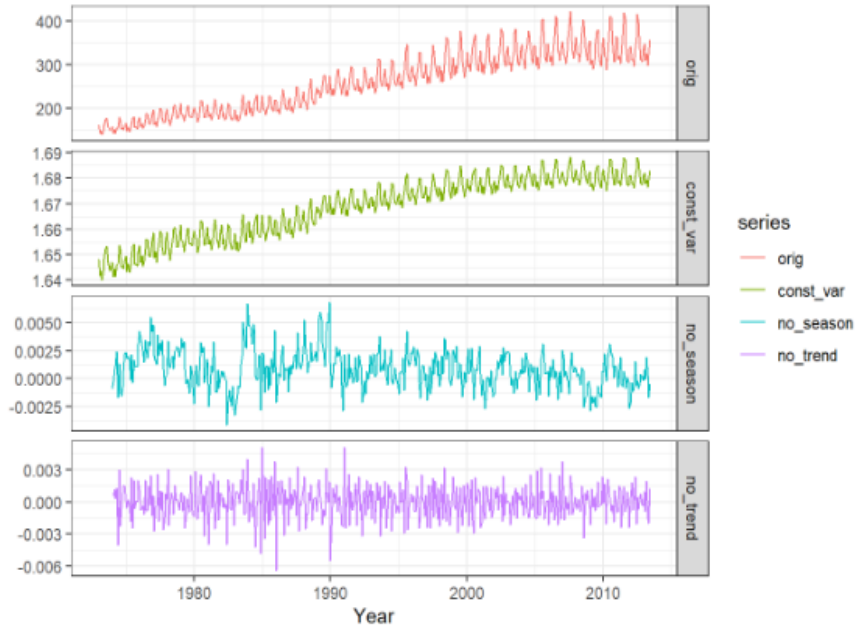
Electricity monthly total net generation. January 1973 - June 2013.

Description

Electricity net generation measured in billions of kilowatt hours (kWh).

See the change

```
dat = cbind(orig = usmelec,
      const_var = usmelec_const_var,
      no_season = usmelec_const_var_no_season,
      no_trend = usmelec_const_var_no_season_no_trend)
autoplot(dat,colour = T,facets = T)+xlab('Year')+ylab('')+theme_bw()
```



KPSS Test for Stationarity

Non-significant test statistic indicates series is now stationary and differencing is no longer needed.

```
kpss.test(usmelec_const_var_no_season_no_trend)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  usmelec_const_var_no_season_no_trend
## KPSS Level = 0.01669, Truncation lag parameter = 5, p-value = 0.1
```

H_o: data is stationary

ARIMA: Fitting a Model

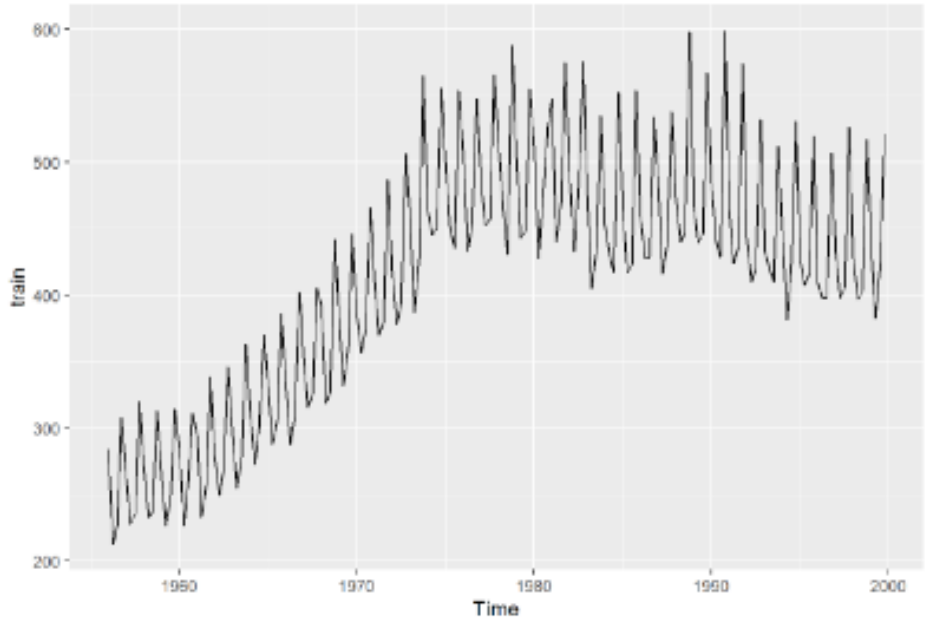
- Using ausbeer data set

ARIMA Model

Fit an ARIMA model to ausbeer

Plot data to identify any unusual observations

```
autoplot(train)
```



Files | Plots | Packages | Help | Viewer

R: Quarterly Australian Beer production — Find in Topic

ausbeer {fpp2}

Quarterly Australian Beer production

Description

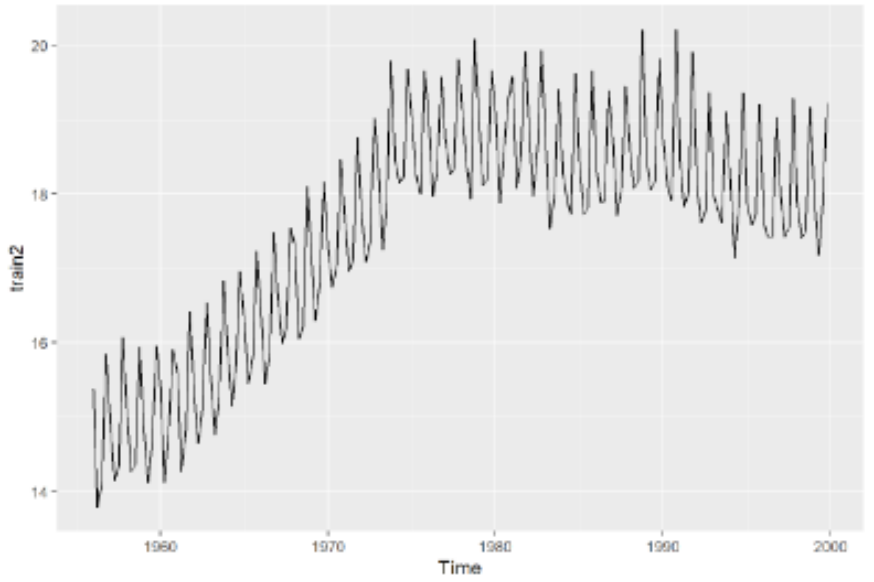
Total quarterly beer production in Australia (in megalitres) from 1956:Q1 to 2010:Q2.

Format

Quarterly time series of class `ts`.

Stabilize Variance

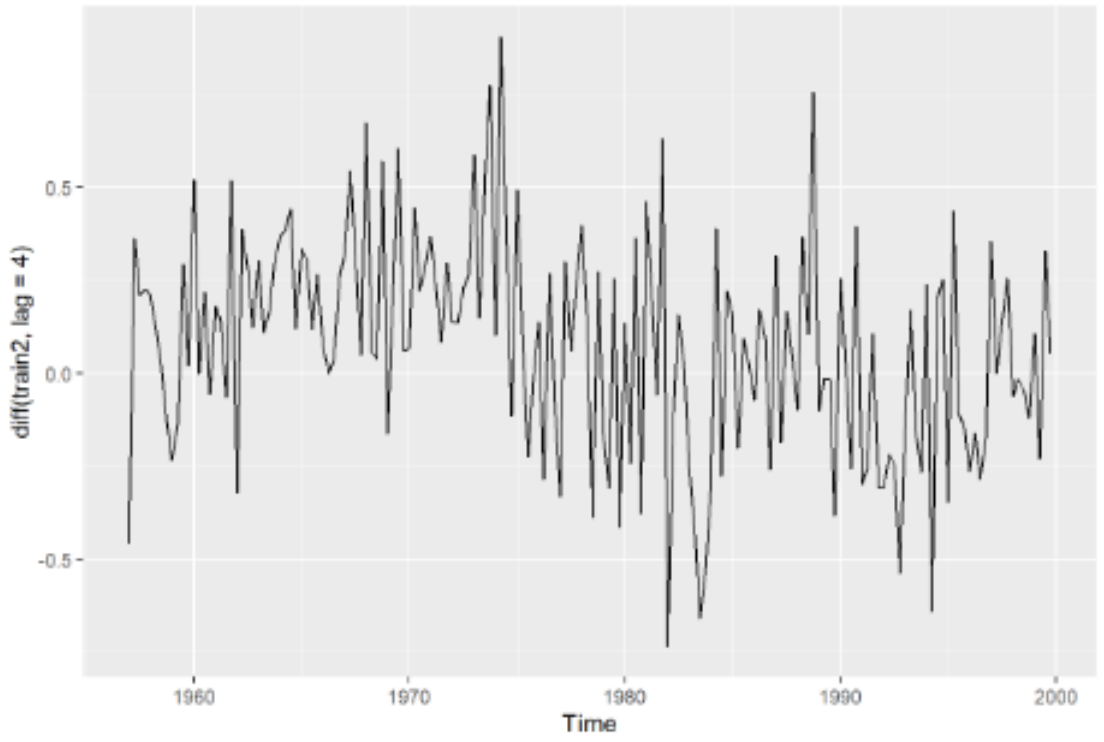
```
train2 = BoxCox(train,lambda = BoxCox.lambda(train))
autoplot(train2)
```



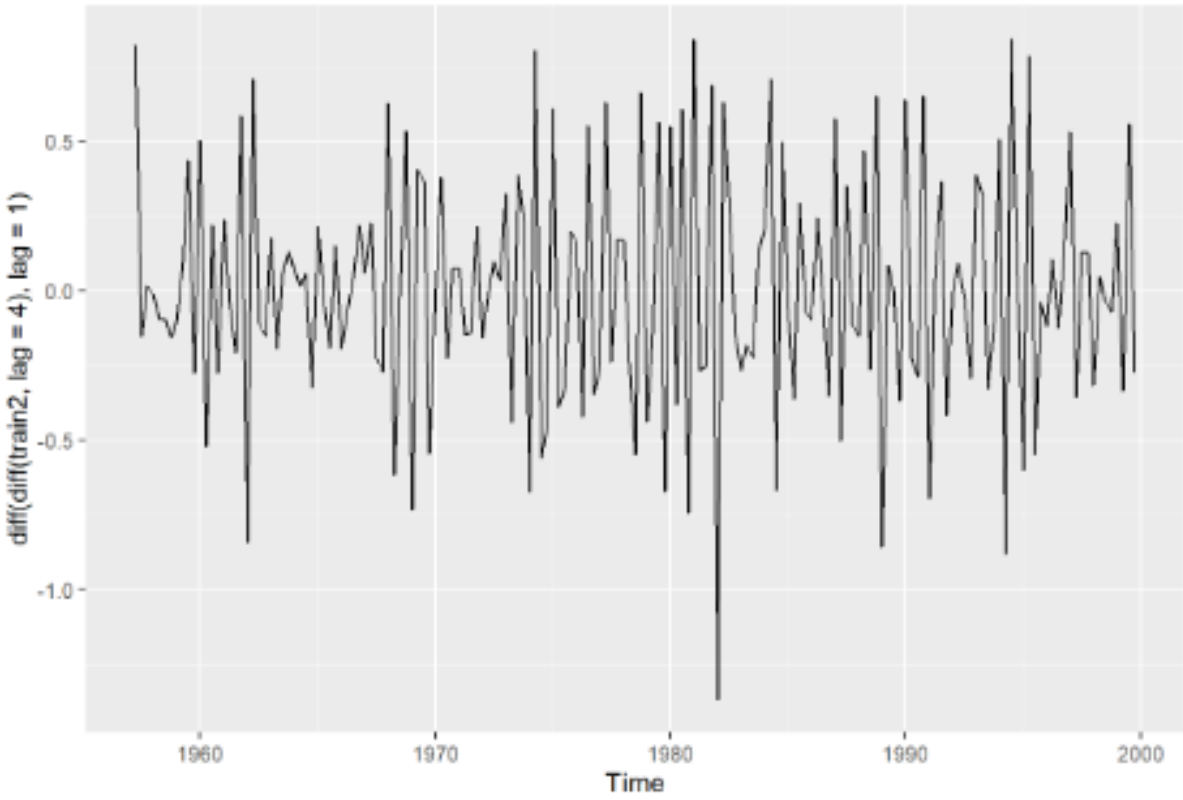
ARIMA: Fitting a Model

If the data are non-stationary, take first differences until the data are stationary

```
autoplot(diff(train2,lag = 4))
```




```
autoplot(diff(diff(train2,lag = 4),lag=1))
```



```
nsdiffs(train)
```

```
## [1] 1
```

```
nsdiffs(diff(train,lag = 4))
```

```
## [1] 0
```

```
ndiffs(diff(train,lag = 4))
```

```
## [1] 1
```

```
ndiffs(diff(diff(train,lag = 4),lag=1))
```

```
## [1] 0
```

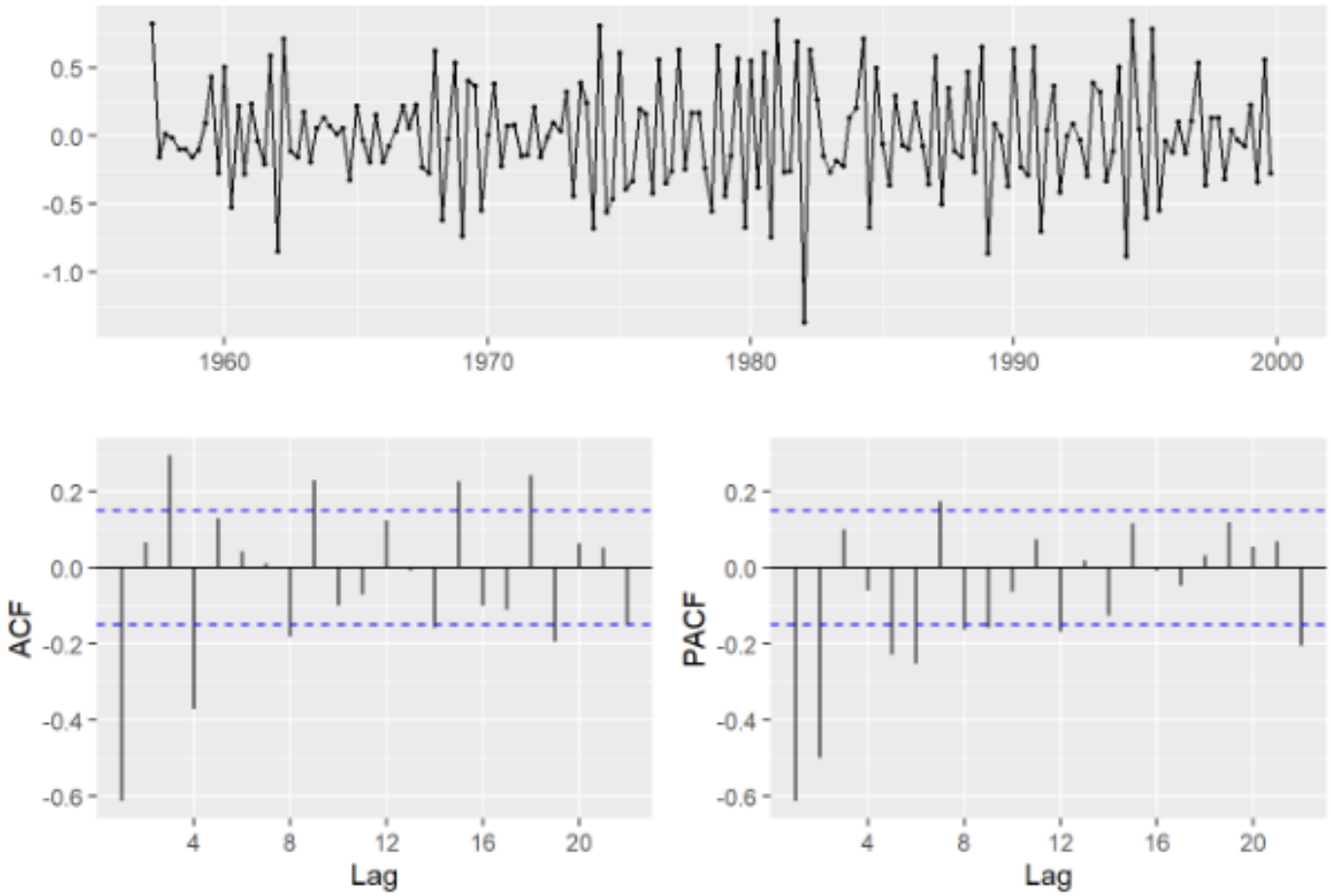
nsdiffs()
- uses to determine the appropriate number of seasonal differences required to make a given time series stationary

ndiffs()
– use to determine the appropriate number of first differences required to make a given time series stationary

ARIMA: Fitting a Model

Examine ACF and PACF to decide on AR (ARIMA(p,d,0) and/or MA terms (ARIMA(0,d,q))

```
train2 %>%
  diff(lag=4)%>%
  diff(lag=1)%>%
  ggtsdisplay()
```



ARIMA: Fitting a Model

Try chosen model and use AICc to search for a better model

```
model1 = Arima(y = train,order = c(0,1,2),seasonal = c(0,1,1),lambda = BoxCox.lambda(train))
model1
```

$(p,d,q)(P,D,Q)[m]$:

```
## Series: train
## ARIMA(0,1,2)(0,1,1)[4]
## Box Cox transformation: lambda= 0.3102148
##
## Coefficients:
##           ma1     ma2     sma1
##       -0.9665  0.4085  -0.8268
## s.e.   0.0713  0.0729   0.0499
##
## sigma^2 estimated as 0.05151:  log likelihood=9.93
## AIC=-11.86   AICc=-11.62   BIC=0.71
```

- The model is ARIMA (0,1,2)(0,1,1)

  p = 0 indicates that no ordinary AR lag is being used.

  d = 1 indicates 1 ordinary difference is being used.

  q = 2 indicates 2 ordinary MA lags are being used.

  P = 0 indicates that no seasonal AR lag is being used

  D = 1 indicates 1 seasonal difference is being used.

  Q = 1 indicates 1 seasonal MA lag is being used.

Seasonal ARIMA

- Non-Seasonal ARIMA + seasonal terms = Seasonal ARIMA

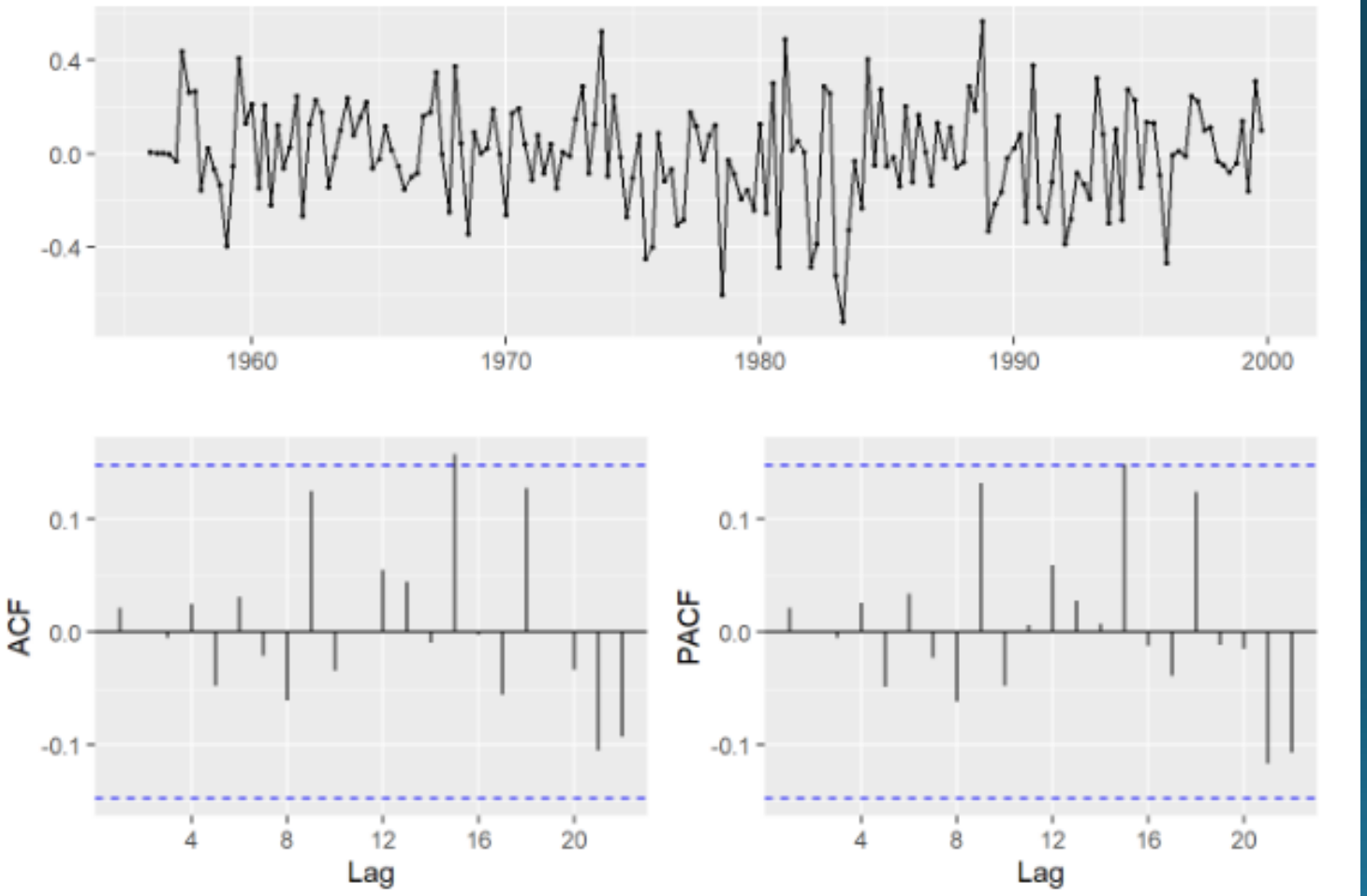
- ARIMA(p,d,q)(P,D,Q)m
  - (P,D,Q) is seasonal component
  - m is seasonal period

  where

  - p = Number of ordinary Autoregressive lags
  - d = Number of ordinary differences
  - q = Number of ordinary Moving average lags
  - P = Number of seasonal Autoregressive lags
  - D = Number of seasonal differences
  - Q = Number of seasonal Moving Average lags
  - m = Seasonal period or number of observations per year

ARIMA: Fitting a Model

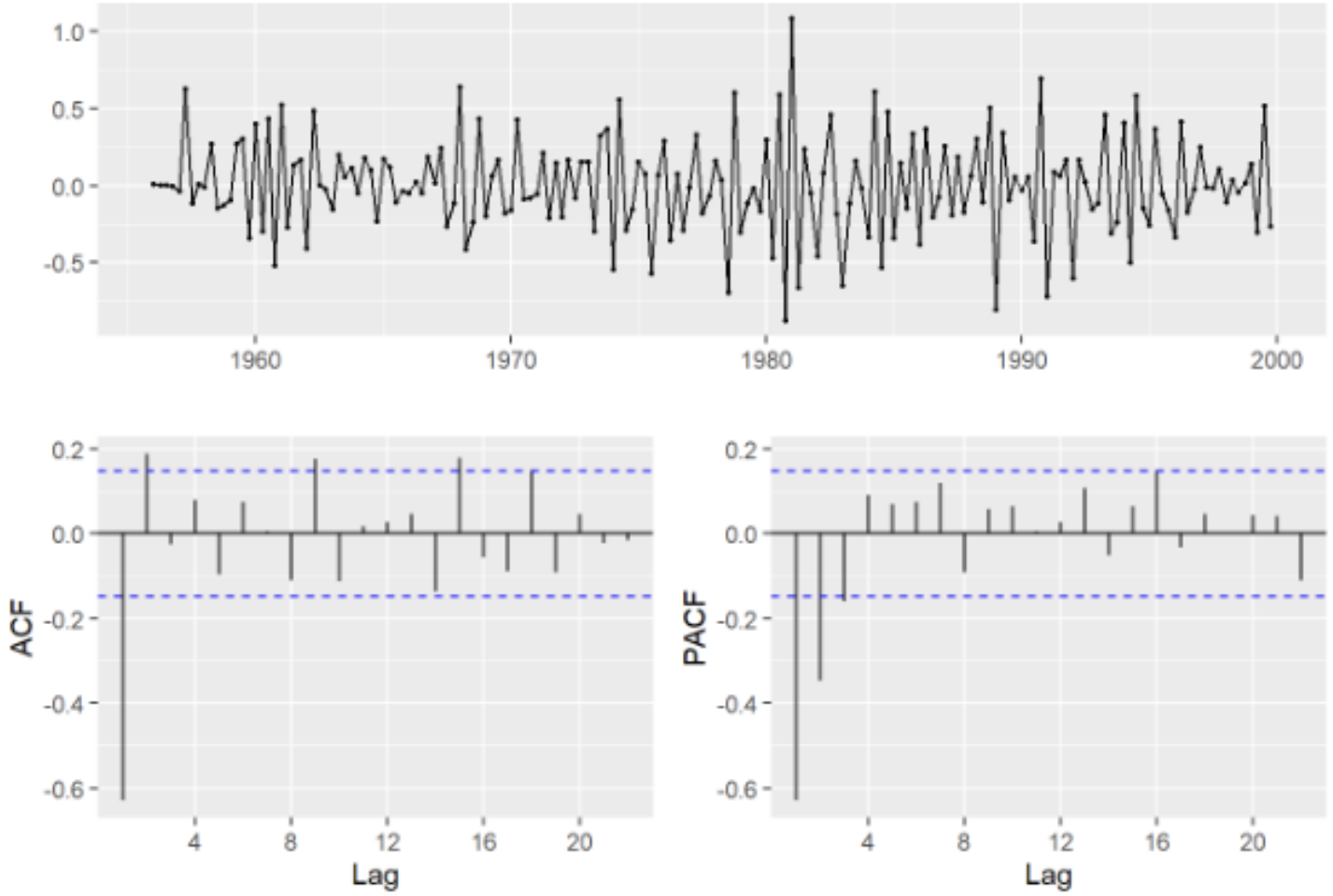

```
ggtsdisplay(residuals(model1))
```



ARIMA: Fitting a Model

Trying out an alternative model

```
model2 = Arima(y = train,order = c(0,1,0),seasonal = c(0,1,1),lambda = BoxCox.lambda(train))
ggtsdisplay(residuals(model2))
```

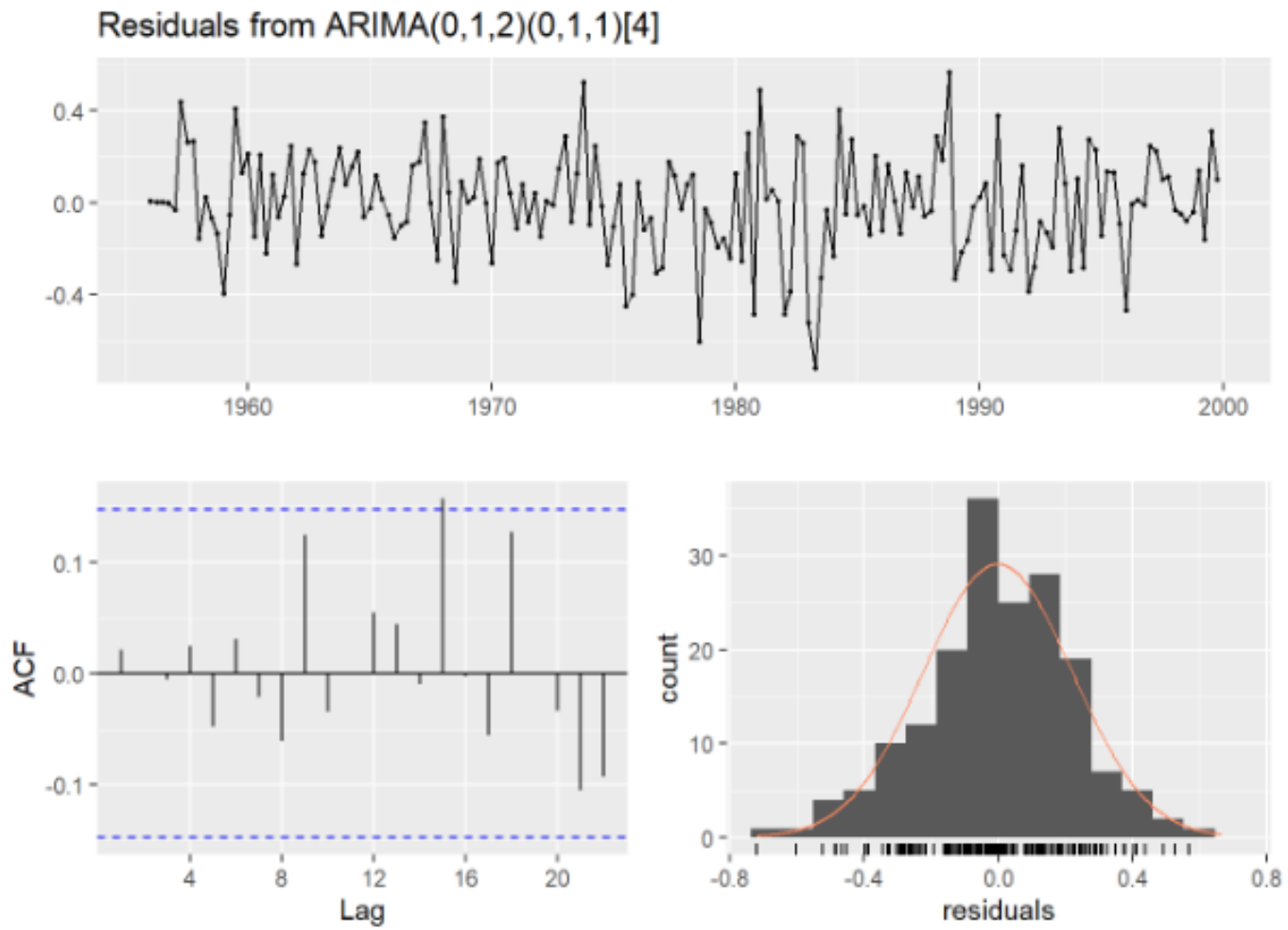


ARIMA: Fitting a Model – Check Residuals of Model1

COLUMBIA UNIVERSITY
School of Professional Studies

Check residuals by plotting. If they do not look like white noise, try a modified model

```
checkresiduals(model1)
```



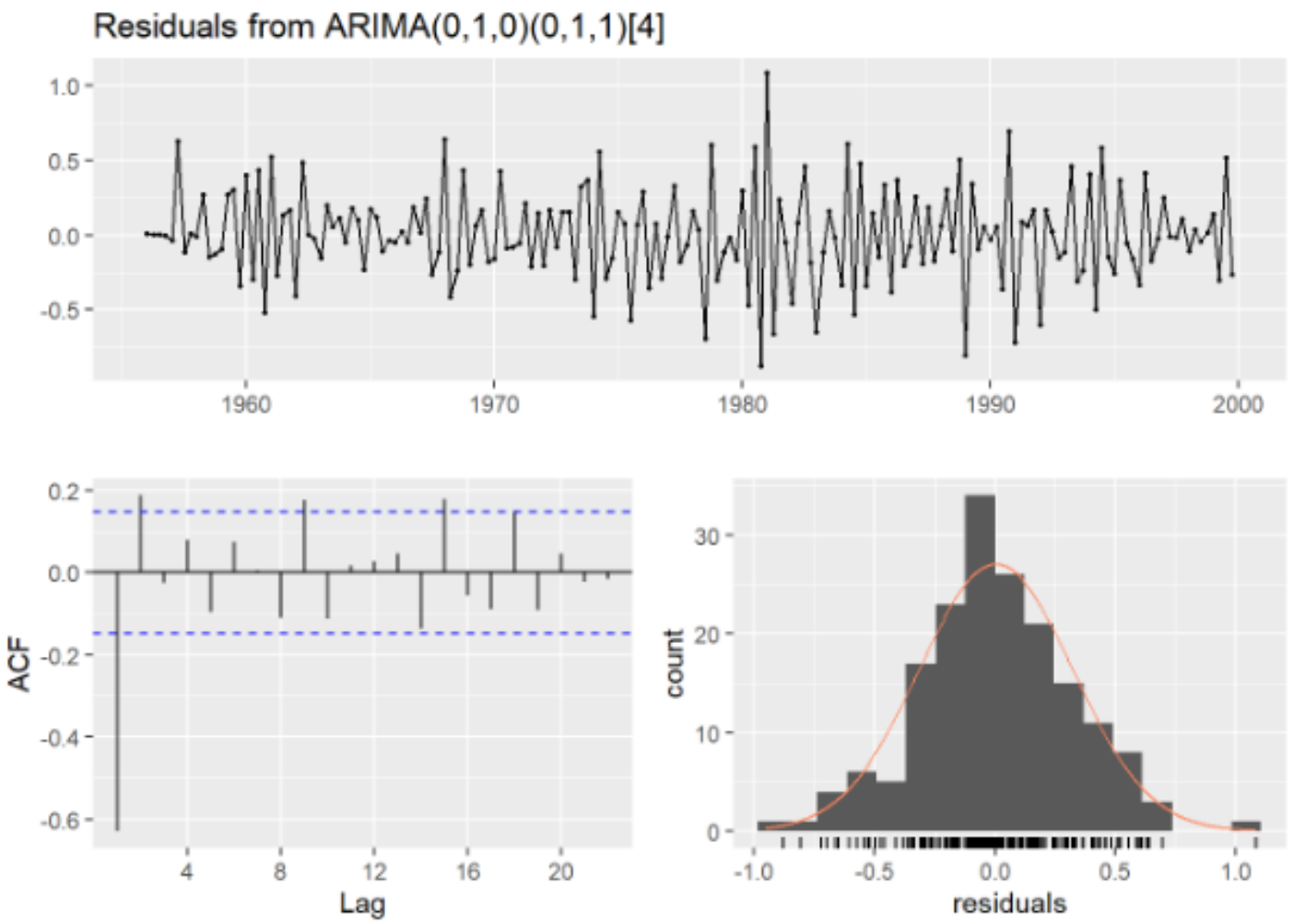
H_0 : the autocorrelations up to lag k are all 0
H_A : the autocorrelations of one or more lags differ from 0.

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,2)(0,1,1)[4]
## Q* = 1.5798, df = 5, p-value = 0.9037
##
## Model df: 3.   Total lags used: 8
```

Residuals resemble white noise

ARIMA: Fitting a Model – Check Residuals of Model2


```
checkresiduals(model2)
```



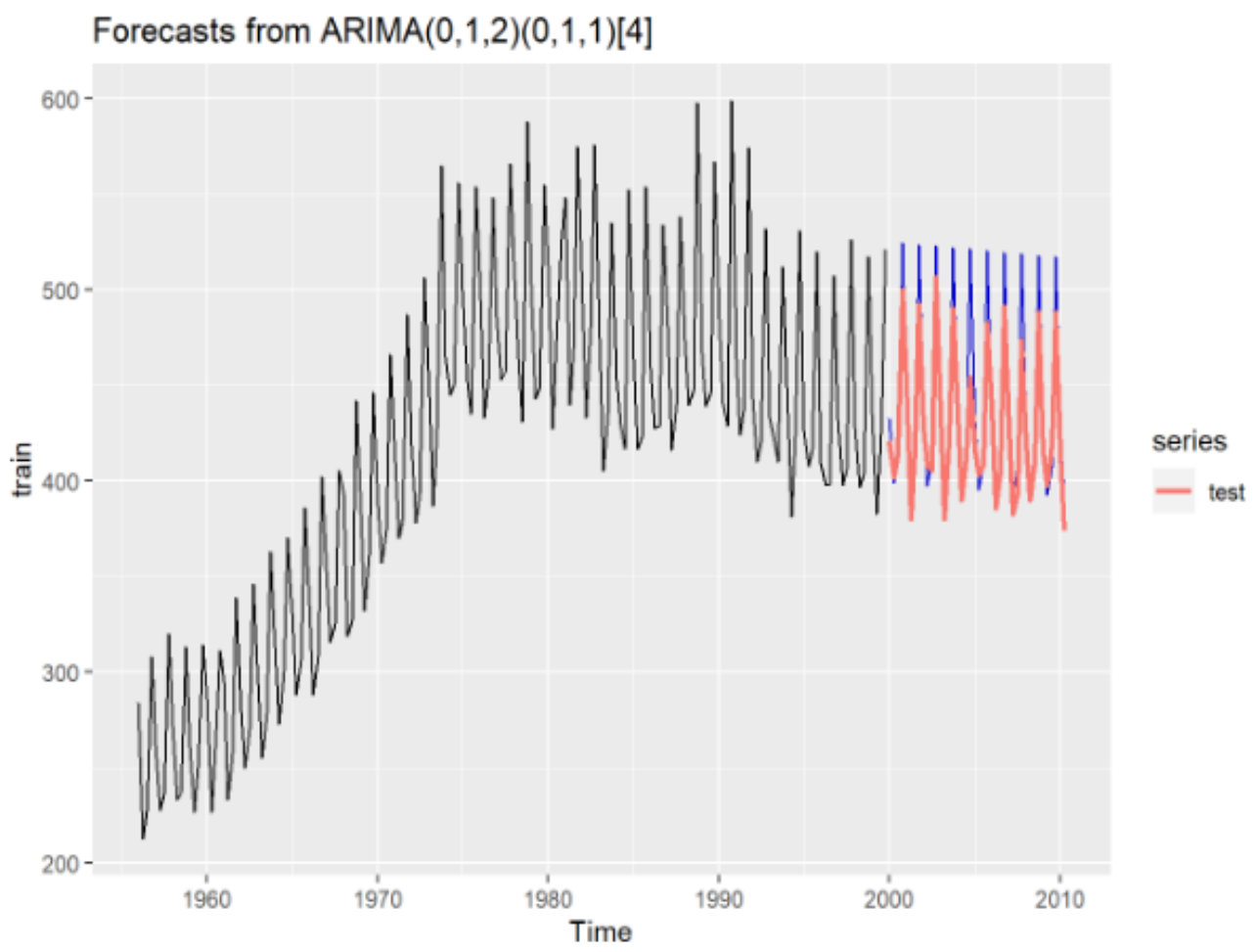

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,0)(0,1,1)[4]
## Q* = 83.58, df = 7, p-value = 2.554e-15
##
## Model df: 1.   Total lags used: 8
```

ARIMA: Forecast

COLUMBIA UNIVERSITY
School of Professional Studies

Once the residuals look like white noise, calculate forecasts

```
autoplot(forecast(model1,h = 42),PI=F)+
  autolayer(test,size=1)
```



- Use model 1 to construct a forecast for Australian beer over the next 42 months of the test sample.

```
model1_forecast = forecast(model1,h=42)
```

```
accuracy(f = model1_forecast,x = ausbeer)
```

```
##                      ME     RMSE      MAE         MPE     MAPE      MASE
## Training set  -0.4443474 14.79078 11.11090 -0.06714964 2.659077 0.6956951
## Test set     -10.3114618 19.88872 14.49978 -2.26976852 3.261576 0.9078855
##                    ACF1 Theil's U
## Training set -0.004102863        NA
## Test set     -0.269479034 0.3887556
```

ARIMA – Automatic Model Selection

ARIMA - Automatic Model Selection

Use auto.arima to pick the best model based on AICc. Sometimes auto.arima() does not yield an optimal solution as it uses computational shortcuts. By setting stepwise and approximation to False, we will ensure a more extensive search.

As it turns out, the automatic process picked the model we manually configured.

```
model_auto = auto.arima(y = train,d = 1,D = 1,stepwise = F,approximation = F)
model_auto
```

```
## Series: train
## ARIMA(0,1,2)(0,1,1)[4]
##
## Coefficients:
##           ma1     ma2     sma1
##       -0.9758  0.3959  -0.7659
## s.e.   0.0722  0.0762   0.0520
##
## sigma^2 estimated as 244.5:  log likelihood=-713.37
## AIC=1434.74   AICc=1434.98   BIC=1447.3
```

Compare All Models

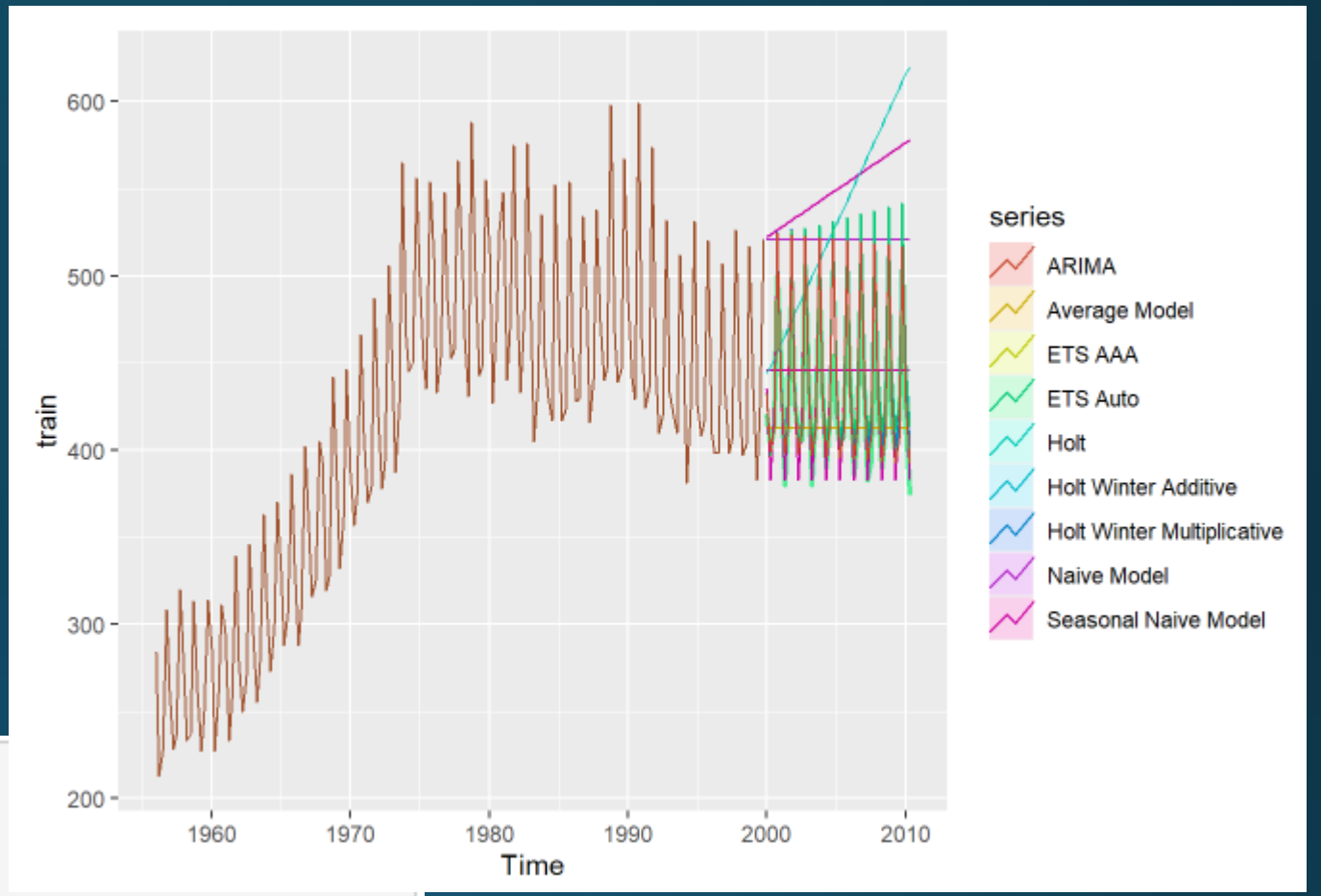
COLUMBIA UNIVERSITY
School of Professional Studies

Comparing Forecasting Models

Let us compare all the models in terms of accuracy metrics on the test sample.

```
rbind(average_model = accuracy(f = average_model,x = ausbeer)[2,],
      naive_model = accuracy(f = naive_model,x = ausbeer)[2,],
      seasonal_naive_model = accuracy(f = seasonal_naive_model,x = ausbeer)[2,],
      drift_model = accuracy(f = drift_model,x = ausbeer)[2,],
      ses_model = accuracy(f = ses_model,x = ausbeer)[2,],
      holt_model = accuracy(f = holt_model,x = ausbeer)[2,],
      holt_damped_model = accuracy(f = holt_damped_model,x = ausbeer)[2,],
      hw_additive_model = accuracy(f = hw_additive,x = ausbeer)[2,],
      hw_multiplicative = accuracy(f = hw_multiplicative,x = ausbeer)[2,],
      ets_aaa = accuracy(ets_aaa_forecast,x = ausbeer)[2,],
      ets_auto = accuracy(ets_auto_forecast,x = ausbeer)[2,],
      arima = accuracy(model1_forecast,x=ausbeer)[2,]
      )
```

```
##                          ME      RMSE       MAE        MPE      MAPE
## average_model         15.25866  40.10097  28.70346   2.876448  6.348359
## naive_model          -93.30952 100.40881  93.30952 -22.690269 22.690269
## seasonal_naive_model -11.54762  21.85286  17.02381  -2.521773  3.863916
## drift_model          -122.42667 129.44954 122.42667 -29.581531 29.581531
## ses_model            -18.09737  41.26471  37.68469  -4.978561  8.973630
## holt_model           -104.37559 124.04137 108.00085 -25.405766 26.131936
## holt_damped_model     -32.18055  49.84832  44.97675  -8.303939 10.897279
## hw_additive_model     -14.05834  22.67754  16.71429  -3.151230  3.775641
## hw_multiplicative     -16.47439  24.38794  18.20496  -3.721500  4.125710
## ets_aaa               -20.38991  28.22830  22.12902  -4.654819  5.060769
## ets_auto              -20.48601  28.61288  22.23155  -4.669352  5.076505
## arima                 -10.31146  19.88872  14.49978  -2.269769  3.261576
##                           MASE         ACF1 Theil's U
## average_model         1.7972318 -0.052981076 0.7832398
## naive_model           5.8424602 -0.052981076 1.8984090
## seasonal_naive_model  1.0659247  0.029098002 0.4278746
## drift_model           7.6655940  0.152038516 2.4589272
## ses_model             2.3595800 -0.052981076 0.7972014
## holt_model            6.7623392  0.611282124 2.3753720
## holt_damped_model     2.8161636 -0.005103579 0.9593471
## hw_additive_model     1.0465444 -0.185710461 0.4430322
## hw_multiplicative     1.1398809 -0.169487536 0.4757030
## ets_aaa               1.3855813  0.015455576 0.5492527
## ets_auto              1.3920011  0.010216303 0.5569420
## arima                 0.9078855 -0.269479034 0.3887556
```




```
autoplot(train, color='sienna')+
  autolayer(test,size=1.05,color='seagreen2')+
  autolayer(average_model,series = 'Average Model',PI=F)+
  autolayer(naive_model,series = 'Naive Model',PI=F)+
  autolayer(seasonal_naive_model,series = 'Seasonal Naive Model',PI=F)+
  autolayer(drift_model,series = 'Seasonal Naive Model',PI=F)+
  autolayer(ses_model,series = 'Seasonal Naive Model',PI=F)+
  autolayer(holt_model,series = 'Holt',PI=F)+
  autolayer(hw_additive,series = 'Holt Winter Additive',PI=F)+
  autolayer(hw_multiplicative,series = 'Holt Winter Multiplicative',PI=F)+
  autolayer(ets_aaa_forecast,series = 'ETS AAA',PI=F)+
  autolayer(ets_auto_forecast,series = 'ETS Auto',PI=F)+
  autolayer(model1_forecast,series = 'ARIMA',PI=F)
```

Summary

- In this module we,
  - Examined, explored, and visualized time-series data
  - Reviewed basic/simple forecasting methods
  - Explored exponential smoothing
  - Examined ARIMA models
  - Constructed forecasts using simple methods, exponential smoothing, and ARIMA.