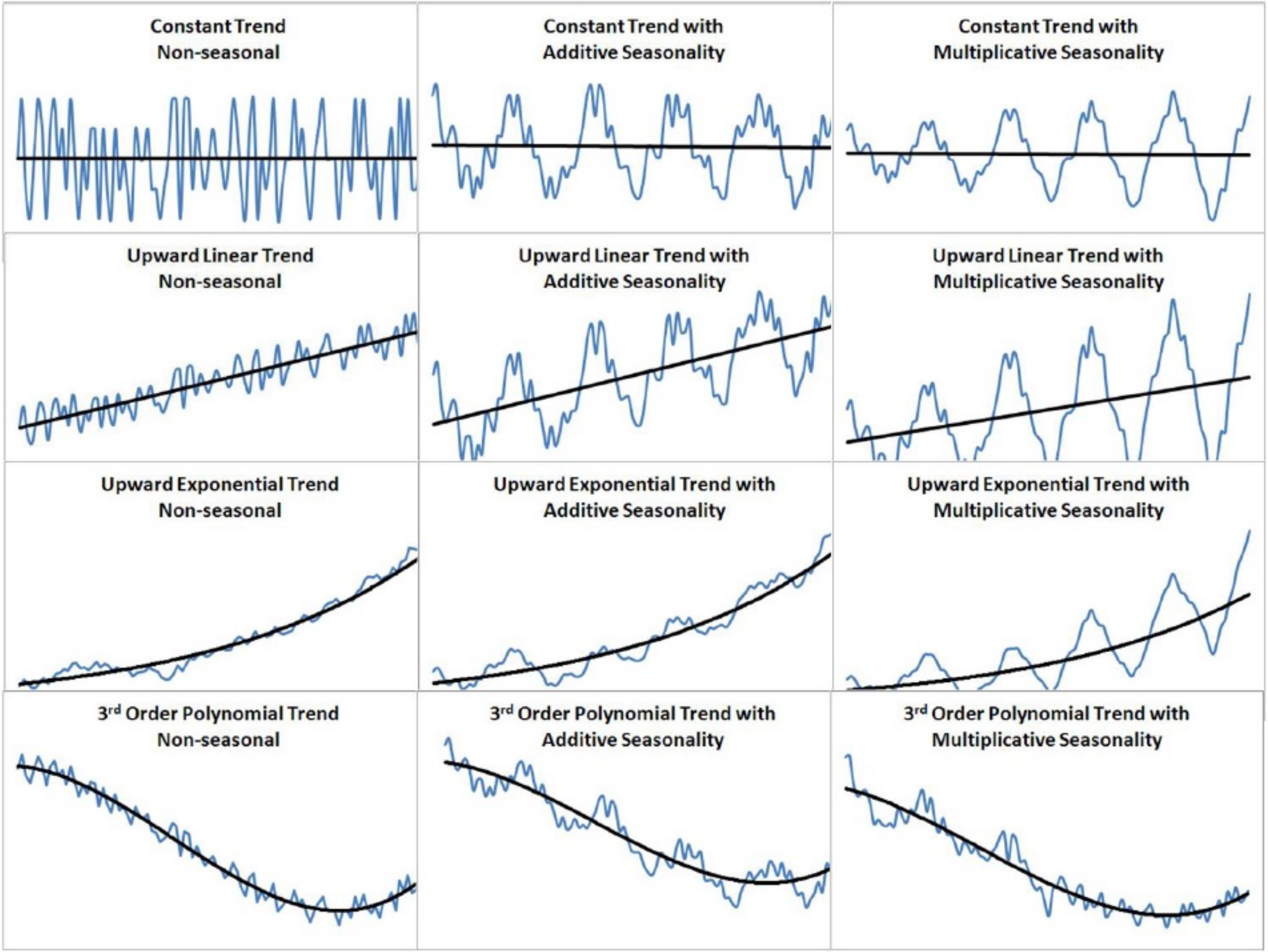
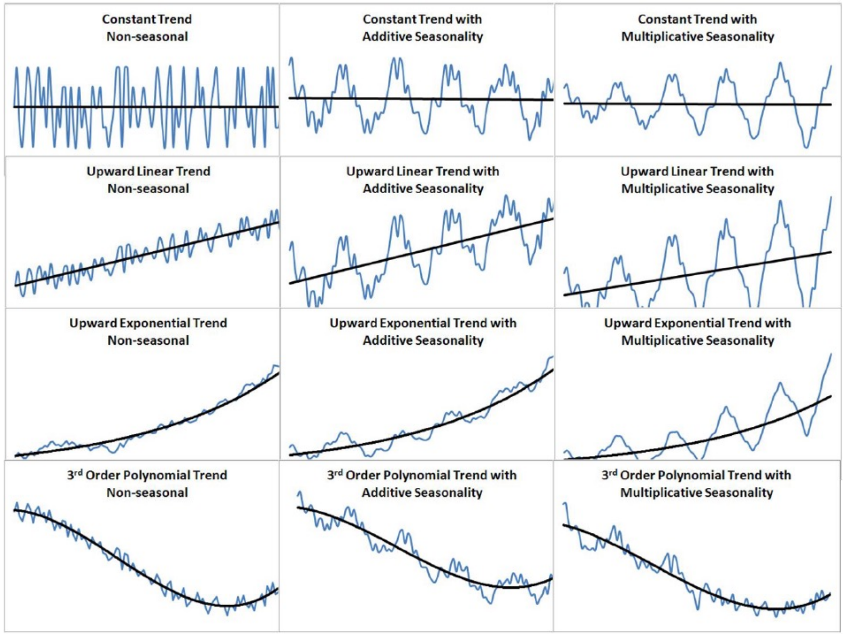
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Business Analytics using Forecasting

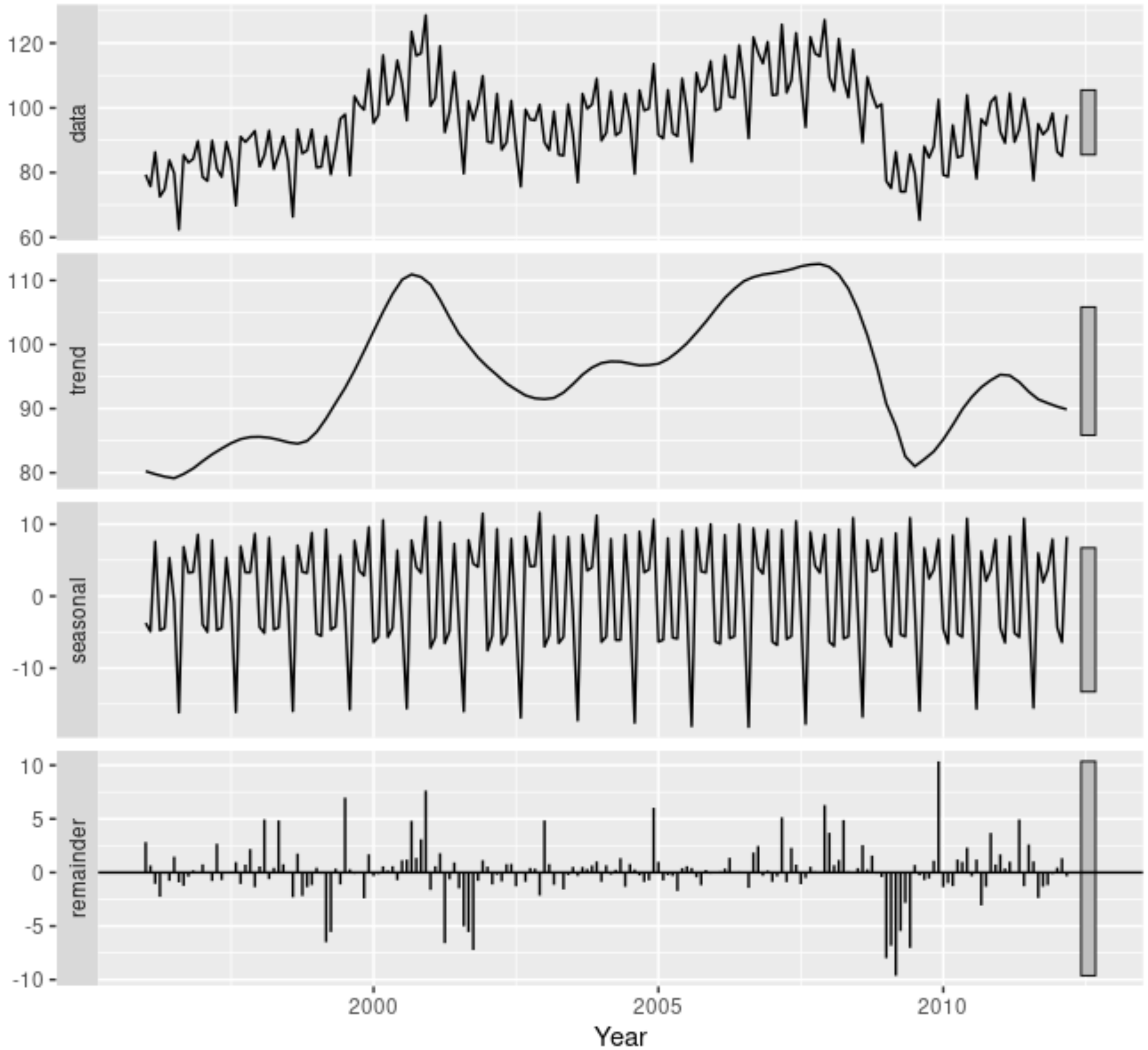
BU7143 & BU7144

Dr. Nicholas P. Danks
Business Analytics
Email address


Constant Trend
Non-seasonal
Constant Trend with
Additive Seasonality
Constant Trend with
Multiplicative Seasonality
Upward Linear Trend
Non-seasonal
Upward Linear Trend with
Additive Seasonality
Upward Linear Trend with
Multiplicative Seasonality
Upward Exponential Trend
Non-seasonal
Upward Exponential Trend with
Additive Seasonality
Upward Exponential Trend with
Multiplicative Seasonality
3rd Order Polynomial Trend
Non-seasonal
3rd Order Polynomial Trend with
Additive Seasonality
3rd Order Polynomial Trend with
Multiplicative Seasonality



Constant Trend
Non-seasonal
Constant Trend with
Additive Seasonality
Constant Trend with
Multiplicative Seasonality
Upward Linear Trend
Non-seasonal
Upward Linear Trend with
Additive Seasonality
Upward Linear Trend with
Multiplicative Seasonality
Upward Exponential Trend
Non-seasonal
Upward Exponential Trend with
Additive Seasonality
Upward Exponential Trend with
Multiplicative Seasonality
3rd Order Polynomial Trend
Non-seasonal
3rd Order Polynomial Trend with
Additive Seasonality
3rd Order Polynomial Trend with
Multiplicative Seasonality



data
120
100
80
60
trend
110
100
90
80
seasonal
10
0
-10
remainder
10
5
0
-5
-10
2000
2005
2010
Year

Exponential Trend

Appropriate model when increase/decrease in series over time is multiplicative

e.g. t_1 is x% more than t_0, t_2 is x% more than t_1…

Replace Y with log(Y) then fit linear regression

$log(Y_i) = B_0 + B_1 t + e$

Exponential trend - forecast errors

Note that performance measures in standard linear regression software are not in original units

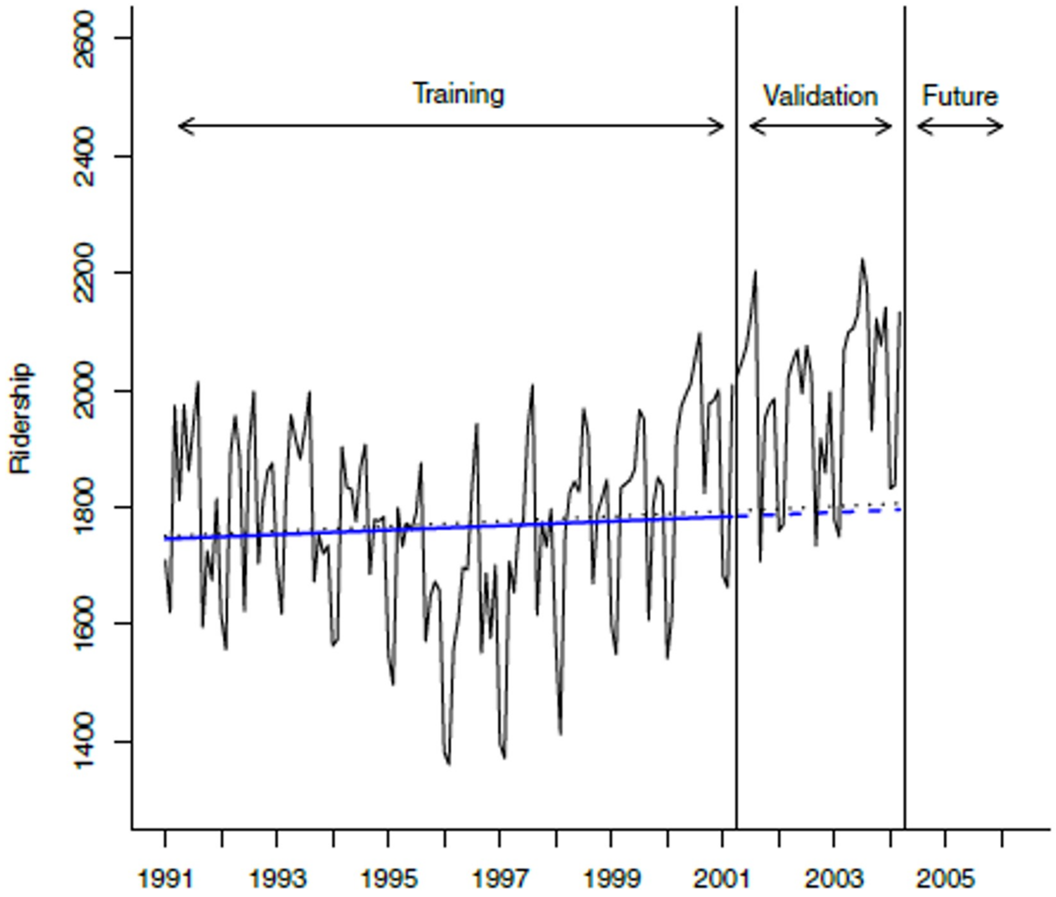
Model forecasts will be in the form log(*Y*)

Return to original units by taking exponent of model forecasts

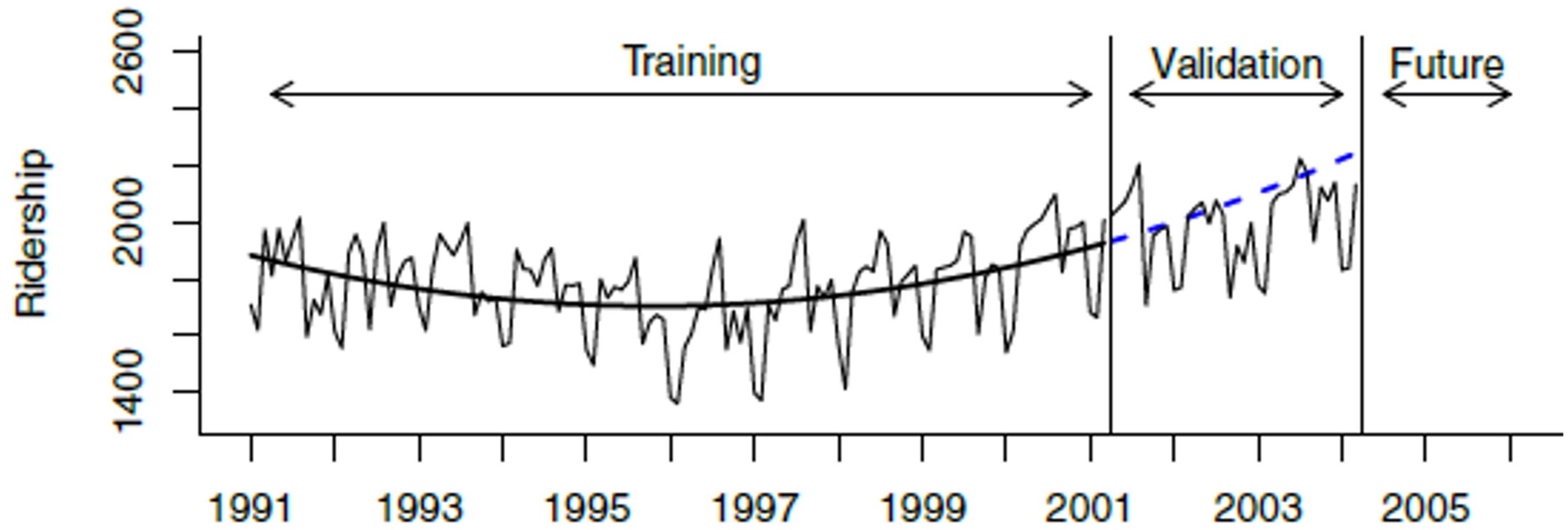
Calculate standard deviation of these forecast errors to get RMSE

```
# fit exponential trend using tslm() with argument
# lambda = 0

train.lm.expo.trend <- tslm(train.ts ~ trend, lambda = 0)
train.lm.expo.trend.pred <- forecast(train.lm.expo.trend,
     h = nValid, level = 0)
```

Exponential trend (dotted line) very similar to linear trend (solid line)



```
# fit quadratic trend using function I(), which treats an
# object "as is".

train.lm.poly.trend <- tslm(train.ts ~ trend + I(trend^2))
summary(train.lm.poly.trend)
train.lm.poly.trend.pred <- forecast(train.lm.poly.trend,
     h = nValid, level = 0)
```



Better job capturing the trend, though it over forecasts in validation period.
Next: we'll try capturing seasonality.

Polynomial Trend

Add additional predictors as appropriate

For example, for quadratic relationship add a t^2 predictor

Fit linear regression using both t and t^2

Handling Seasonality

- Seasonality is any recurring cyclical pattern of consistently higher or lower values (daily, weekly, monthly, quarterly, etc.)
- Handle in regression by adding categorical variable for season, e.g.

Month	Ridership	Season
Jan-91	1709	Jan
Feb-91	1621	Feb
Mar-91	1973	March
Apr-91	1812	April

11, not 12, to avoid multicollinearity

```
# include season as a predictor in tslm(). Here it creates 11
# dummies, one for each month except for first season, January
train.lm.season <- tslm(train.ts ~ season)
summary(train.lm.season)
```

Final model, Amtrak data

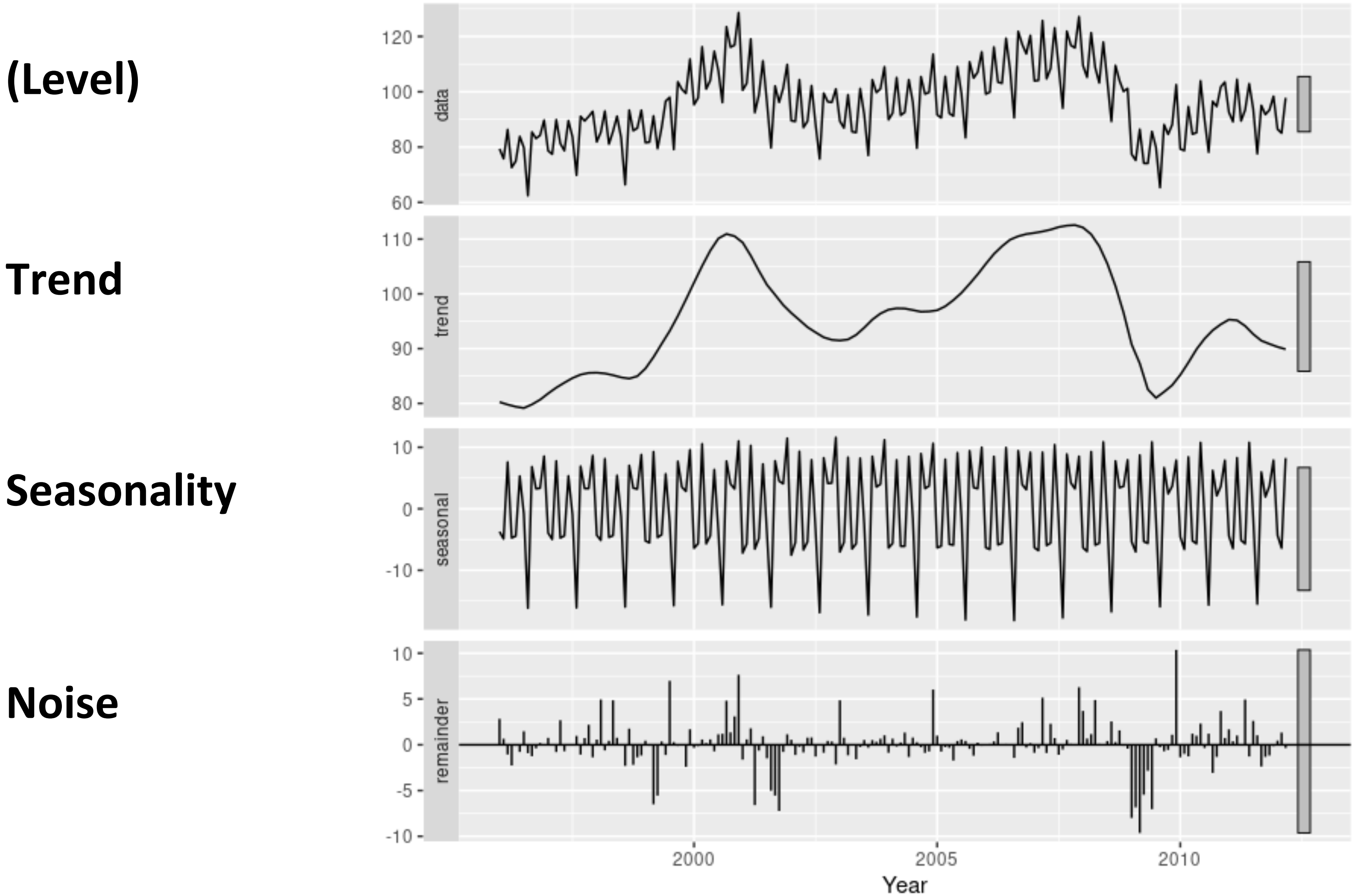
Incorporates trend and seasonality

13 predictors

- 11 monthly dummies
- t
- t^2

```
train.lm.trend.season <- tslm(train.ts ~ trend +
     I(trend^2) + season)
```

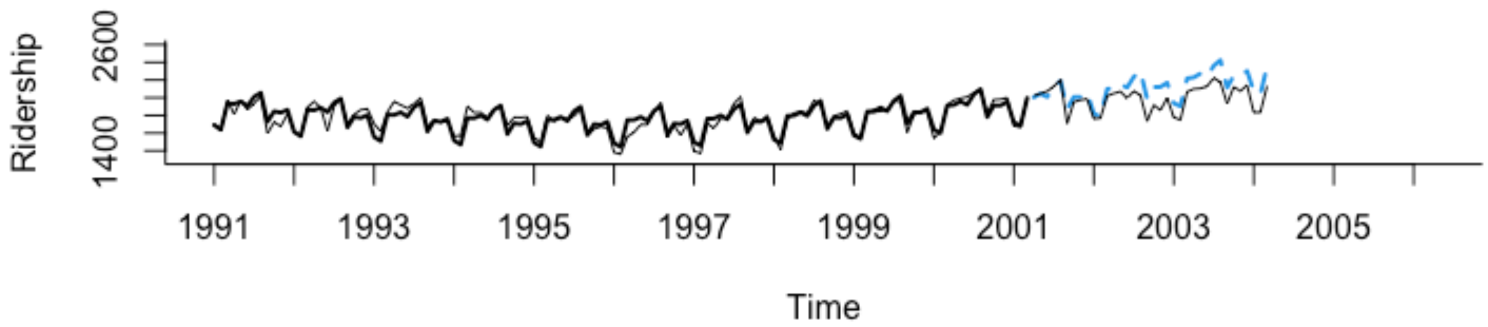
Time Series Components

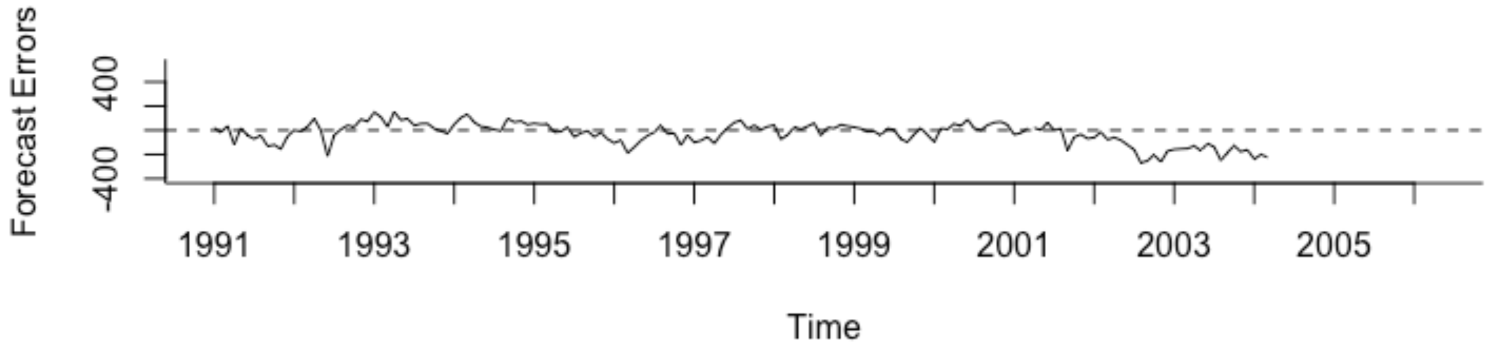


Output of full model

```
> train.lm.trend.season <- tslm(train.ts ~ trend + I(trend^2) + season)
> summary(train.lm.trend.season)

Call:
tslm(formula = train.ts ~ trend + I(trend^2) + season)

Residuals:
     Min       1Q   Median       3Q      Max 
-213.775  -39.363    9.711   42.422  152.187 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.697e+03  2.768e+01  61.318  < 2e-16 ***
trend       -7.156e+00  7.293e-01  -9.812  < 2e-16 ***
I(trend^2)   6.074e-02  5.698e-03  10.660  < 2e-16 ***
season2     -4.325e+01  3.024e+01  -1.430  0.15556    
season3      2.600e+02  3.024e+01   8.598 6.60e-14 ***
season4      2.606e+02  3.102e+01   8.401 1.83e-13 ***
season5      2.938e+02  3.102e+01   9.471 6.89e-16 ***
season6      2.490e+02  3.102e+01   8.026 1.26e-12 ***
season7      3.606e+02  3.102e+01  11.626  < 2e-16 ***
season8      4.117e+02  3.102e+01  13.270  < 2e-16 ***
season9      9.032e+01  3.102e+01   2.911  0.00437 ** 
season10     2.146e+02  3.102e+01   6.917 3.29e-10 ***
season11     2.057e+02  3.103e+01   6.629 1.34e-09 ***
season12     2.429e+02  3.103e+01   7.829 3.44e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 70.92 on 109 degrees of freedom
Multiple R-squared:  0.8246,	Adjusted R-squared:  0.8037 
F-statistic: 39.42 on 13 and 109 DF,  p-value: < 2.2e-16
```

Full model Performance






```
> train.lm.trend.season.pred <- forecast(train.lm.trend.season, h = nValid, level = 0)
> accuracy(train.lm.trend.season.pred, valid.ts)
                    ME      RMSE       MAE        MPE     MAPE      MASE      ACF1 Theil's U
Training set  3.693205e-15  66.76143  51.95091 -0.1525653 3.015509 0.6297693 0.6040588        NA
Test set     -1.261654e+02 153.25066 131.72503 -6.4314945 6.698700 1.5968226 0.7069291 0.8960679
>
> train.lm.season.pred <- forecast(train.lm.season, h = nValid, level = 0)
> accuracy(train.lm.season.pred, valid.ts)
                   ME      RMSE       MAE        MPE      MAPE      MASE      ACF1 Theil's U
Training set 3.685399e-15  96.34038  75.13099 -0.3099246  4.326881 0.9107674 0.7856939        NA
Test set     2.179267e+02 229.65092 217.92668 10.8646179 10.864618 2.6417928 0.6346963  1.330938
```

Autocorrelation

Unlike cross-sectional data, time-series values are typically correlated with nearby values ("autocorrelation")

Ordinary regression does not account for this

Computing autocorrelation

Create "lagged" series

Copy of the original series, offset by one or more timer periods

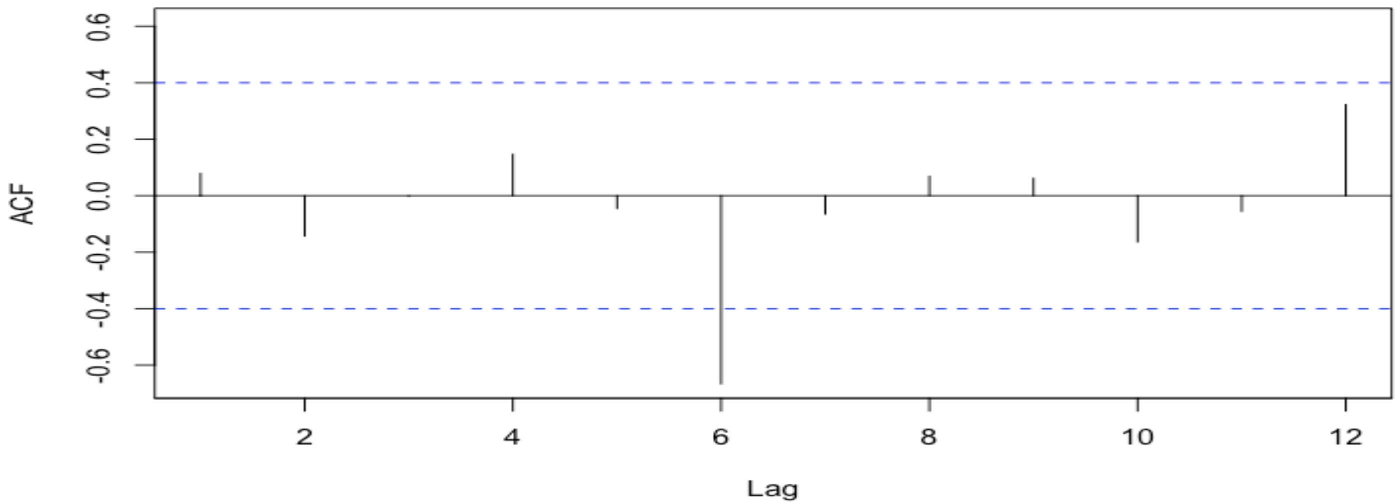
Compute correlation between original series and lagged series

- Lag-1, lag-2, etc.

Amtrak – original series and Lag-1, Lag-2

TABLE 16.1 FIRST 24 MONTHS OF AMTRAK RIDERSHIP SERIES

Month	Ridership	Lag-1 Series	Lag-2 Series
Jan-91	1709		
Feb-91	1621	1709	
Mar-91	1973	1621	1709
Apr-91	1812	1973	1621
May-91	1975	1812	1973
Jun-91	1862	1975	1812
Jul-91	1940	1862	1975
Aug-91	2013	1940	1862
Sep-91	1596	2013	1940
Oct-91	1725	1596	2013
Nov-91	1676	1725	1596
Dec-91	1814	1676	1725
Jan-92	1615	1814	1676

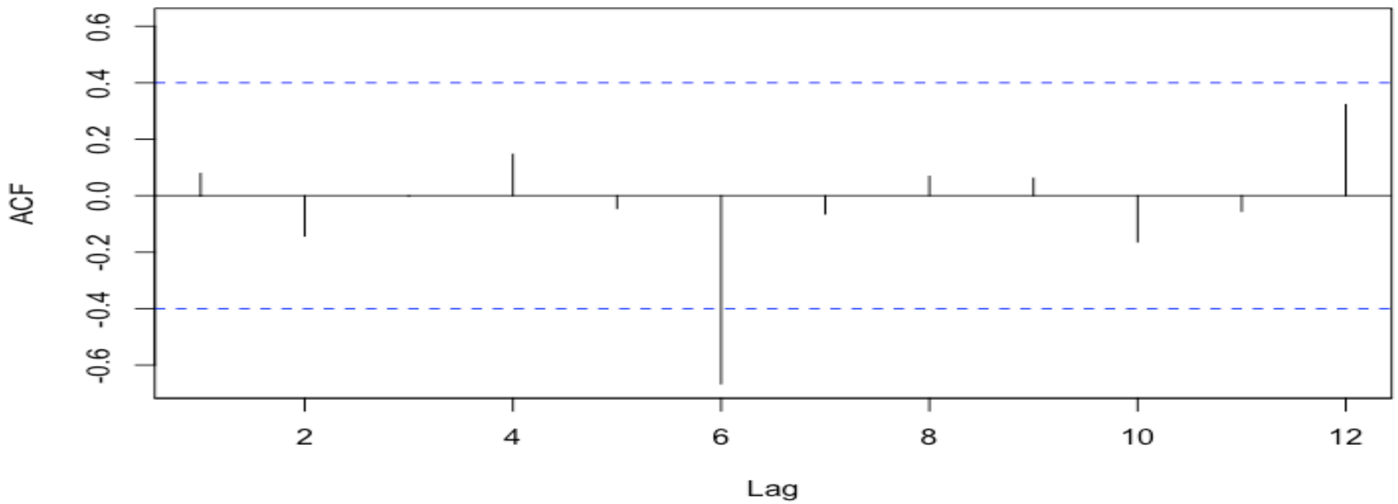


Autocorrelation

Positive autocorrelation at lag-1 = stickiness

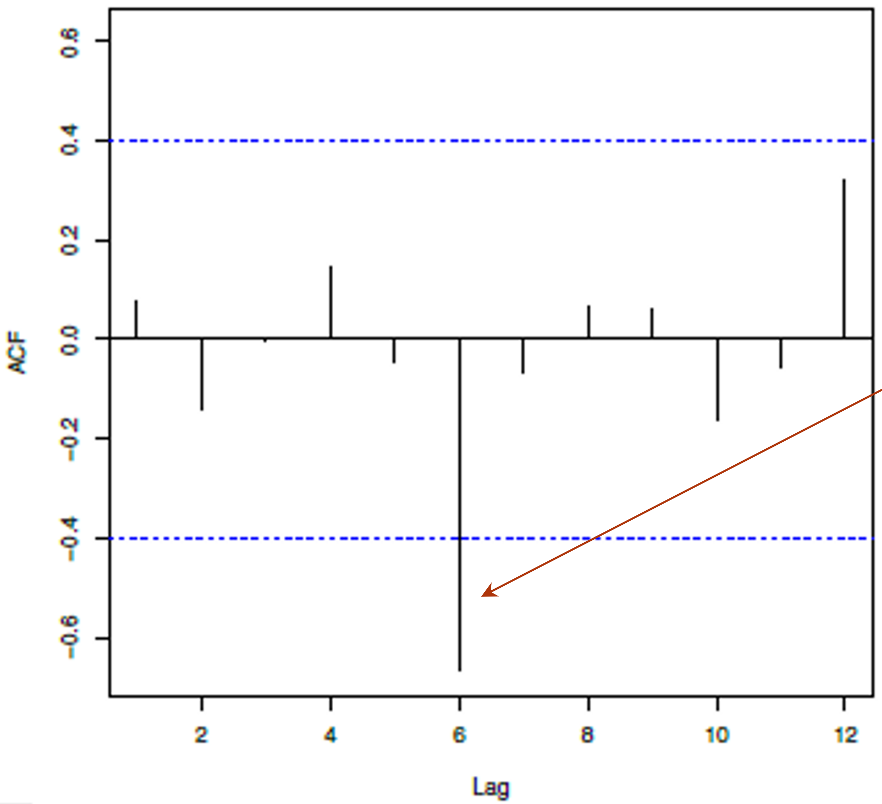
Strong autocorrelation (positive or negative) at a lag > 1 indicates seasonal (cyclical) pattern

Autocorrelation in residuals indicates the model has not fully captured the seasonality in the data



Compute & display autocorrelation for different lags, over 24 months:

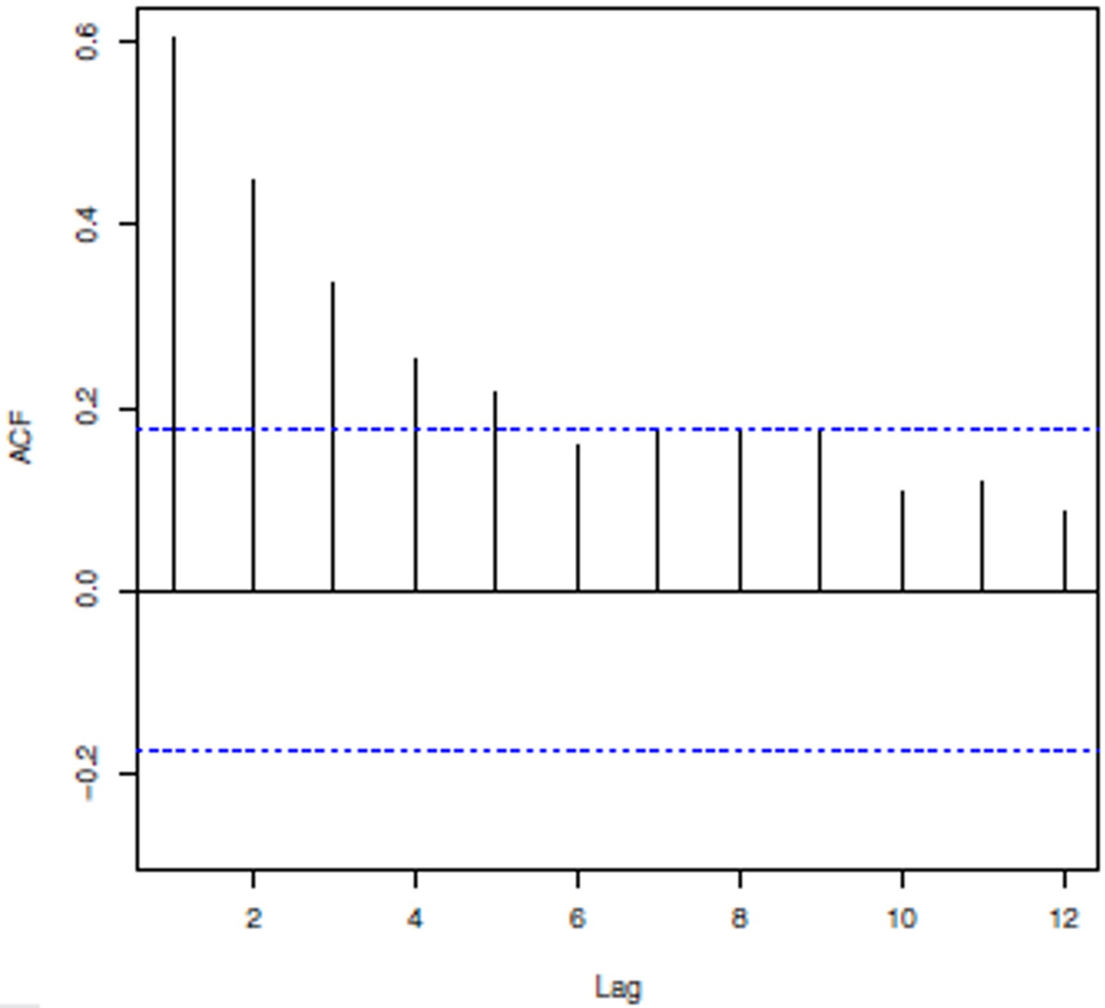
```
ridership.24.ts <- window(train.ts, start = c(1991, 1),
     end = c(1991, 24))
Acf(ridership.24.ts, lag.max = 12, main = "")
```



Strong negative correlation at 6 months shows seasonal pattern (high summer traffic, low winter)

The dotted lines are confidence bounds for judging statistical significance

It is useful to examine autocorrelation for the residuals:



Strong autocorrelation from lag 1 on, but lag 6 no longer dominates.

Note: If you have correlation at lag 1, it will naturally propagate to lag 2, 3, etc., tapering off

Incorporating autocorrelation into models

Use a forecasting method to forecast *k*-steps ahead

Fit AR (autoregressive) model to residuals

Incorporate residual forecasts

$Improved\ F_{t+k} = F_{t+k} + E_{t+k}$

Choose order of the AR model

If autocorrelation exists at Lag-1, a Lag-1 model should be sufficient to capture lags at other periods as well

$E_t = B_0 + B_1 E_{t-1} + e$

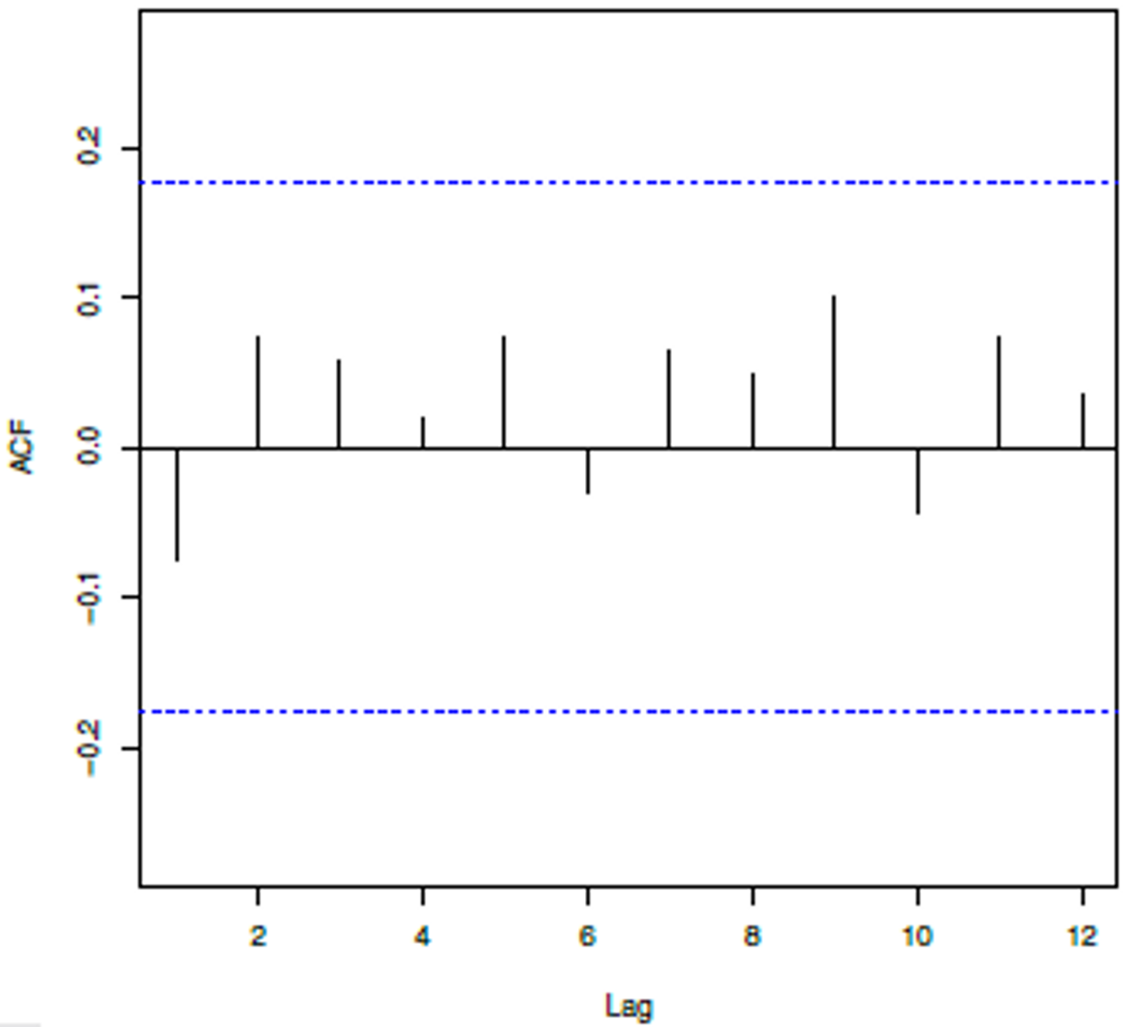
Where E_t is residual (forecast error) at time t

Adding ARIMA lag 1 - AR(1) - to earlier model...

```
# fit linear regression with quadratic trend and seasonality
# to Ridership

train.lm.trend.season <- tslm(train.ts ~ trend + I(trend^2) +
     season)

# fit AR(1) model to training residuals
# use Arima() in the forecast package to fit an ARIMA model
# (that includes AR models); order = c(1,0,0) gives an AR(1).

train.res.arima <- Arima(train.lm.trend.season$residuals,
     order = c(1,0,0))
valid.res.arima.pred <- forecast(train.res.arima, h = 1)
```

Plot autocorrelation of “residuals of residuals”

Autocorrelation is mostly gone – AR(1) has adequately captured the autocorrelation in the data:



Random walks

- Before forecasting, consider “is the time series predictable?”
- Or is it a random walk?
- Do a statistical hypothesis test that slope = 1 in an AR(1) model (i.e. that the forecast for a period is the most recently-observed value)
- If hypothesis cannot be rejected, series is statistically equivalent to a random walk (i.e. we have not shown that it is predictable).

Summary – Regression Based Forecasting

- Can use linear regression for exponential models (use logs) and polynomials (exponentiation)
- For seasonality, use categorical variable (make dummies)
- Incorporate autocorrelation by modeling it, then using those error forecasts in the main model

Part II
Smoothing

Smoothing is “data driven”

- Regression methods assume underlying unchanging structure (linear, exponential, polynomial)
- Smoothing derives forecasts based directly on the data alone (e.g. averaging), with no mathematical structural assumptions
- Suitable where the components (trend, seasonality) change over time

Simple moving average (MA)

- **Set window width "*w*" – take average of the *w* values.**
- **For centered moving average, window is centered around forecast point**

  For w=5, the forecast for t_3 averages the values t_1 … t_5

  Not useful for future forecasts
- **For future forecasts, use "trailing average" = the value being forecast is at the end of the window**

Choosing window width

- **Goal is to suppress seasonality and noise**
- **Typically choose window width = season length**

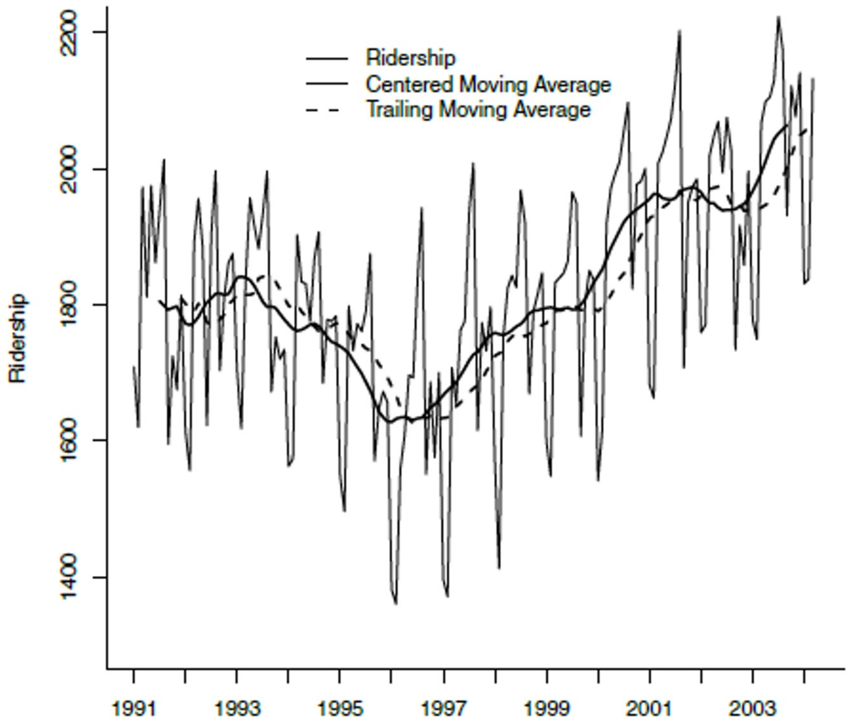
Moving Average Functions

```
order = 12 (window)
```

```
ma.centered <- ma(ridership.ts,
                  order = 12)

ma.trailing <- rollmean(ridership.ts,
                        k = 12,
                        align = "right")
```

```
k = 12 (window)
align = right (trailing)
```

Amtrak Ridership:
Moving Average Smoothing
Window W = 12

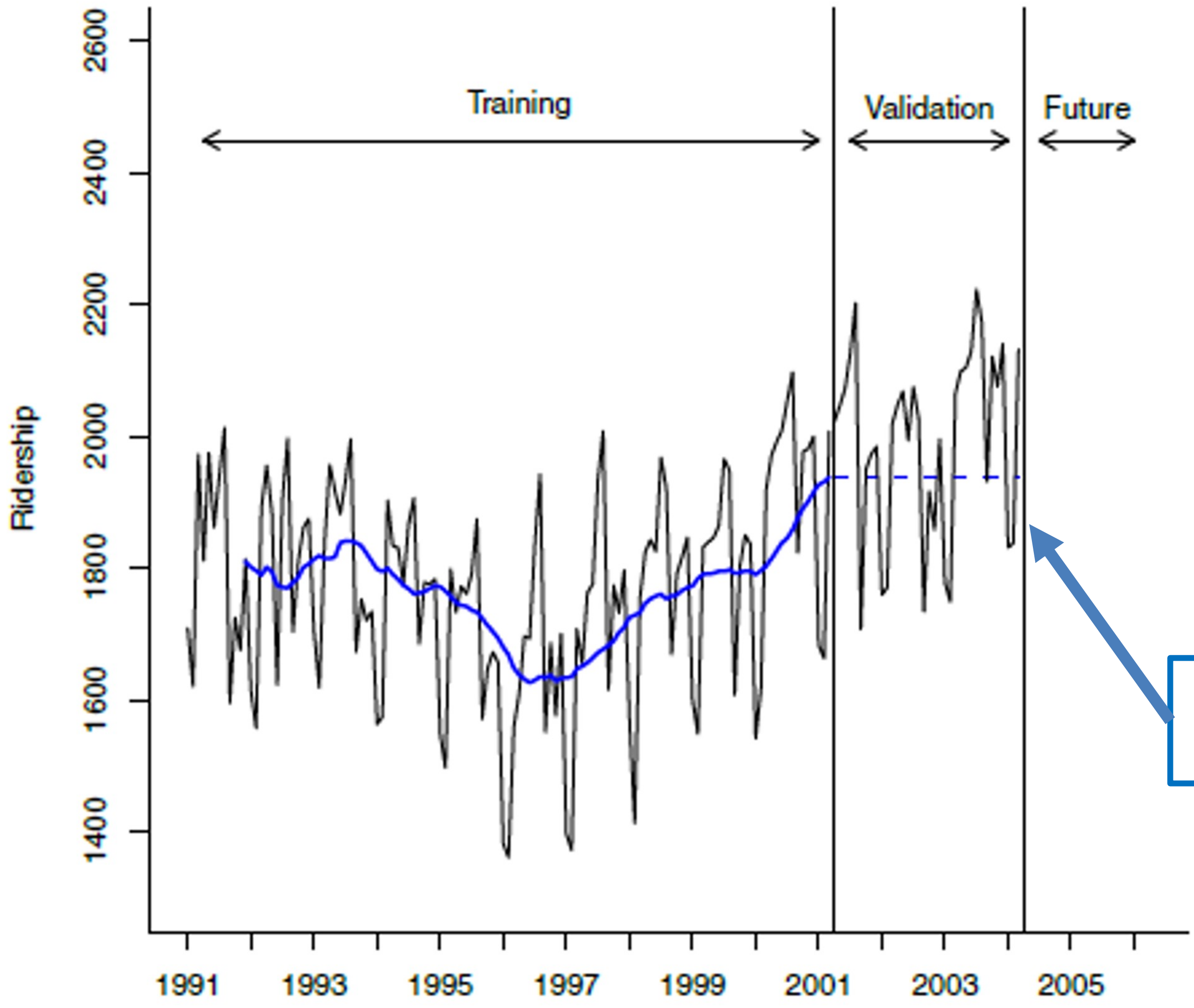


Plotting Code

```
# generate a plot
plot(ridership.ts,
     ylim = c(1300, 2200),
     ylab = "Ridership",
     xlab = "Time",
     bty = "l",
     xaxt = "n",
     xlim = c(1991,2004.25),
     main = "")
axis(1,
     at = seq(1991, 2004.25, 1),
     labels = format(seq(1991, 2004.25, 1)))
lines(ma.centered, lwd = 2)
lines(ma.trailing, lwd = 2, lty = 2)
legend(1994,2200,
       c("Ridership",
       "Centered Moving Average",
       "Trailing Moving Average"),
       lty=c(1,1,2),
       lwd=c(1,2,2),
       bty = "n")
```

MA forecast, and checking it in the validation period:

```
# partition the data
nValid <- 36
nTrain <- length(ridership.ts) - nValid
train.ts <- window(ridership.ts,
            start = c(1991, 1),
            end = c(1991, nTrain))
valid.ts <- window(ridership.ts,
            start = c(1991, nTrain + 1),
            end = c(1991, nTrain + nValid))

# moving average on training
ma.trailing <- rollmean(train.ts,
               k = 12,
               align = "right")

# obtain the last moving average in the training period
last.ma <- tail(ma.trailing,
                1)

# create forecast based on last MA
ma.trailing.pred <- ts(rep(last.ma, nValid),
                       start = c(1991, nTrain + 1),
                       end = c(1991, nTrain + nValid),
                       freq = 12)
```



Shortcomings

Suppresses seasonality, but does not forecast seasonal component

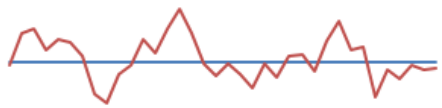
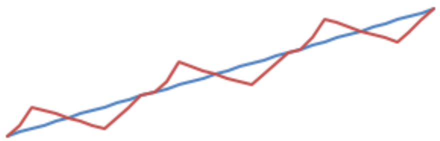
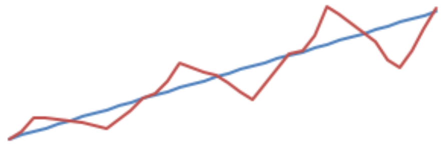
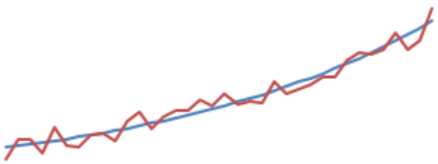
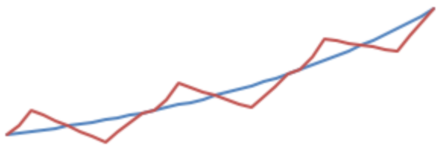
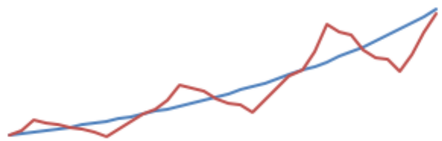
Lags behind trends

Validation forecast is the last moving average from training period

Thus, simple Moving Average useful only for series that lack trend and seasonality

Solutions:

- Use regression model to de-trend and de-seasonalize
- Use Moving Average to forecast the de-trended and de-seasonalized series
- Add trend and seasonality back to the forecast

Profile	No Seasonality	Additive Seasonality	Multiplicative Seasonality
No Trend			
Additive Trend			
Multiplicative Trend			

Simple exponential smoothing

Like MA, except use weighted average of all past values, instead of simple average in a window

Forecast at time t+1:

$$F_{t+1} = \alpha Y_t + \alpha(1-\alpha)Y_{t-1} + \alpha(1-\alpha)^2 Y_{t-2} + \ldots$$

Equivalent to:

$$F_{t+1} = F_t + \alpha E_t$$

Smoothing parameter α

Simple exponential smoother corrects based on error

- If last period forecast was too high, next period is adjusted down
- If last period forecast was too low, next period is adjusted up

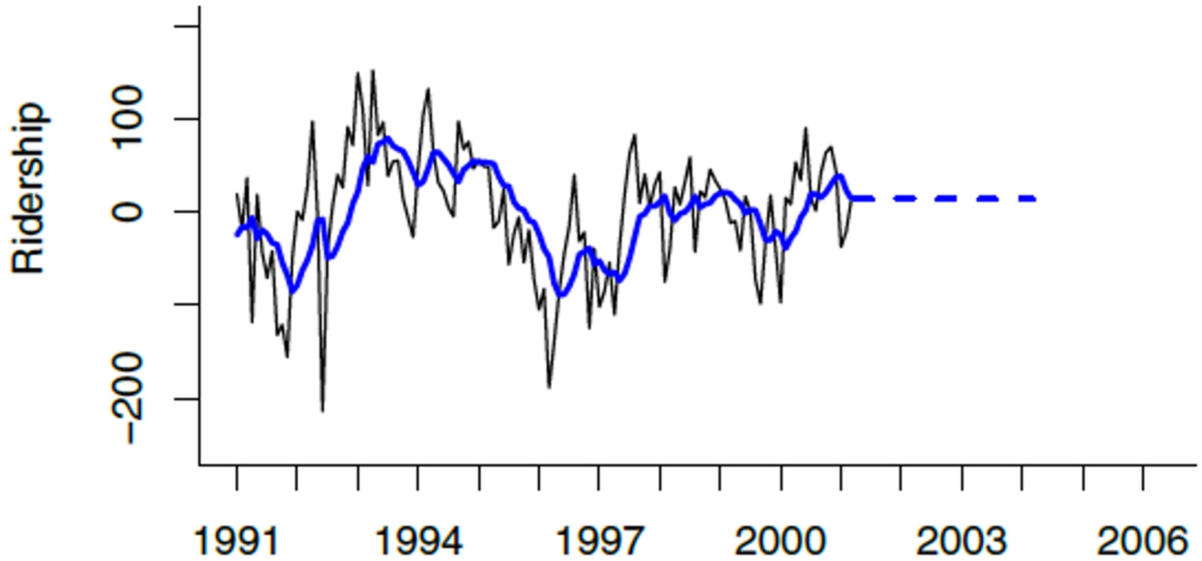
Amount of correction depends on value of α

- Value close to 1 > fast learning, close to 0 > low learning

Output for simple exponential smoothing applied to residuals from regression model:

```
# get residuals
residuals.ts <- train.lm.trend.season$residuals

# run simple exponential smoothing
# use ets() with model = "ANN" (additive error (A),
# no trend (N), no seasonality (N))
# and alpha = 0.2 to fit simple exponential smoothing.

ses <- ets(residuals.ts, model = "ANN", alpha = 0.2)
ses.pred <- forecast(ses, h = nValid, level = 0)
```

y-axis is residual from regression model



Moving average and simple exponential smoothing can be used only when there is no trend or seasonality. When those features are present:

- One solution is to remove those components via regression
- Another is to use advanced exponential smoothing, which can capture trend and seasonality
- Double-exponential smoothing used for series with a trend

Double exponential smoothing

Incorporates trend

K-step ahead forecast is derived from the level (L) and trend (T) estimates at time t

$$F_{t+k} = L_t + kT_t$$

where

$$L_t = \alpha Y_t + (1-\alpha)(L_{t-1} + T_{t-1})$$
$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$$

Holt Winters exponential smoothing

- Extension of double exponential smoothing
- Incorporate both trend and seasonality

Holt Winters forecast for time t+k

Adds seasonality to double exponential

For M seasons (e.g. M=7 for weekly), forecast is

$$F_{t+k} = (L_t + kT_t)S_{t+k-M}$$

Where L = level, T = trend, S = season

Updating L, T and S

Like eq. for double exponential, except for seasonal adjustment term

$$L_t = \frac{\alpha Y_t}{S_{t-M}} + (1-\alpha)(L_{t-1} + T_{t-1}),$$

Like double exponential equation

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1},$$

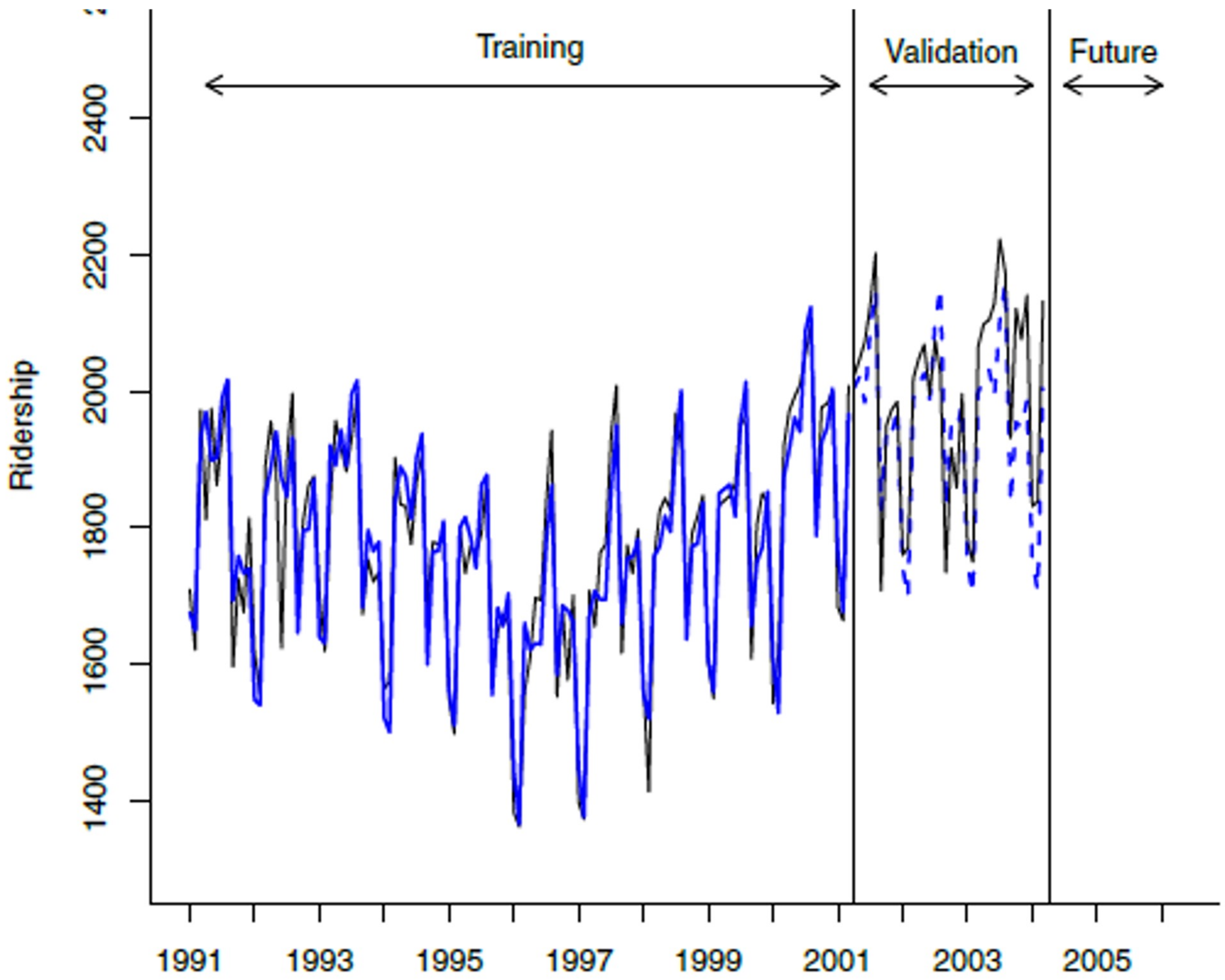
Equation to update seasonal index

$$S_t = \frac{\gamma Y_t}{L_t} + (1-\gamma)S_{t-M}.$$

Holt Winters predictions:

```
# run Holt-Winters exponential smoothing
# use ets() with option model = "MAA" to fit Holt-
# Winter's exponential smoothing
# with multiplicative error, additive trend, and
# additive seasonality.

hwin <- ets(train.ts, model = "MAA")

# create predictions

hwin.pred <- forecast(hwin, h = nValid, level = 0)
```

Holt-Winters Predictions



Summary

- Smoothing methods rely on local data, not mathematical structure
- Simple smoothing does not account for trend and seasonality, but can be combined with model-based forecasts to improve the forecast
- Holt-Winters smoothing incorporates seasonality and trend