ARIMA models

R. Gevorgyan

May 8, 2019

General description

In this tutorial, we walk through an example of examining time series for demand at a bike-sharing service, fitting an ARIMA model, and creating a basic forecast.

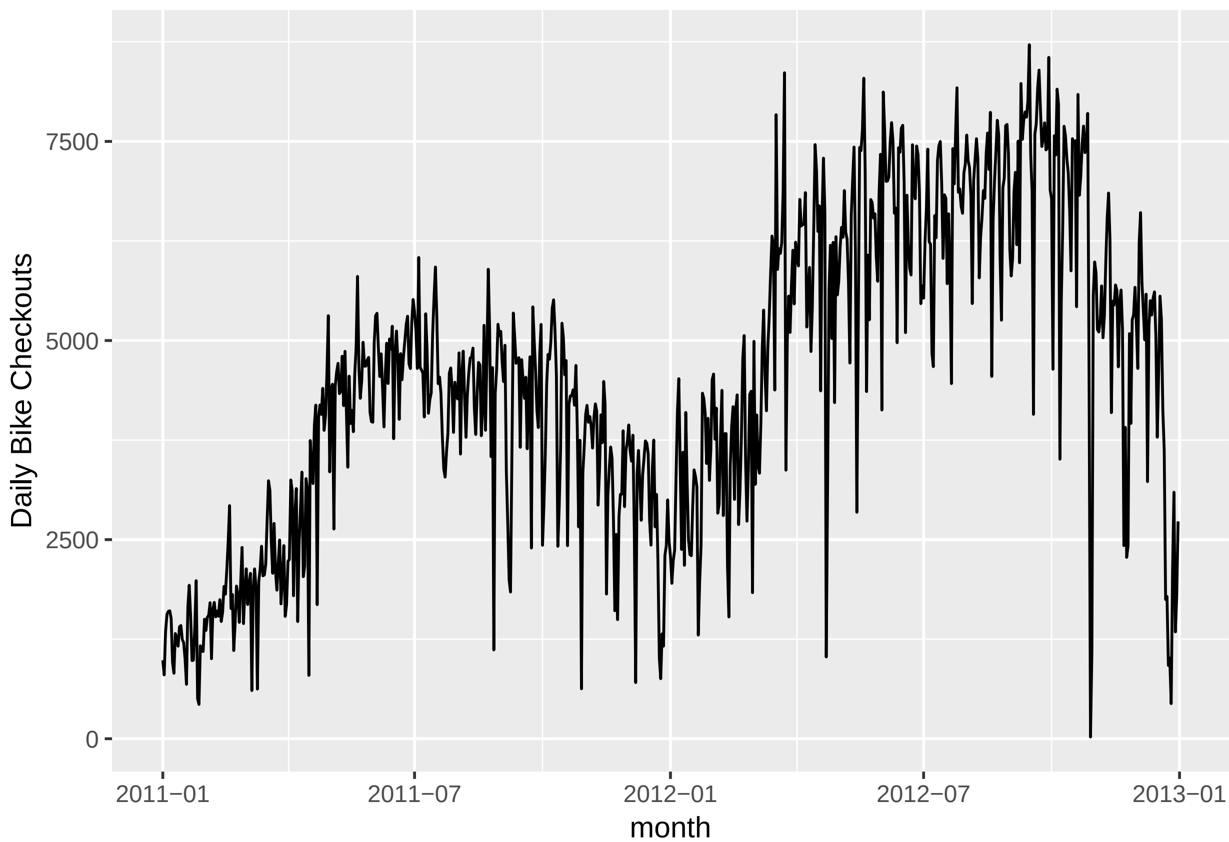
```
#Load R Packages
library('ggplot2')
library('forecast')
library('tseries')
#getwd()
#setwd("C:/Users/dell/Desktop")
daily_data = read.csv('day.csv', header=TRUE, stringsAsFactors=FALSE)
head(daily_data)
```

```
##   instant     dteday season yr mnth holiday weekday workingday weathersit
## 1       1 2011-01-01      1  0    1       0       6          0          2
## 2       2 2011-01-02      1  0    1       0       0          0          2
## 3       3 2011-01-03      1  0    1       0       1          1          1
## 4       4 2011-01-04      1  0    1       0       2          1          1
## 5       5 2011-01-05      1  0    1       0       3          1          1
## 6       6 2011-01-06      1  0    1       0       4          1          1
##       temp    atemp      hum windspeed casual registered  cnt
## 1 0.344167 0.363625 0.805833 0.1604460    331        654  985
## 2 0.363478 0.353739 0.696087 0.2485390    131        670  801
## 3 0.196364 0.189405 0.437273 0.2483090    120       1229 1349
## 4 0.200000 0.212122 0.590435 0.1602960    108       1454 1562
## 5 0.226957 0.229270 0.436957 0.1869000     82       1518 1600
## 6 0.204348 0.233209 0.518261 0.0895652     88       1518 1606
```

Examine Data

Lower usage of bicycles occurs in the winter months and higher checkout numbers are observed in the summer months:

```
daily_data$Date = as.Date(daily_data$dteday)
ggplot(daily_data, aes(Date, cnt)) + geom_line() + scale_x_date('month')  + ylab("Daily Bike Checkouts")
            xlab("")
```

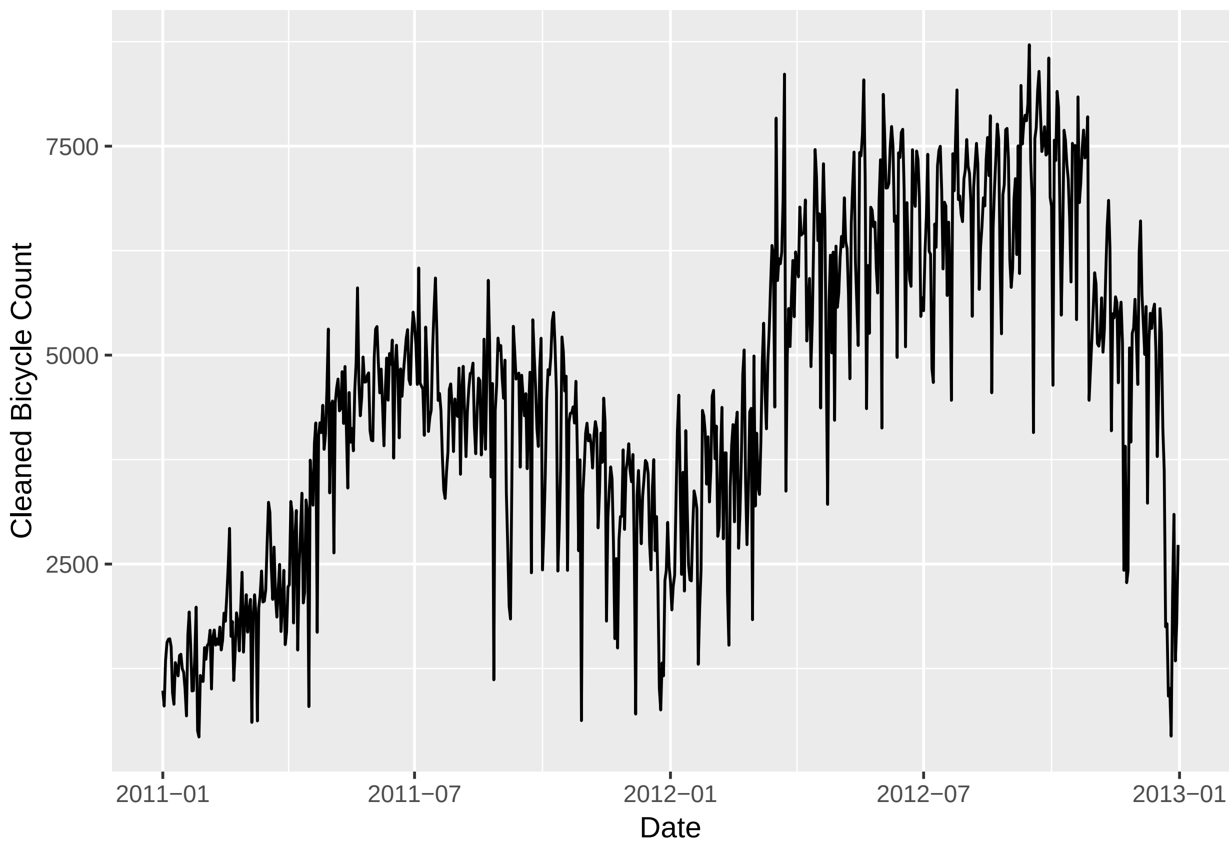


Outliers

R provides a convenient method for removing time series outliers: tsclean() as part of its forecast package. tsclean() identifies and replaces outliers using series smoothing and decomposition.

```
count_ts = ts(daily_data[, c('cnt')])

daily_data$clean_cnt = tsclean(count_ts)

ggplot() +
  geom_line(data = daily_data, aes(x = Date, y = clean_cnt)) + ylab('Cleaned Bicycle Count')
```

```
## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.
```

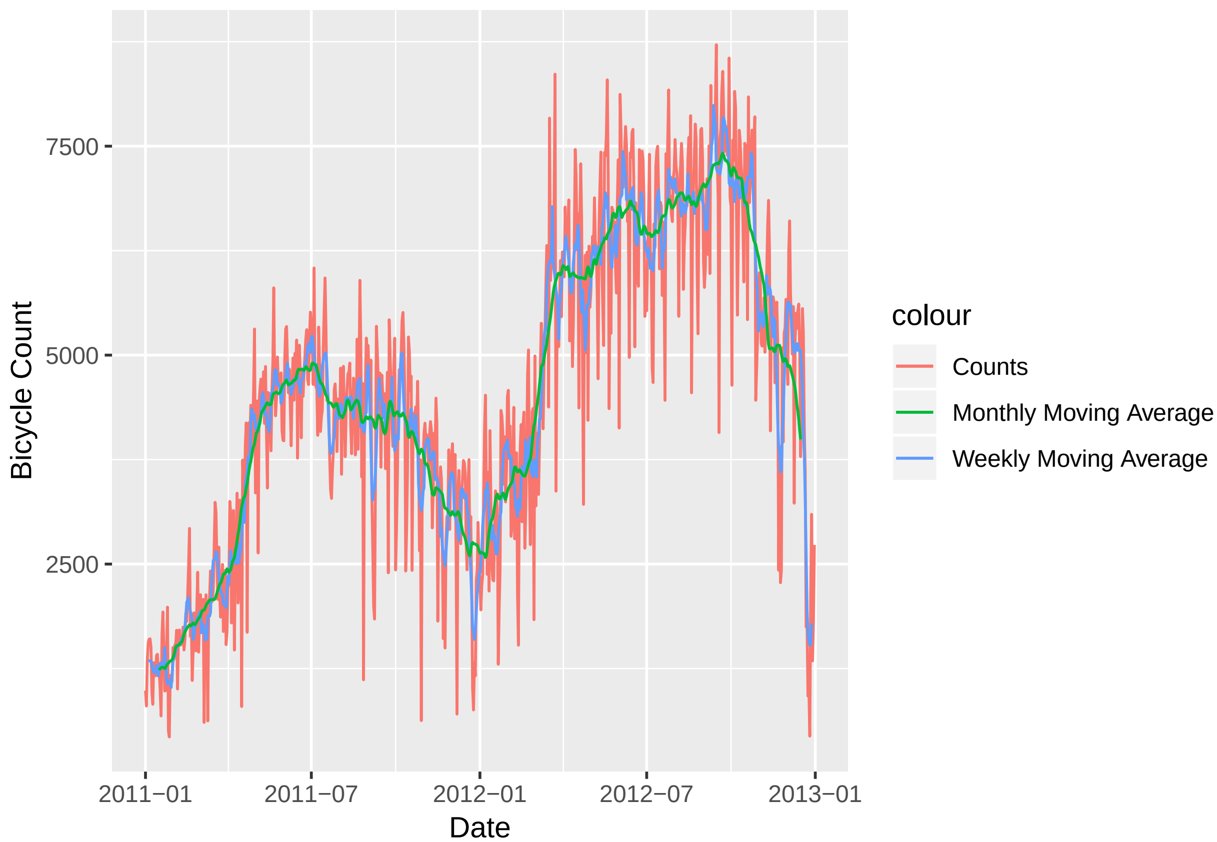


Smoothing

We can take weekly or monthly moving average, smoothing the series into something more stable and therefore predictable:

```
daily_data$cnt_ma = ma(daily_data$clean_cnt, order=7)
# using the clean count with no outliers
daily_data$cnt_ma30 = ma(daily_data$clean_cnt, order=30)

ggplot() +
  geom_line(data = daily_data, aes(x = Date, y = clean_cnt, colour = "Counts")) +
  geom_line(data = daily_data, aes(x = Date, y = cnt_ma,   colour = "Weekly Moving Average"))  +
  geom_line(data = daily_data, aes(x = Date, y = cnt_ma30, colour = "Monthly Moving Average"))  +
  ylab('Bicycle Count')
```

```
## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.

## Warning: Removed 6 rows containing missing values (geom_path).

## Warning: Removed 30 rows containing missing values (geom_path).
```



Decomposition

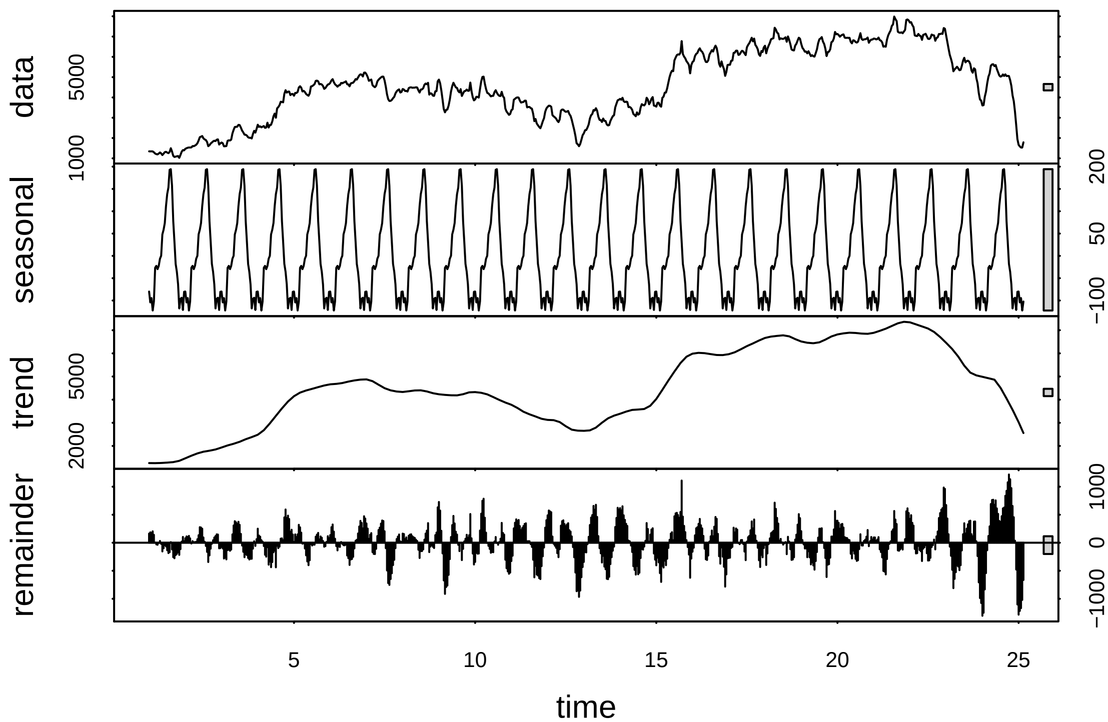
Formally, if Y is the number of bikes rented, we can decompose the series in two ways: by using either an additive or multiplicative model,

$$Y = S_t + T_t + E_t$$
$$Y = S_t * T_t * E_t$$

where St is the seasonal component, T is trend and cycle, and E is the remaining error.

An additive model is usually more appropriate when the seasonal or trend component is not proportional to the level of the series, as we can just overlay (i.e. add) components together to reconstruct the series. On the other hand, if the seasonality component changes with the level or trend of the series, a simple "overlay," or addition of components, won't be sufficient to reconstruct the series. In that case, a multiplicative model might be more appropriate.

```
count_ma = ts(na.omit(daily_data$cnt_ma), frequency=30)
decomp = stl(count_ma, s.window="periodic")
deseasonal_cnt <- seasadj(decomp)
plot(decomp)
```



Stationarity

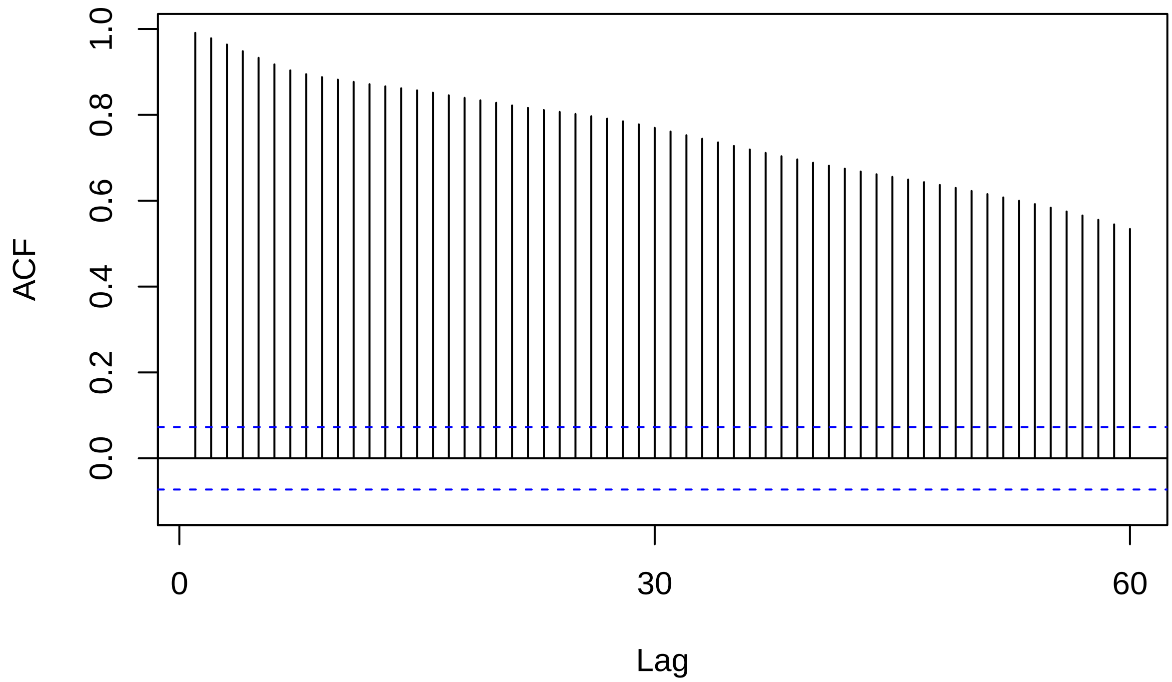
```r
adf.test(count_ma, alternative = "stationary")
```

```
## Warning in adf.test(count_ma, alternative = "stationary"): p-value greater
## than printed p-value
```

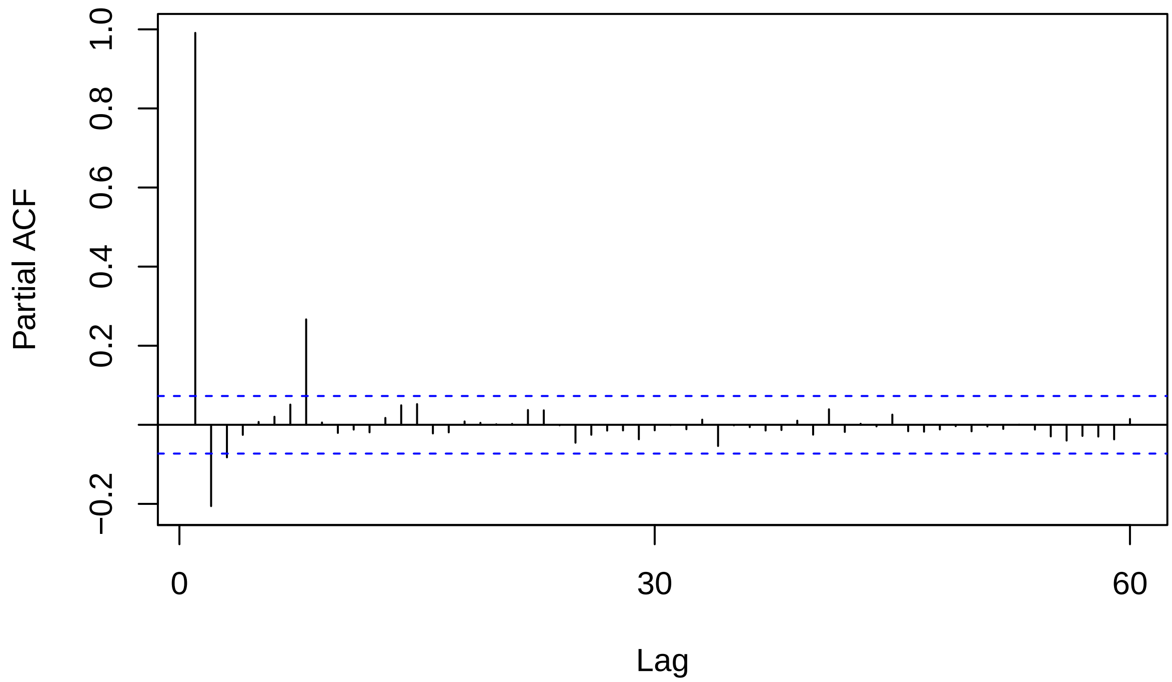
```
##
##  Augmented Dickey-Fuller Test
##
## data:  count_ma
## Dickey-Fuller = -0.2557, Lag order = 8, p-value = 0.99
## alternative hypothesis: stationary
```

Autocorrelations and Choosing Model Order

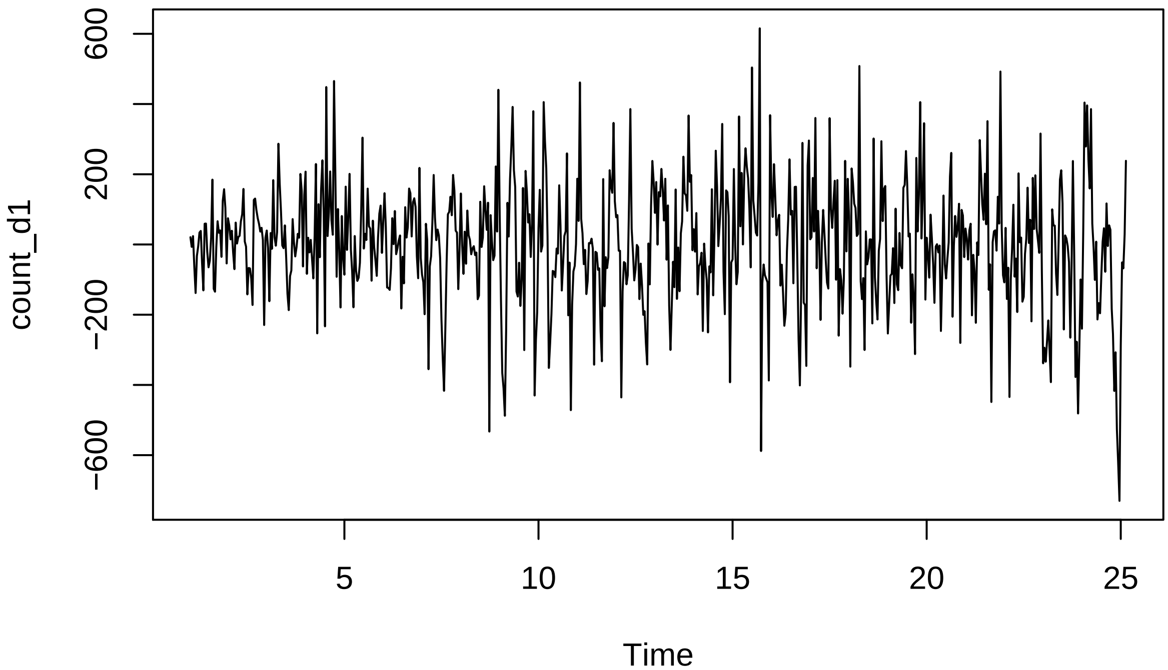
```r
Acf(count_ma, main='')
```




```
Pacf(count_ma, main='')
```



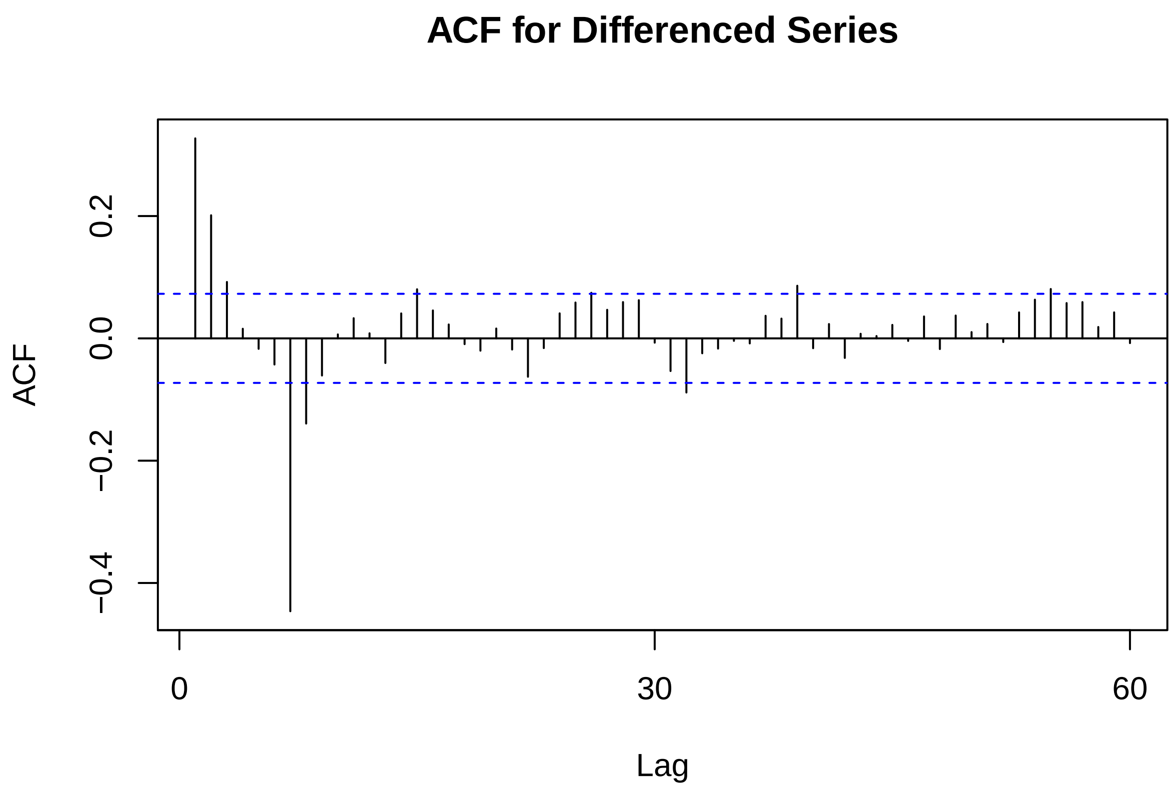

```r
count_d1 = diff(deseasonal_cnt, differences = 1)
plot(count_d1)
```



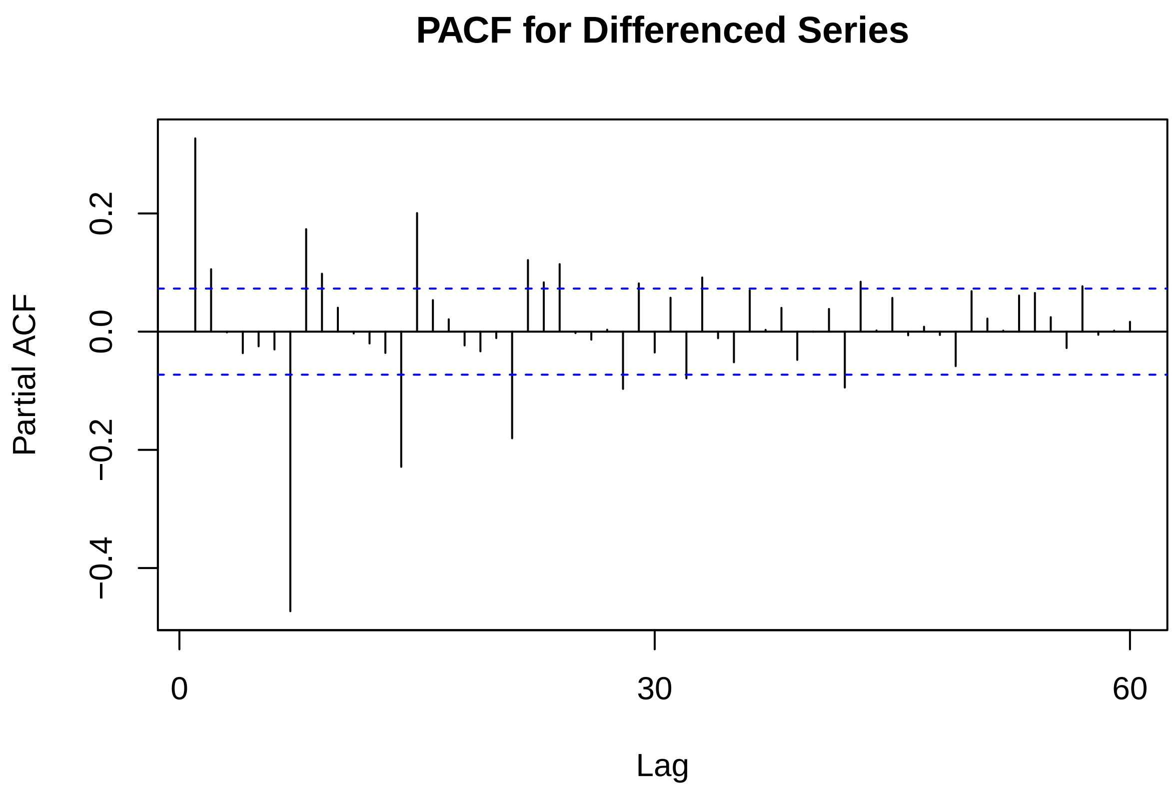

```r
adf.test(count_d1, alternative = "stationary")
```

```
## Warning in adf.test(count_d1, alternative = "stationary"): p-value smaller
## than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  count_d1
## Dickey-Fuller = -9.9255, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

```r
Acf(count_d1, main='ACF for Differenced Series')
```




```
Pacf(count_d1, main='PACF for Differenced Series')
```



There are significant auto correlations at lag 1 and 2 and beyond. Partial correlation plots show a significant spike at lag 1 and 7. This suggests that we might want to test models with AR or MA components of order 1, 2, or 7. A spike at lag 7 might suggest that there is a seasonal pattern present, perhaps as day of the week. We talk about how to choose model order in the next step.

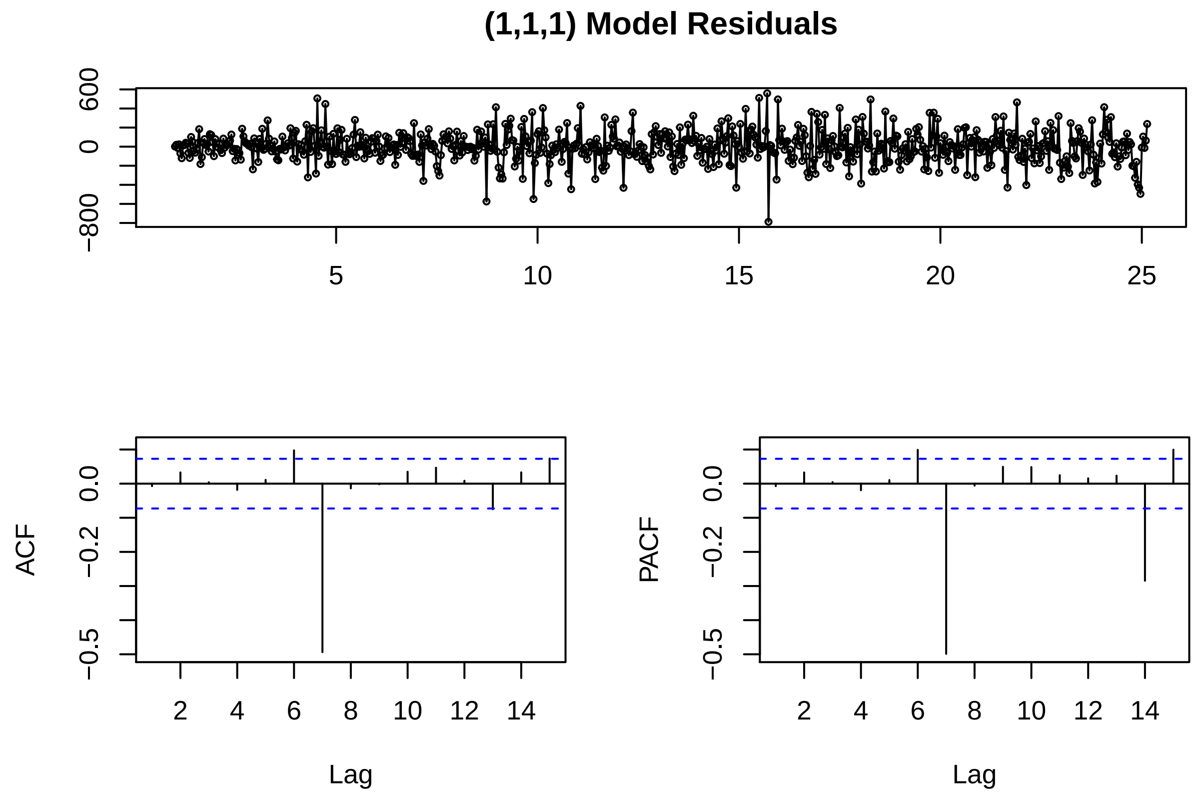
Fitting an ARIMA model

While auto.arima() can be very useful, it is still important to complete steps 1-5 in order to understand the series and interpret model results. Note that auto.arima() also allows the user to specify maximum order for (p, d, q), which is set to 5 by default.

```r
auto.arima(deseasonal_cnt, seasonal=FALSE)
```

```
## Series: deseasonal_cnt
## ARIMA(1,1,1)
##
## Coefficients:
##          ar1      ma1
##       0.5510  -0.2496
## s.e.  0.0751   0.0849
##
## sigma^2 estimated as 26180:  log likelihood=-4708.91
## AIC=9423.82   AICc=9423.85   BIC=9437.57
```

```r
fit<-auto.arima(deseasonal_cnt, seasonal=FALSE)
tsdisplay(residuals(fit), lag.max=15, main='(1,1,1) Model Residuals')
```

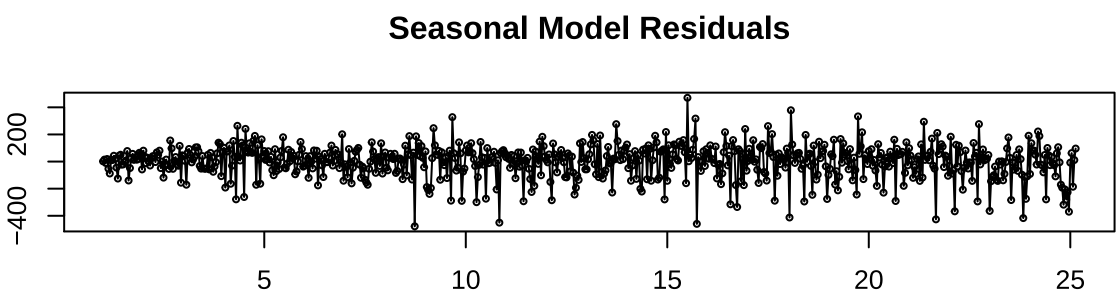


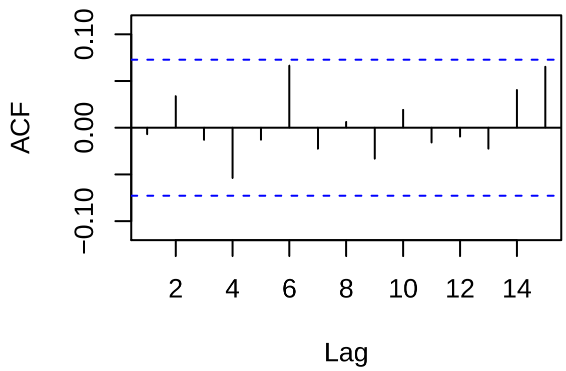
There is a clear pattern present in ACF/PACF and model residuals plots repeating at lag 7. This suggests that our model may be better off with a different specification, such as p = 7 or q = 7.

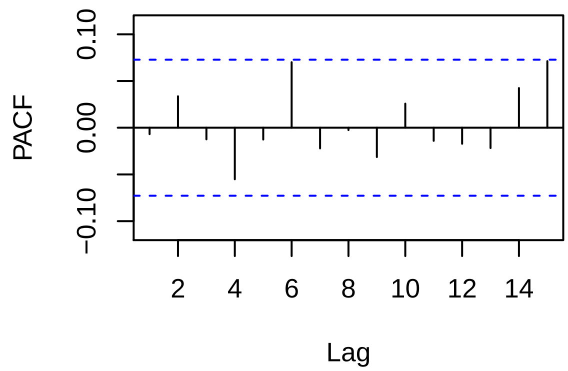
```
fit2 = arima(deseasonal_cnt, order=c(1,1,7))

fit2
```

```
##
## Call:
## arima(x = deseasonal_cnt, order = c(1, 1, 7))
##
## Coefficients:
##          ar1     ma1     ma2     ma3     ma4     ma5     ma6      ma7
##       0.2803  0.1465  0.1524  0.1263  0.1225  0.1291  0.1471  -0.8353
## s.e.  0.0478  0.0289  0.0266  0.0261  0.0263  0.0257  0.0265   0.0285
##
## sigma^2 estimated as 14392:  log likelihood = -4503.28,  aic = 9024.56
```

```
tsdisplay(residuals(fit2), lag.max=15, main='Seasonal Model Residuals')
```

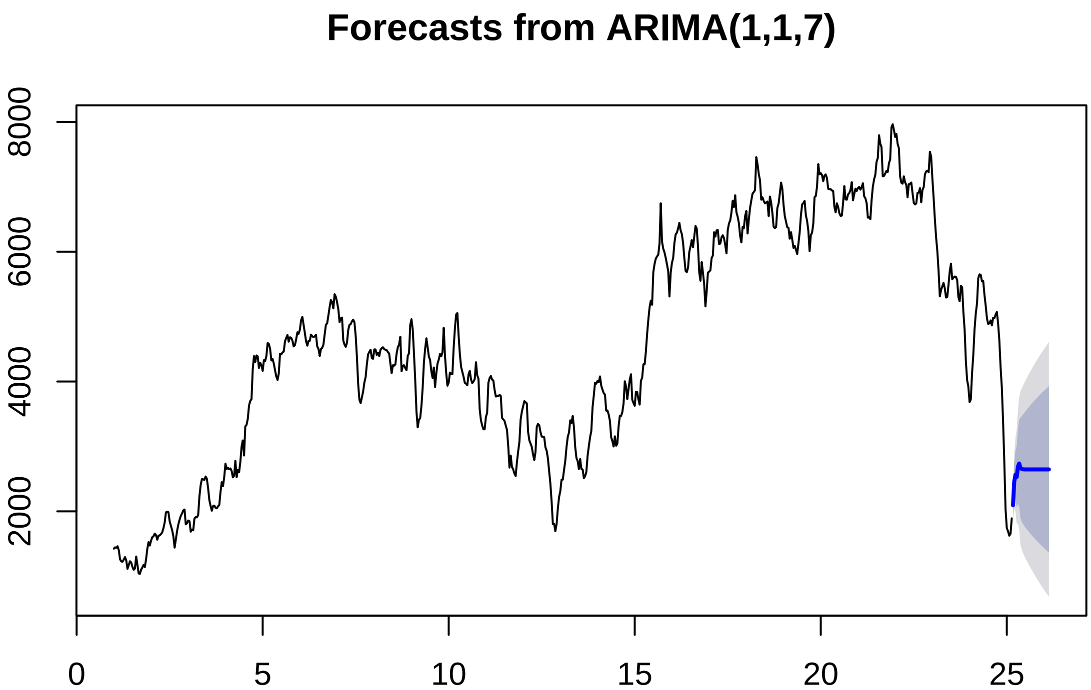






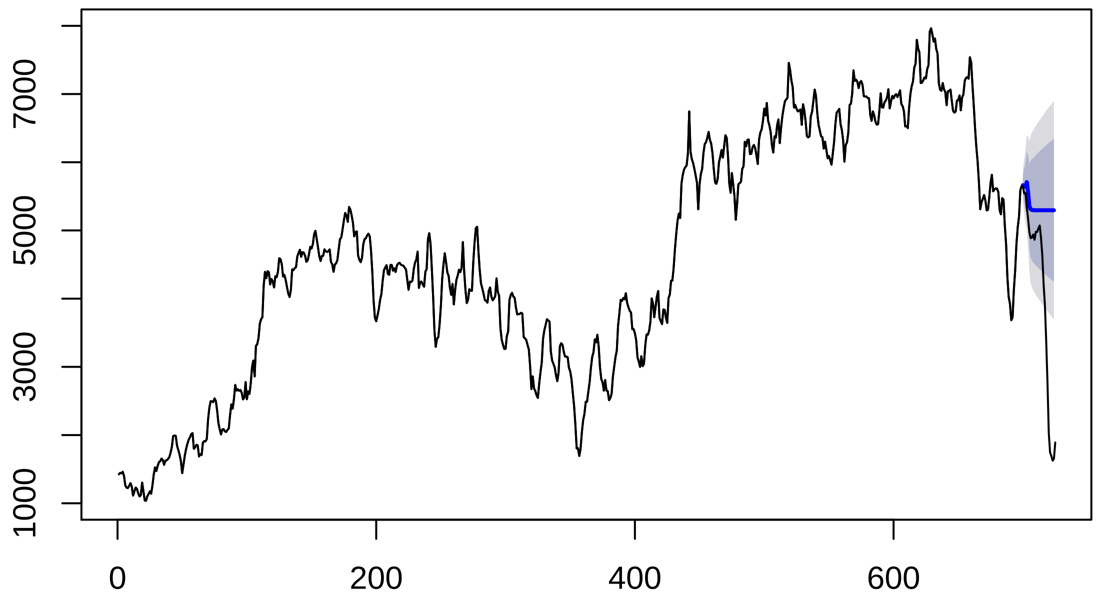
Forecasting

```
fcast <- forecast(fit2, h=30)
plot(fcast)
```



Train and test data

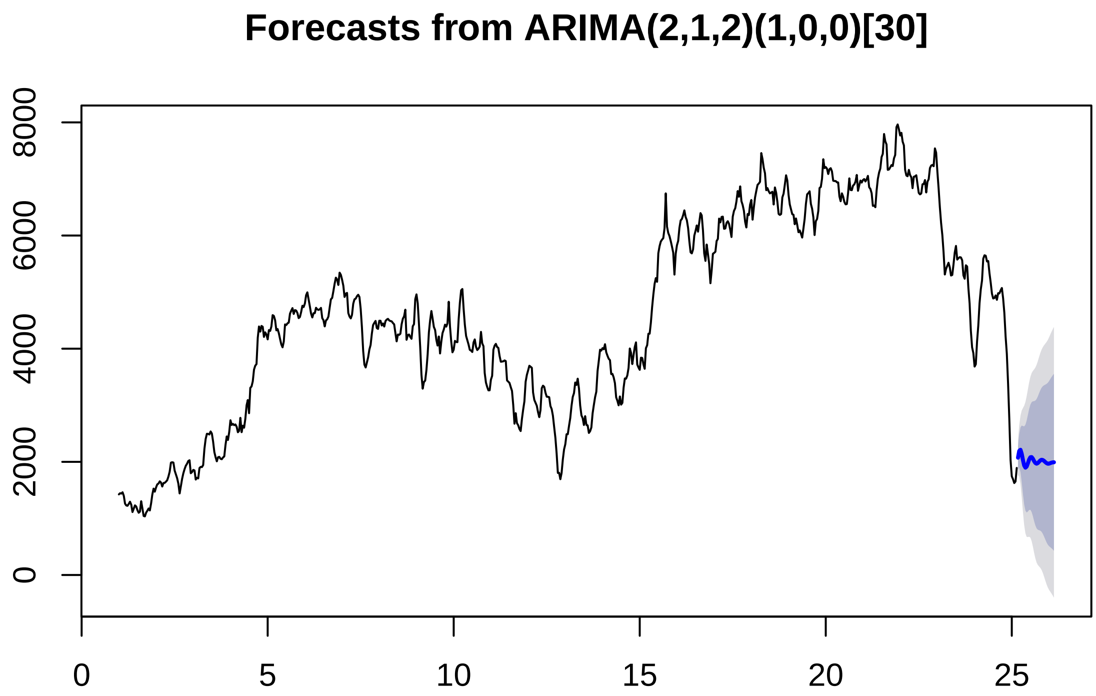
```
test <- window(ts(deseasonal_cnt), start=700)

fit_training = arima(ts(deseasonal_cnt[-c(700:725)]), order=c(1,1,7))

fcast_training <- forecast(fit_training,h=25)
plot(fcast_training, main=" ")
lines(ts(deseasonal_cnt))
```



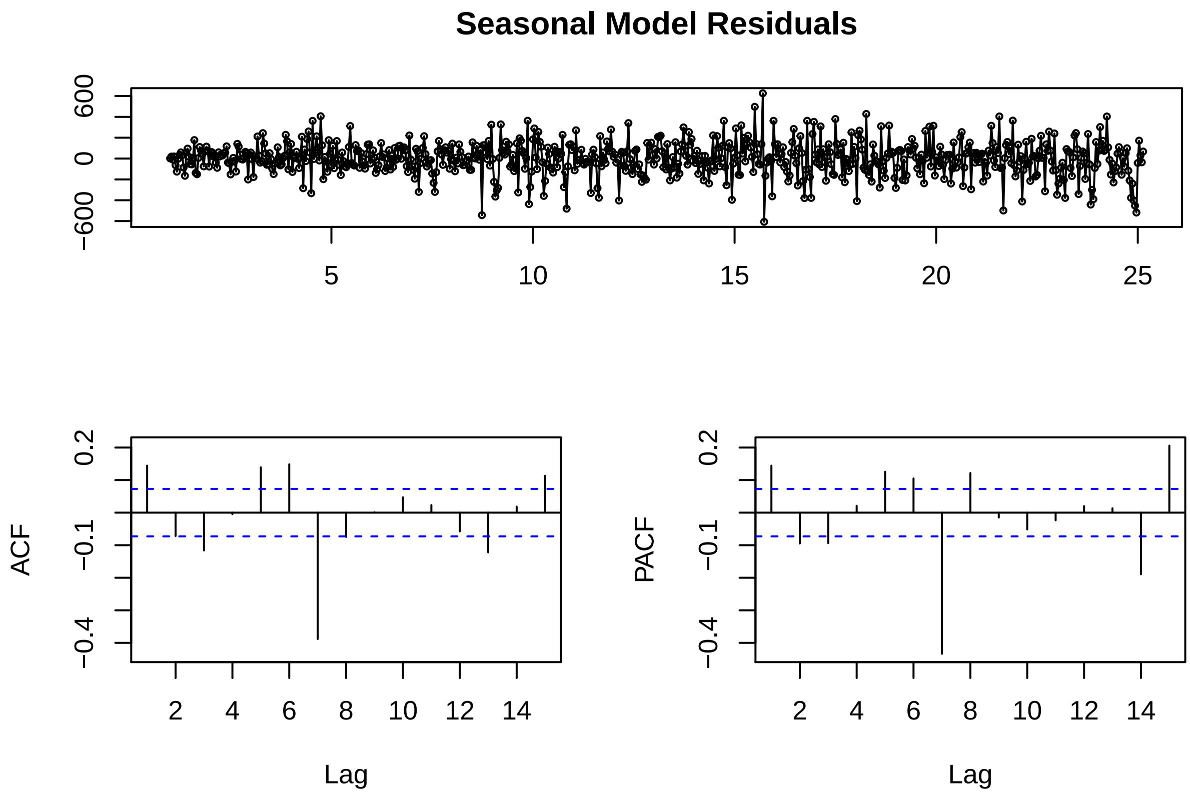

```r
fit_w_seasonality = auto.arima(deseasonal_cnt, seasonal=TRUE)
fit_w_seasonality
```

```
## Series: deseasonal_cnt
## ARIMA(2,1,2)(1,0,0)[30]
##
## Coefficients:
##          ar1      ar2      ma1     ma2    sar1
##       1.3644  -0.8027  -1.2903  0.9146  0.0100
## s.e.  0.0372   0.0347   0.0255  0.0202  0.0388
##
## sigma^2 estimated as 24810:  log likelihood=-4688.59
## AIC=9389.17   AICc=9389.29   BIC=9416.68
```

```r
seas_fcast <- forecast(fit_w_seasonality, h=30)
plot(seas_fcast)
```




```r
tsdisplay(residuals(seas_fcast), lag.max=15, main='Seasonal Model Residuals')
```


Seasonal Model Residuals
600
0
−600
5
10
15
20
25
ACF
0.2
−0.1
−0.4
2
4
6
8
10
12
14
Lag
PACF
0.2
−0.1
−0.4
2
4
6
8
10
12
14
Lag