DATASCIENCE

Introduction to Forecasting with ARIMA in R

Ruslana Dalinina | 01.10.17

Prerequisites

Experience with the specific topic: Novice

Professional experience: Some industry experience

Previous knowledge of forecasting is not required, but the reader should be familiar with basic data analysis and statistics (e.g., averages, correlation). To follow the example, the reader should also be familiar with R syntax. R packages needed: **`forecast`**, **`tseries`**, **`ggplot2`**. The sample dataset can be downloaded here.

DATASCIENCE

This tutorial will provide a step-by-step guide for fitting an ARIMA model using R. ARIMA models are a popular and flexible class of forecasting model that utilize historical information to make predictions. This type of model is a basic forecasting technique that can be used as a foundation for more complex models. 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. We also provide a checklist for basic ARIMA modeling to be used as a loose guide.

Business Uses

Time series analysis can be used in a multitude of business applications for forecasting a quantity into the future and explaining its historical patterns. Here are just a few examples of possible use cases:

- Explaining seasonal patterns in sales
- Predicting the expected number of incoming or churning customers
- Estimating the effect of a newly launched product on number of sold units
- Detecting unusual events and estimating the magnitude of their effect

Objectives

At the end of this tutorial, the reader can expect to learn how to:

- Plot, examine, and prepare series for modeling
- Extract the seasonality component from the time series
- Test for stationarity and apply appropriate transformations
- Choose the order of an ARIMA model

Readers can use the following ARIMA cheat sheet as an outline of this tutorial and general guidance when fitting these types of models:

1. Examine your data
   - Plot the data and examine its patterns and irregularities
   - Clean up any outliers or missing values if needed
   - **`tsclean()`** is a convenient method for outlier removal and inputing missing values
   - Take a logarithm of a series to help stabilize a strong growth trend
2. Decompose your data
   - Does the series appear to have trends or seasonality?
   - Use **`decompose()`** or **`stl()`** to examine and possibly remove components of the series
3. Stationarity
   - Is the series stationary?
   - Use **`adf.test()`**, ACF, PACF plots to determine order of differencing needed
4. Autocorrelations and choosing model order
   - Choose order of the ARIMA by examining ACF and PACF plots
5. Fit an ARIMA model
6. Evaluate and iterate

- If there are visible patterns or bias, plot ACF / PACF. Are any additional order parameters needed?
- Refit model if needed. Compare model errors and fit criteria such as AIC or BIC.
- Calculate forecast using the chosen model

A Short Introduction to ARIMA

ARIMA stands for auto-regressive integrated moving average and is specified by these three order parameters: (p, d, q). The process of fitting an ARIMA model is sometimes referred to as the Box-Jenkins method.

An **auto regressive (AR(p))** component is referring to the use of past values in the regression equation for the series Y. The auto-regressive parameter p specifies the number of lags used in the model. For example, AR(2) or, equivalently, ARIMA(2,0,0), is represented as

$$Y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + e_t$$

where φ_1, φ_2 are parameters for the model.

The d represents the degree of differencing in the **integrated ($I(d)$)** component. Differencing a series involves simply subtracting its current and previous values d times. Often, differencing is used to stabilize the series when the stationarity assumption is not met, which we will discuss below.

A **moving average (MA(q))** component represents the error of the model as a combination of previous error terms e_t. The order q determines the number of terms to include in the model

Differencing, autoregressive, and moving average components make up a non-seasonal ARIMA model which can be written as a linear equation:

$$Y_t = c + \phi_1 y_{d\ t-1} + \phi_p y_{d\ t-p} + \ldots + \theta_1 e_{t-1} + \theta_q e_{t-q} + e_t$$

where y_d is Y differenced d times and c is a constant.

Note that the model above assumes non-seasonal series, which means you might need to de-seasonalize the series before modeling. We will show how this can be done in an example below.

ARIMA models can be also specified through a seasonal structure. In this case, the model is specified by two sets of order parameters: (p, d, q) as described above and $(P, D, Q)_m$ parameters describing the seasonal component of m periods.

ARIMA methodology does have its limitations. These models directly rely on past values, and therefore work best on long and stable series. Also note that ARIMA simply approximates historical patterns and therefore does not aim to explain the structure of the underlying data mechanism.

Step 1: Load R Packages

We start out by loading the necessary R packages and reading in the analysis dataset. Here we are using a dataset on the number of bicycles checkouts from a bike sharing service, which is available as part of the UCI Machine Learning Repository. We will be using the dataset aggregated at daily level.

DATASCIENCE

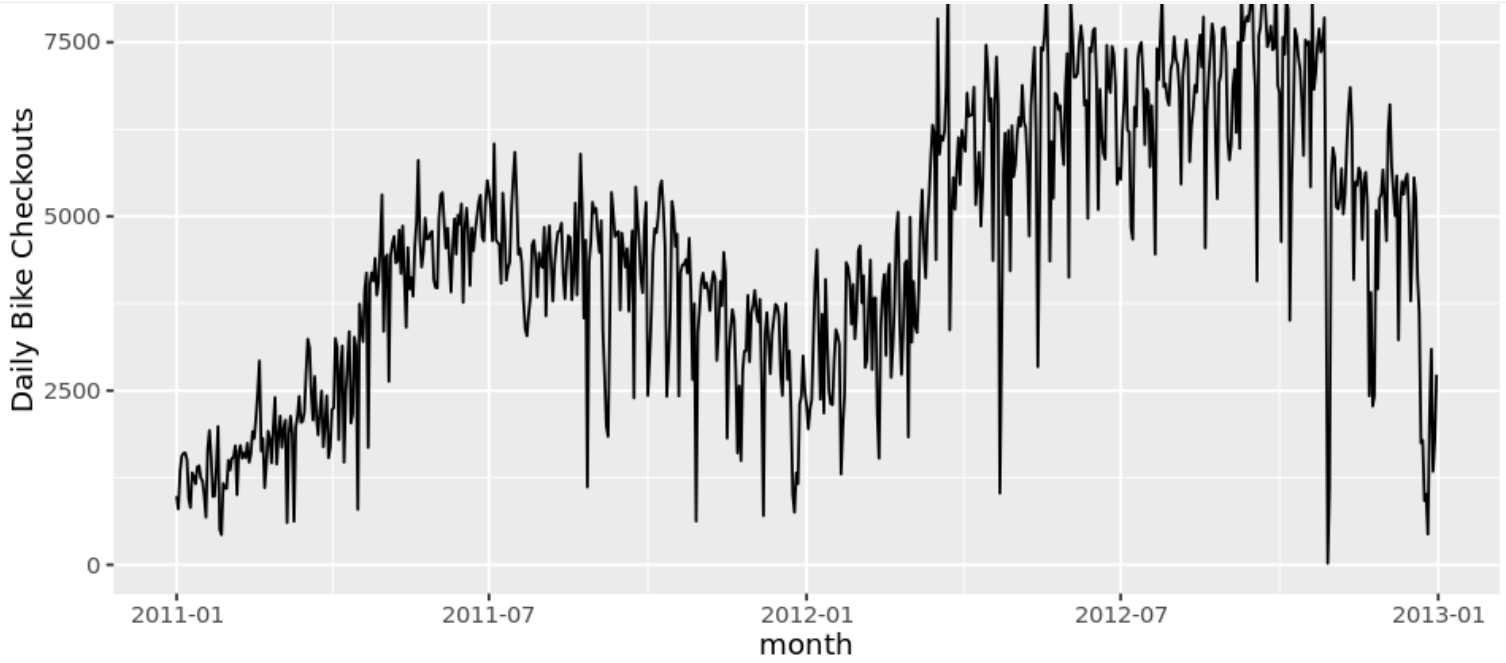
```
library('forecast')
library('tseries')

daily_data = read.csv('day.csv', header=TRUE, stringsAsFactors=FALSE)
```

Step 2: Examine Your Data

A good stating point is to plot the series and visually examine it for any outliers, volatility, or irregularities. In this case, bicycle checkouts are showing a lot of fluctuations from one day to another. However, even with this volatility present, we already see some patterns emerge. For example, 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("Dai
          xlab("")
```

DATASCIENCE

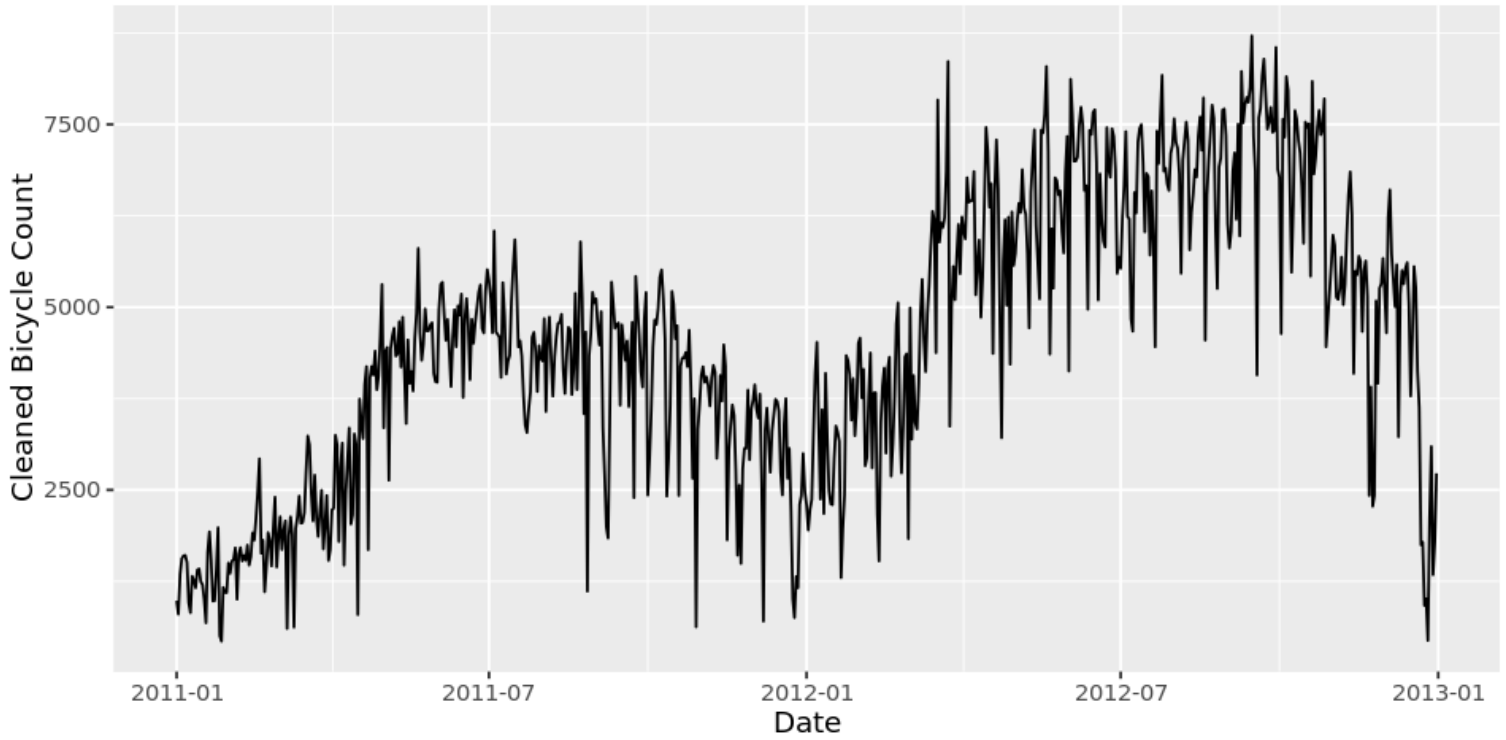


In some cases, the number of bicycles checked out dropped below 100 on day and rose to over 4,000 the next day. These are suspected outliers that could bias the model by skewing statistical summaries. R provides a convenient method for removing time series outliers: tsclean() as part of its forecast package. tsclean() identifies and replaces outliers using series smoothing and decomposition. This method is also capable of inputing missing values in the series if there are any.

Note that we are using the ts() command to create a time series object to pass to tsclean():

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

daily_data$clean_cnt = tsclean(count_ts)

```

We plot the clean series using ggplot:



Even after removing outliers, the daily data is still pretty volatile. Visually, we could a draw a line through the series tracing its bigger troughs and peaks while smoothing out noisy fluctuations. This line can be described by one of the simplest – but also very useful –concepts in time series analysis known as a moving average. It is an intuitive concept that averages points across several time periods, thereby smoothing the observed data into a more stable predictable series.

around each point.

$$MA = \frac{1}{m}\sum_{j=-k}^{k} y_{t+j}$$

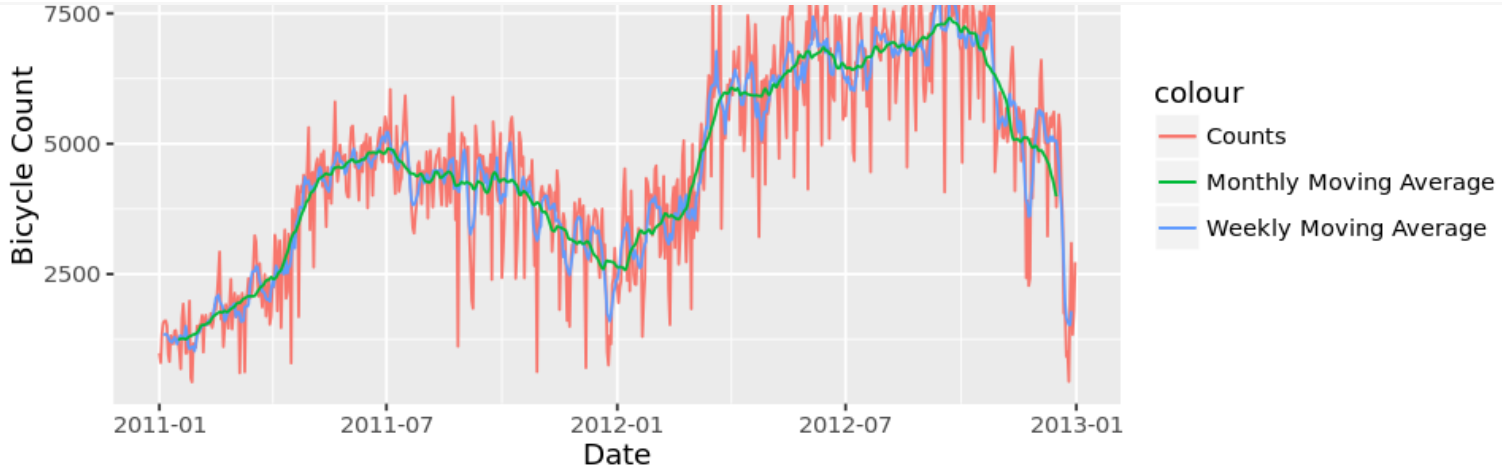
where $m = 2k + 1$. The above quantity is also called a symmetric moving average because data on each side of a point is involved in the calculation.

Note that the moving average in this context is distinct from the $MA(q)$ component in the above ARIMA definition. Moving average $MA(q)$ as part of the ARIMA framework refers to error lags and combinations, whereas the summary statistic of moving average refers to a data smoothing technique.

The wider the window of the moving average, the smoother original series becomes. In our bicycle example, 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
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 Av
  geom_line(data = daily_data, aes(x = Date, y = cnt_ma30, colour = "Monthly Moving A
  ylab('Bicycle Count')
```

DATASCIENCE



In addition to volatility, modeling daily level data might require specifying multiple seasonality levels, such day of the week, week of the year, month of the year, holidays, etc. For the sake of simplicity, we will model the smoothed series of weekly moving average (as shown by the blue line above).

Step 3: Decompose Your Data

The building blocks of a time series analysis are seasonality, trend, and cycle. These intuitive components capture the historical patterns in the series. Not every series will have all three (or any) of these components, but if they are present, deconstructing the series can help you understand its behavior and prepare a foundation for building a forecasting model.

Seasonal component refers to fluctuations in the data related to calendar cycles. For example, more people might be riding bikes in the summer and during warm weather, and less during colder months. Usually, seasonality is fixed at some number; for instance, quarter or month of the year.

Cycle component consists of decreasing or increasing patterns that are not seasonal. Usually, trend and cycle components are grouped together. Trend-cycle component is estimated using moving averages.

Finally, part of the series that can't be attributed to seasonal, cycle, or trend components is referred to as **residual** or **error.**

The process of extracting these components is referred to as **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,

$$Y = S_t * T_t * E_t$$

where S_t 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.

As mentioned above, ARIMA models can be fitted to both seasonal and non-seasonal data. Seasonal ARIMA requires a more complicated specification of the model structure, although the process of

DATASCIENCE

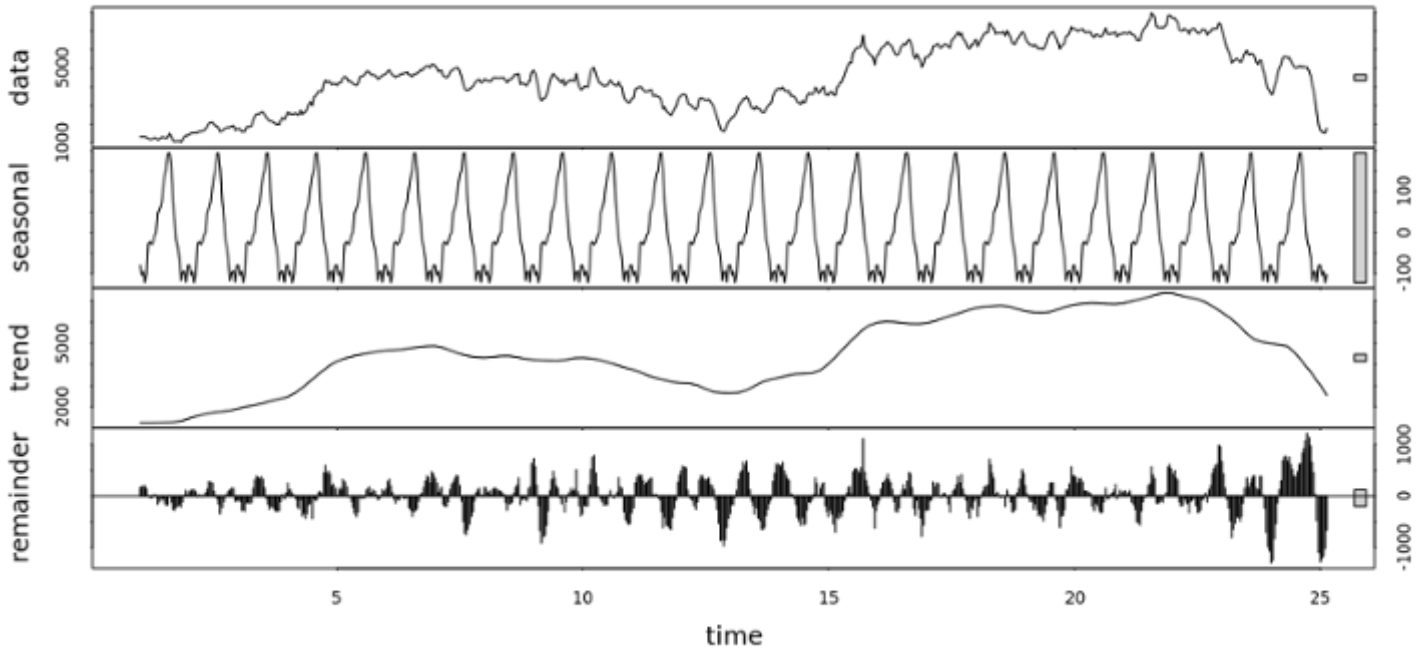
First, we calculate seasonal component of the data using **stl()**. STL is a flexible function for decomposing and forecasting the series. It calculates the seasonal component of the series using smoothing, and adjusts the original series by subtracting seasonality in two simple lines:

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

Note that **stl()** by default assumes additive model structure. Use **allow.multiplicative.trend=TRUE** to incorporate the multiplicative model.

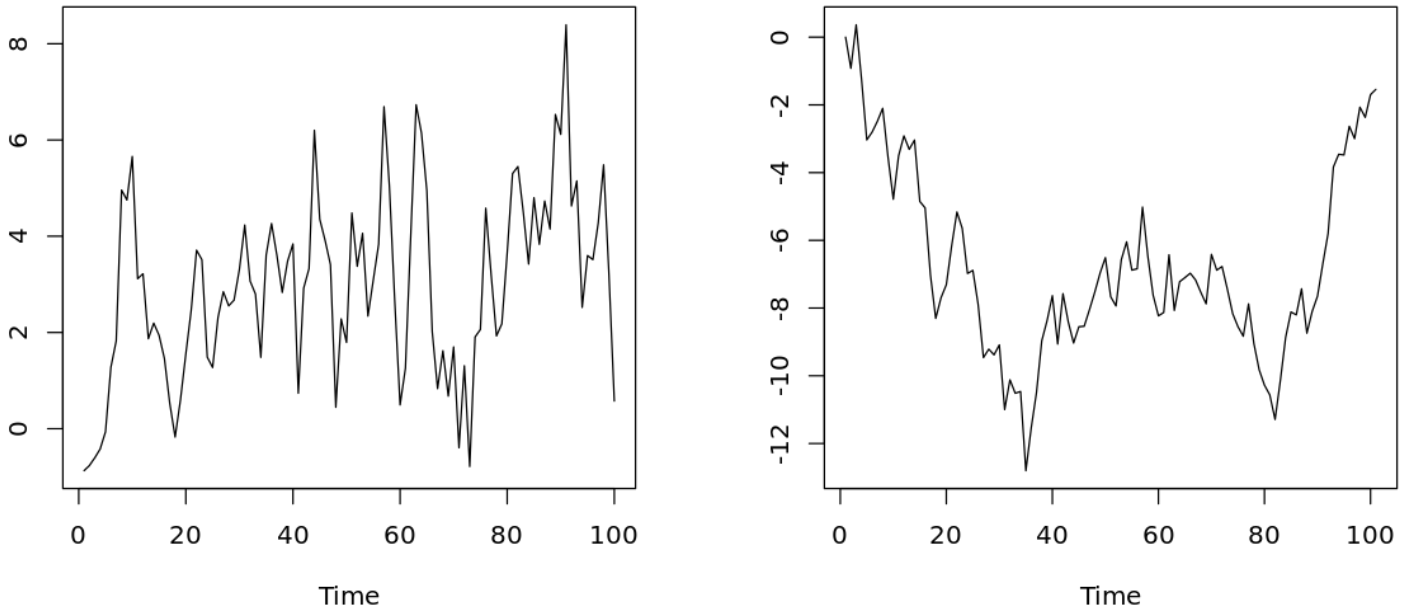
In the case of additive model structure, the same task of decomposing the series and removing the seasonality can be accomplished by simply subtracting the seasonal component from the original series. **seasadj()** is a convenient method inside the **forecast** package.

As for the frequency parameter in **ts()** object, we are specifying periodicity of the data, i.e., number of observations per period. Since we are using smoothed daily data, we have 30 observations per month.



Step 4: Stationarity

Fitting an ARIMA model requires the series to be **stationary**. A series is said to be stationary when its mean, variance, and autocovariance are time invariant. This assumption makes intuitive sense: Since ARIMA uses previous lags of series to model its behavior, modeling stable series with consistent properties involves less uncertainty. The left panel below shows an example of a stationary series, where data values oscillate with a steady variance around the mean of 1. The panel on the right shows a non-stationary series; mean of this series will differ across different time windows.



The augmented Dickey-Fuller (ADF) test is a formal statistical test for stationarity. The null hypothesis assumes that the series is non-stationary. ADF procedure tests whether the change in Y can be explained by lagged value and a linear trend. If contribution of the lagged value to the change in Y is non-significant and there is a presence of a trend component, the series is non-stationary and null hypothesis will not be rejected.

Our bicycle data is non-stationary; the average number of bike checkouts changes through time, levels change, etc. A formal ADF test does not reject the null hypothesis of non-stationarity, confirming our visual inspection:

```
2
3       Augmented Dickey-Fuller Test
4
5   data:  count_ma
6   Dickey-Fuller = -0.2557, Lag order = 8, p-value = 0.99
7   alternative hypothesis: stationary
```

Usually, non-stationary series can be corrected by a simple transformation such as differencing. Differencing the series can help in removing its trend or cycles. The idea behind differencing is that, if the original data series does not have constant properties over time, then the change from one period to another might. The difference is calculated by subtracting one period's values from the previous period's values:

$$Y_{d_t} = Y_t - Y_{t-1}$$

Higher order differences are calculated in a similar fashion. For example, second order differencing $(d = 2)$ is simply expanded to include second lag of the series:

$$Y_{d2_t} = Y_{d_t} - Y_{d_t-1} = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2})$$

Similarly, differencing can be used if there is a seasonal pattern at specific lags. In such a case, subtracting a value for the "season" from a previous period represents the change from one period to another, as well as from one season to another:

The number of differences performed is represented by the d component of ARIMA. Now, we move on to diagnostics that can help determine the order of differencing.

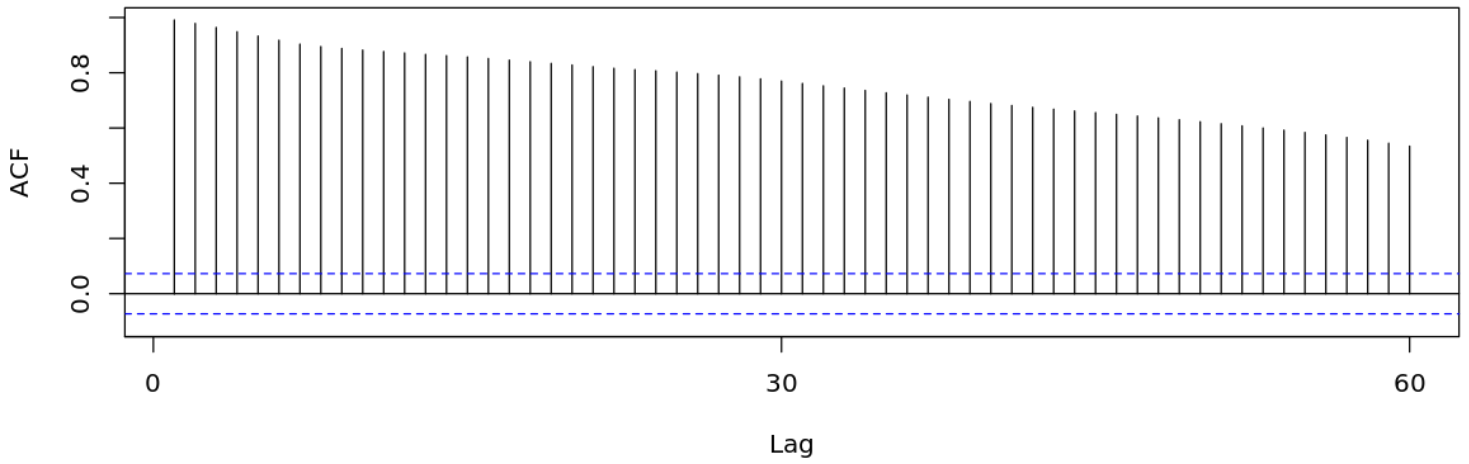
Step 5: Autocorrelations and Choosing Model Order

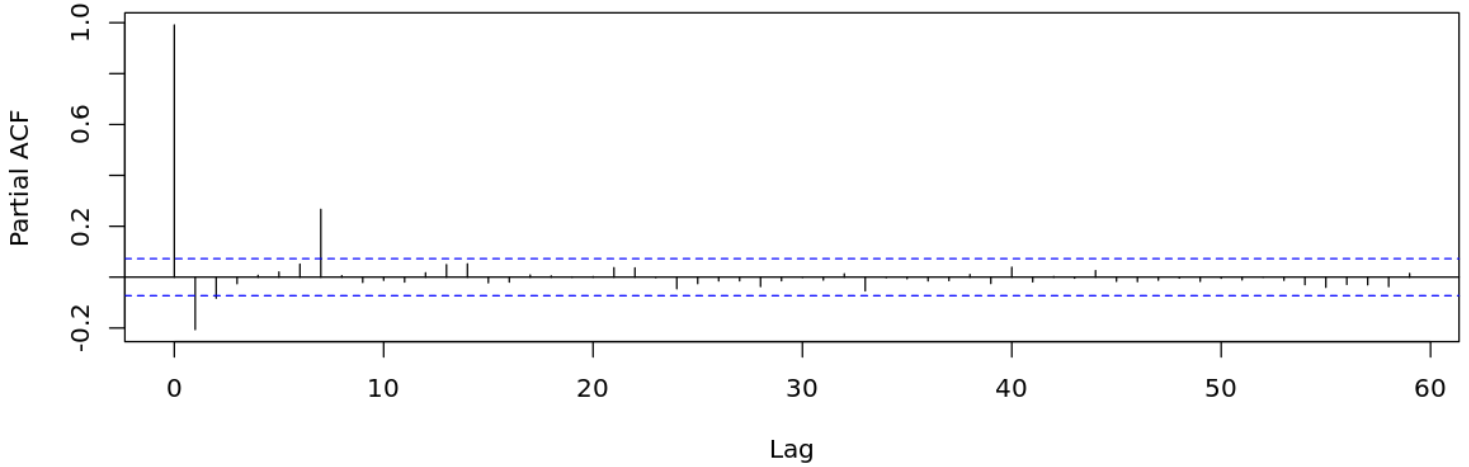
Autocorrelation plots (also known as ACF or the auto correlation function) are a useful visual tool in determining whether a series is stationary. These plots can also help to choose the order parameters for ARIMA model. If the series is correlated with its lags then, generally, there are some trend or seasonal components and therefore its statistical properties are not constant over time.

ACF plots display correlation between a series and its lags. In addition to suggesting the order of differencing, ACF plots can help in determining the order of the $MA\ (q)$ model. Partial autocorrelation plots (PACF), as the name suggests, display correlation between a variable and its lags that is not explained by previous lags. PACF plots are useful when determining the order of the $AR(p)$ model.

R plots 95% significance boundaries as blue dotted lines. There are significant autocorrelations with many lags in our bike series, as shown by the ACF plot below. However, this could be due to carry-over correlation from the first or early lags, since the PACF plot only shows a spike at lags 1 and 7:

```
1  Acf(count_ma, main='')
2
3  Pacf(count_ma, main='')
```



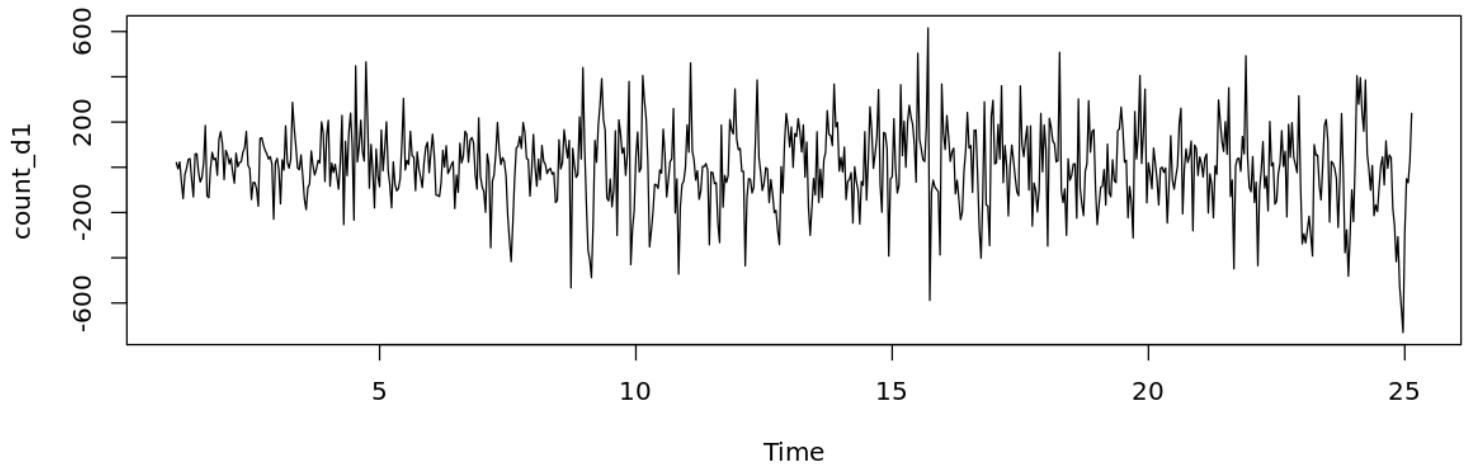


We can start with the order of $d = 1$ and re-evaluate whether further differencing is needed.

DATASCIENCE

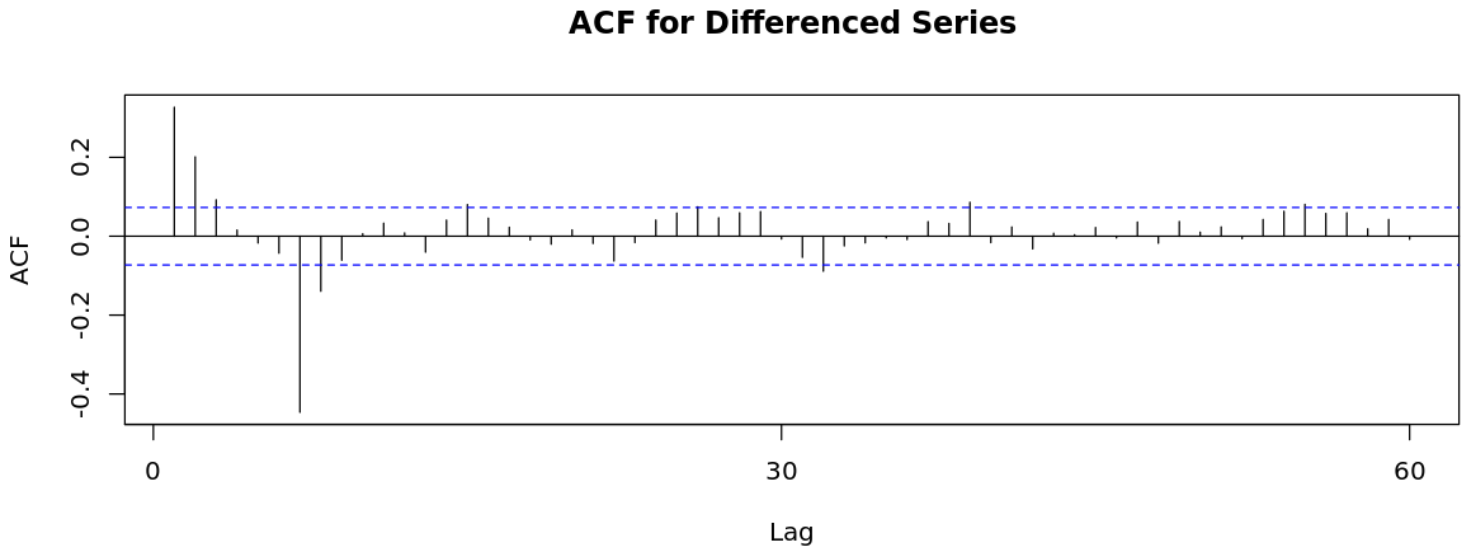
suggests that differencing of order 1 terms is sufficient and should be included in the model.

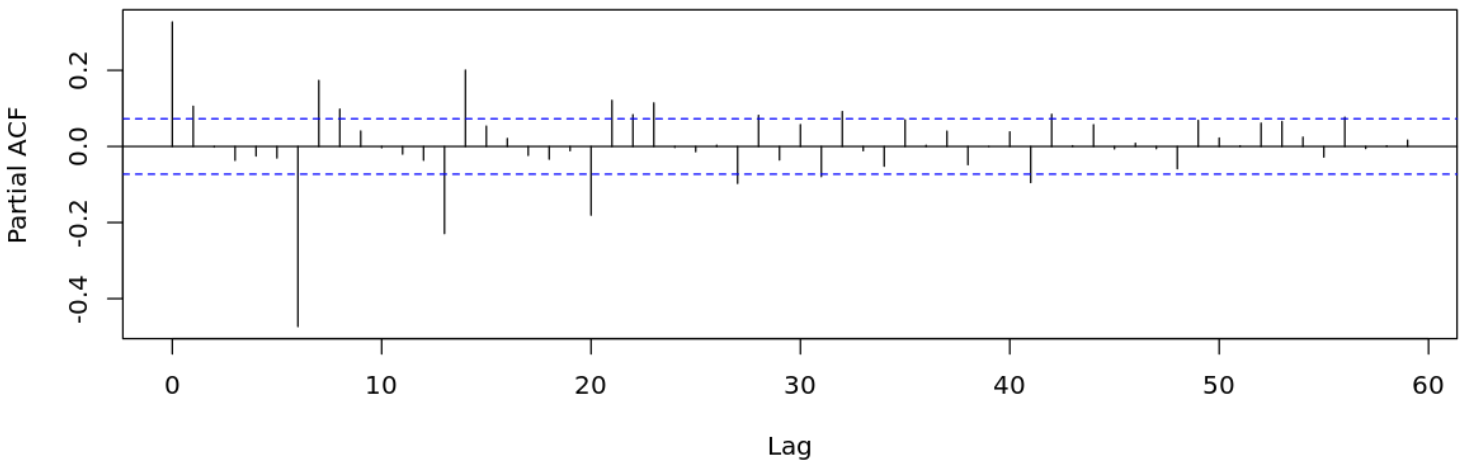
```
count_d1 = diff(deseasonal_cnt, differences = 1)
plot(count_d1)
adf.test(count_d1, alternative = "stationary")

Augmented Dickey-Fuller Test

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



```
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.





Step 6: Fitting an ARIMA model

Now let's fit a model. The **forecast** package allows the user to explicitly specify the order of the model using the **arima()** function, or automatically generate a set of optimal *(p, d, q)* using **auto.arima()**. This function searches through combinations of order parameters and picks the set that optimizes model fit criteria.

There exist a number of such criteria for comparing quality of fit across multiple models. Two of the most widely used are Akaike information criteria (AIC) and Baysian information criteria (BIC). These criteria are closely related and can be interpreted as an estimate of how much information would be lost if a given model is chosen. When comparing models, one wants to minimize AIC and BIC.

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.

model incorporates differencing of degree 1, and uses an autoregressive term of first lag and a moving average model of order 1:

```
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
```

Using the ARIMA notation introduced above, the fitted model can be written as

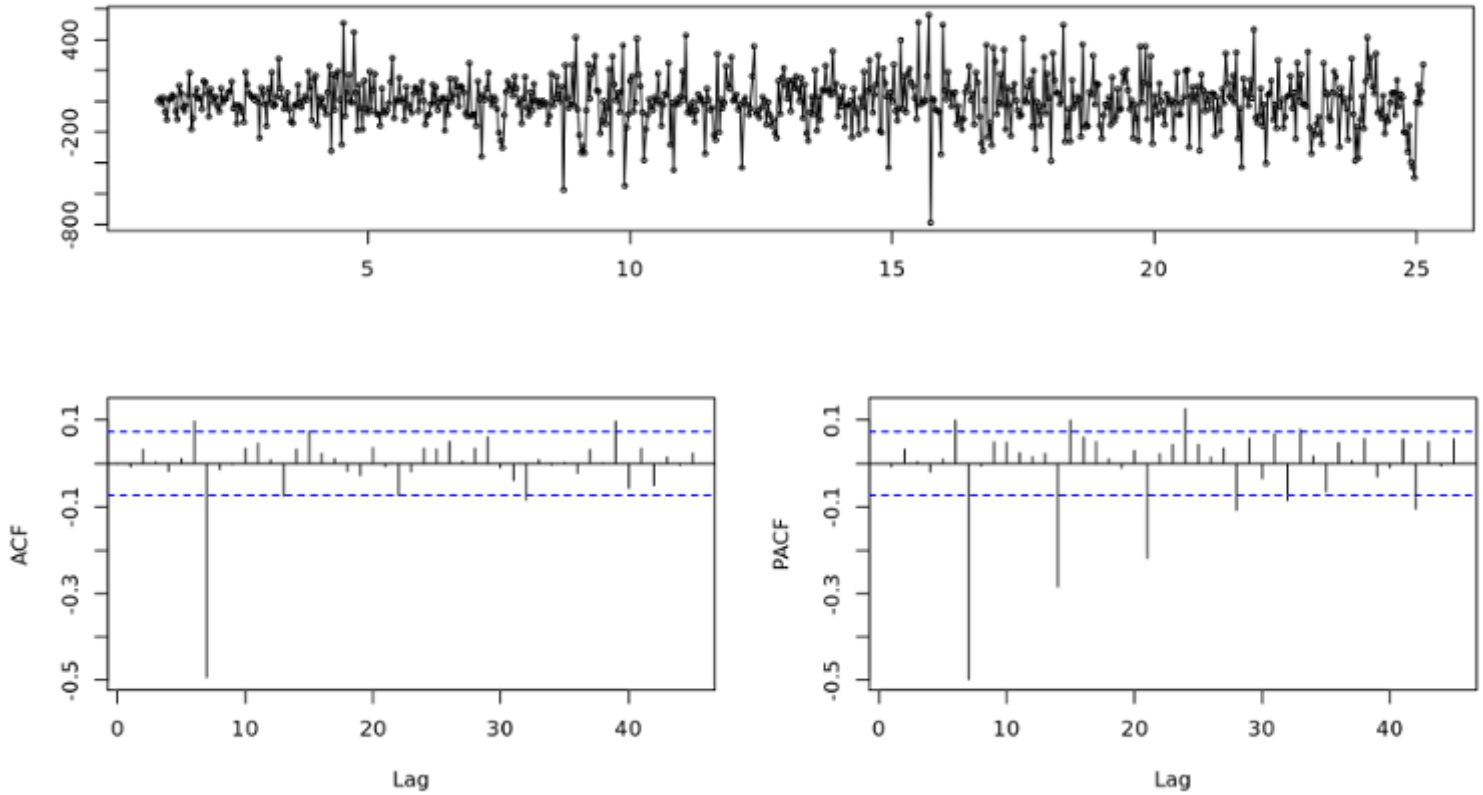
$$\hat{Y}_{d_t} = 0.551Y_{t-1} - 0.2496e_{t-1} + E$$

where E is some error and the original series is differenced with order 1.

Step 7: Evaluate and Iterate

So now we have fitted a model that can produce a forecast, but does it make sense? Can we trust this model? We can start by examining ACF and PACF plots for model residuals. If model order parameters and structure are correctly specified, we would expect no significant autocorrelations present.

```
fit<-auto.arima(deseasonal_cnt, seasonal=FALSE)
tsdisplay(residuals(fit), lag.max=45, main='(1,1,1) Model Residuals')
```



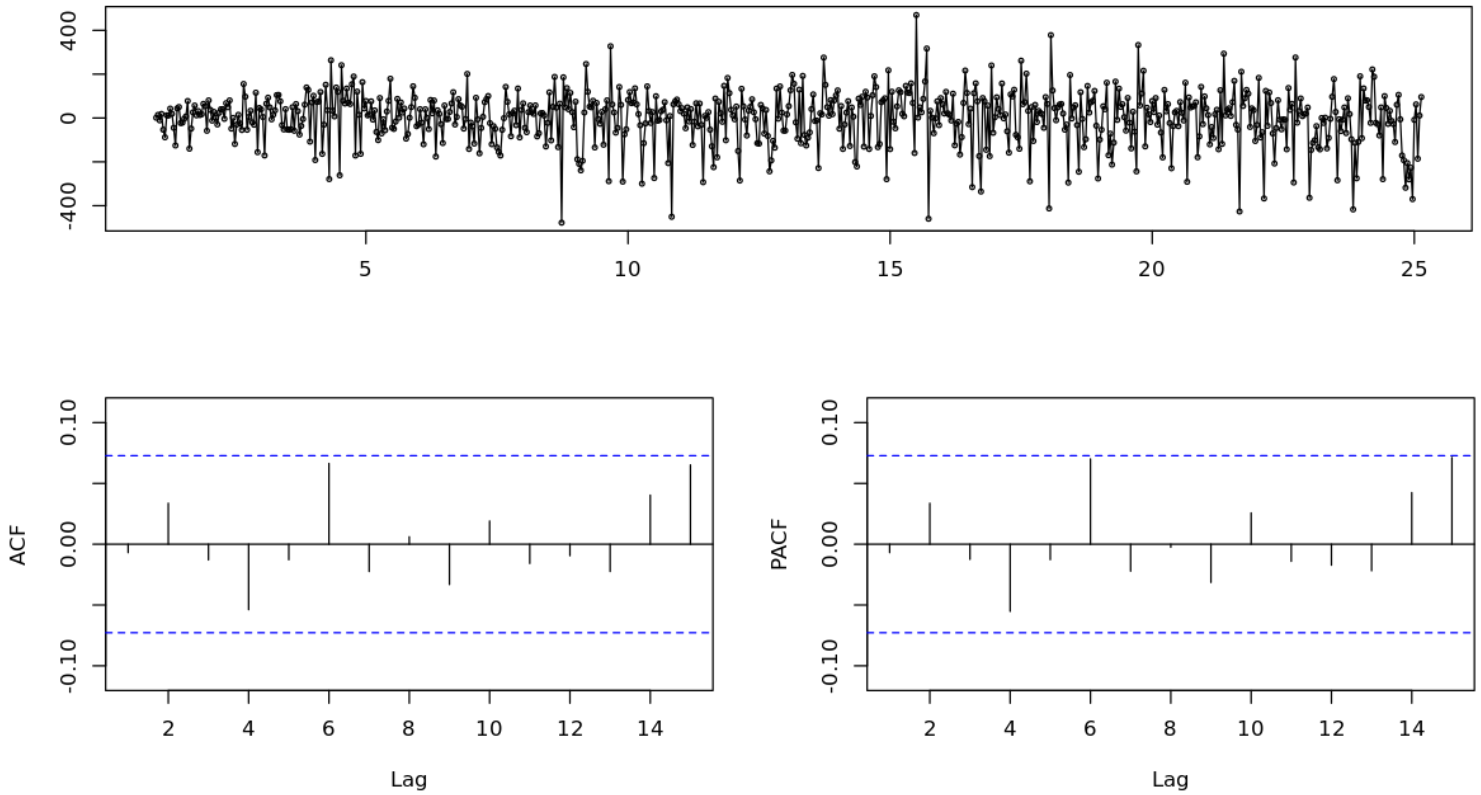
There is a clear pattern present in ACF/PACF and model residuals plots repeating at lag 7. This suggests that our model may be better off with a different specification, such as $p = 7$ or $q = 7$.

We can repeat the fitting process allowing for the MA(7) component and examine diagnostic plots again. This time, there are no significant autocorrelations present. If the model is not correctly specified, that will usually be reflected in residuals in the form of trends, skeweness, or any other patterns not captured by the model. Ideally, residuals should look like white noise, meaning they are normally distributed. A convenience function **`tsdisplay()`** can be used to plot these model diagnostics. Residuals plots show a

DATASCIENCE

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

fit2

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

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
```

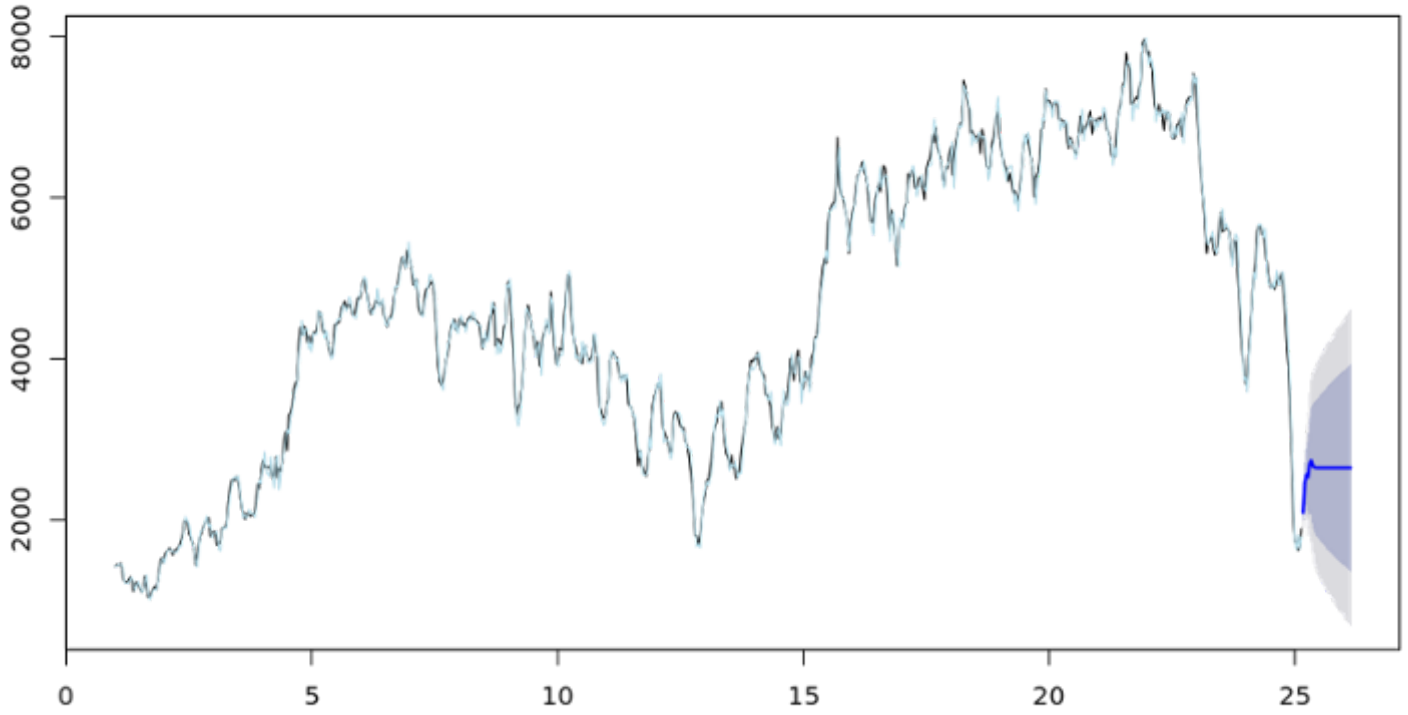
DATASCIENCE



Forecasting using a fitted model is straightforward in R. We can specify forecast horizon h periods ahead for predictions to be made, and use the fitted model to generate those predictions:

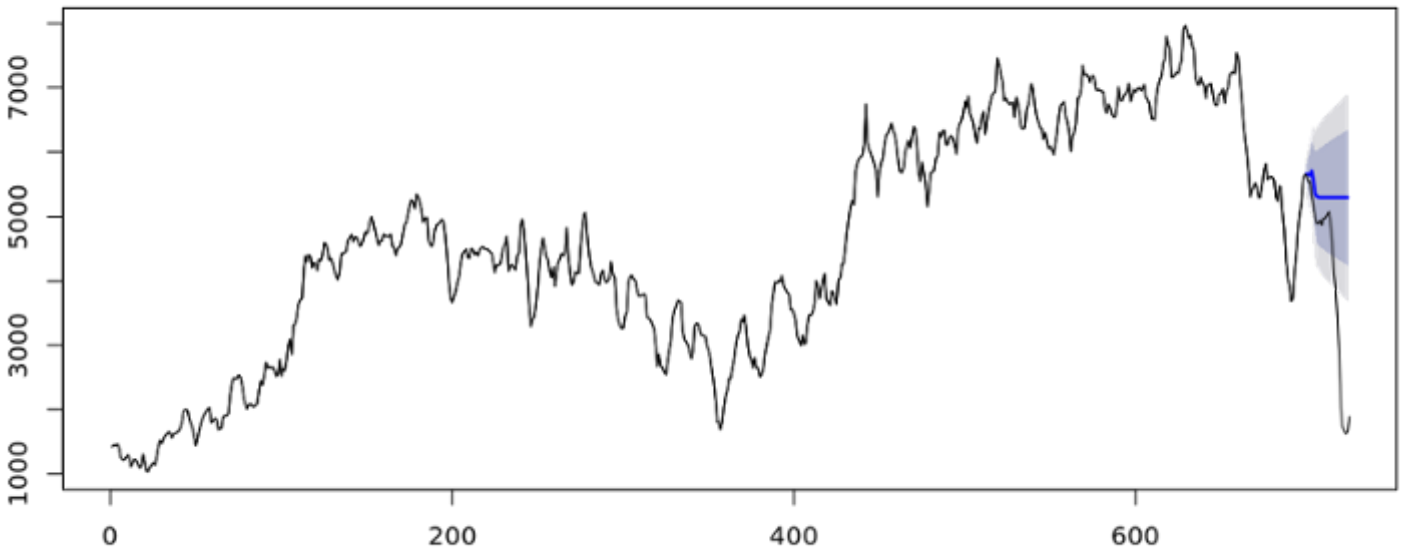
```
fcast <- forecast(fit2, h=30)
plot(fcast)
```

DATASCIENCE



The light blue line above shows the fit provided by the model, but what if we wanted to get a sense of how the model will perform in the future? One method is to reserve a portion of our data as a "hold-out" set, fit the model, and then compare the forecast to the actual observed values:

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

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

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



However, the blue line representing forecast seems very naive: It goes close to a straight line fairly soon, which seems unlikely given past behavior of the series. Recall that the model is assuming a series with no seasonality, and is differencing the original non-stationary data. In other words, plotted predictions are based on the assumption that there will be no other seasonal fluctuations in the data and the change in number of bicycles from one day to another is more or less constant in mean and variance. This forecast may be a naive model, but it illustrates the process of choosing an ARIMA model and could also serve as a benchmark to grade against as more complex models are built.

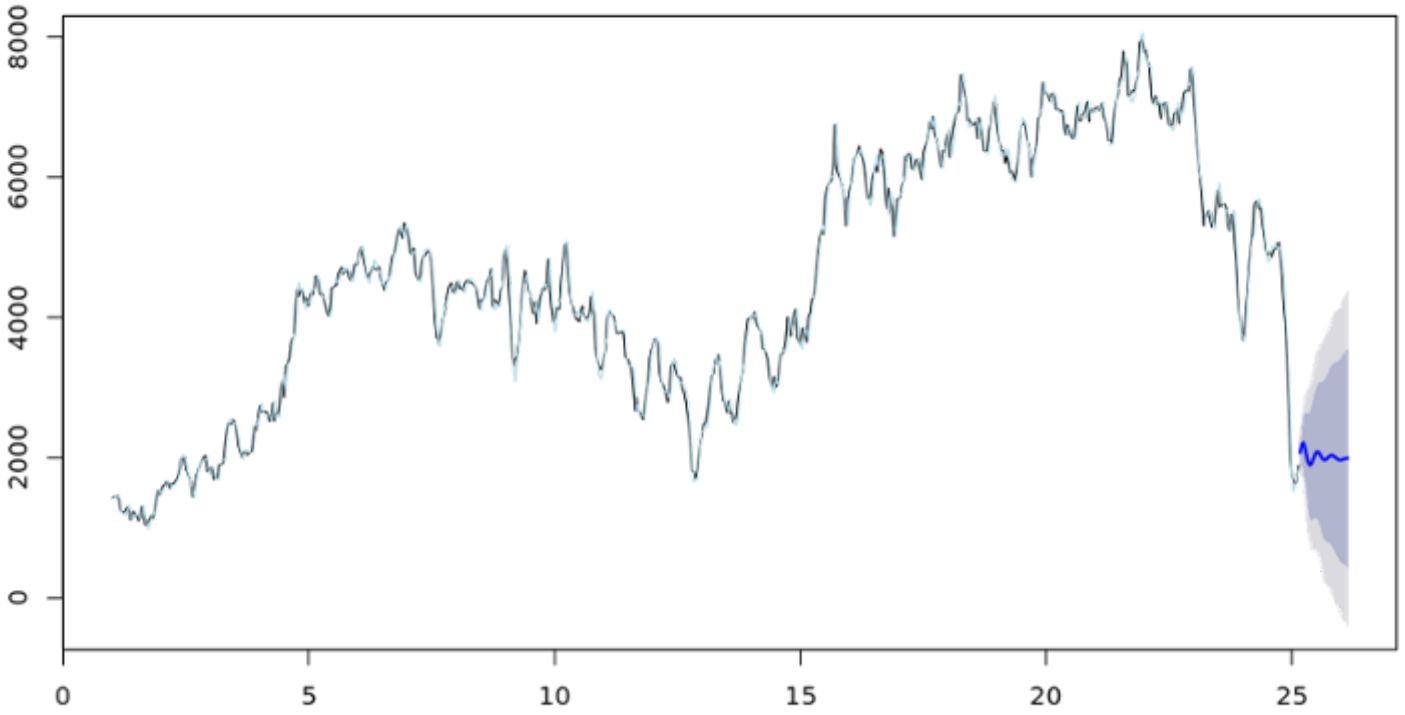
to be included to the model, which is a default in the **auto.arima()** function. Re-fitting the model on the same data, we see that there still might be some seasonal pattern in the series, with the seasonal component described by AR(1):

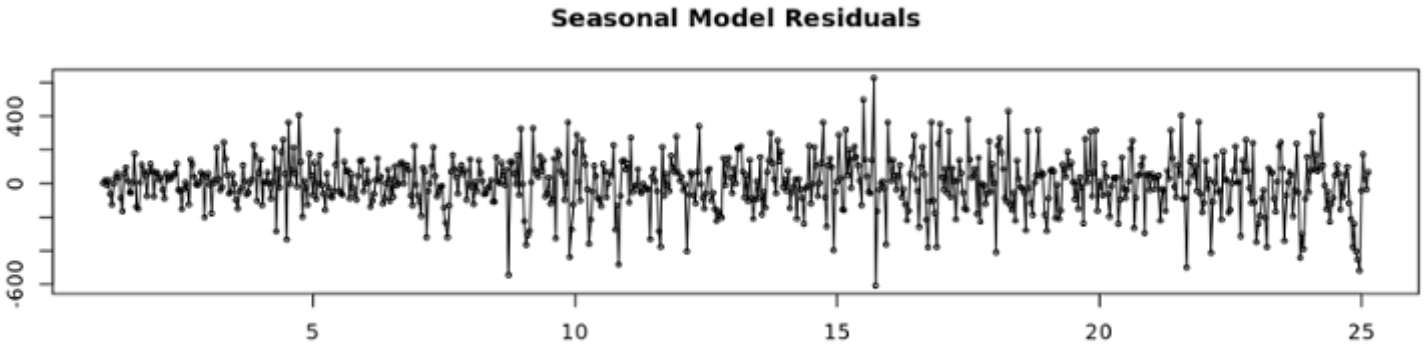
```
fit_w_seasonality = auto.arima(deseasonal_cnt, seasonal=TRUE)
fit_w_seasonality

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


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

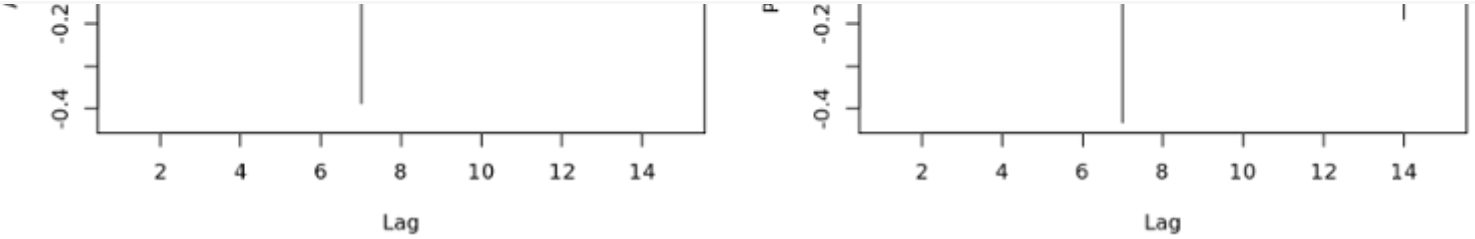
Note that (p,d,q) parameters also changed after we included a seasonal component. We can go through the same process of evaluating model residuals and ACF/PACF plots and adjusting the structure if necessary. For example, we notice the same evidence of higher order is present in the auto correlations with lag 7, which suggests that a higher-order component might be needed:

DATASCIENCE

DATASCIENCE







Both forecast estimates above are provided with confidence bounds: 80% confidence limits shaded in darker blue, and 95% in lighter blue. Longer term forecasts will usually have more uncertainty, as the model will regress future Y on previously predicted values. This is reflected in the shape of the confidence bounds in this case, as they start to widen with increasing horizon. The pattern in confidence bounds may signal the need for a more stable model. It is very important to look at prediction bounds and keep in mind the expected error associated with point estimates.

What's Next?

After an initial naive model is built, it's natural to wonder how to improve on it. Other forecasting techniques, such as exponential smoothing, would help make the model more accurate using a weighted combinations of seasonality, trend, and historical values to make predictions. In addition, daily bicycle demand is probably highly dependent on other factors, such weather, holidays, time of the day, etc. One could try fitting time series models that allow for inclusion of other predictors using methods such ARMAX or dynamic regression. These more complex models allow for control of other factors in predicting the time series.

To summarize, the procedure outlined in this tutorial is an introduction to ARIMA modeling. Material here can be used as a general guideline to examining your series, using ACF and PACF plots to choose model order, and fitting the model in R.

Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg, [http://link.springer.com/article/10.1007/s13748-013-0040-3].

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Products

Solutions

Resources

Education

Company