STQD6134: Business Analytics

Time Series Analysis: Employees' Assessment & Forecasting

(Part II)

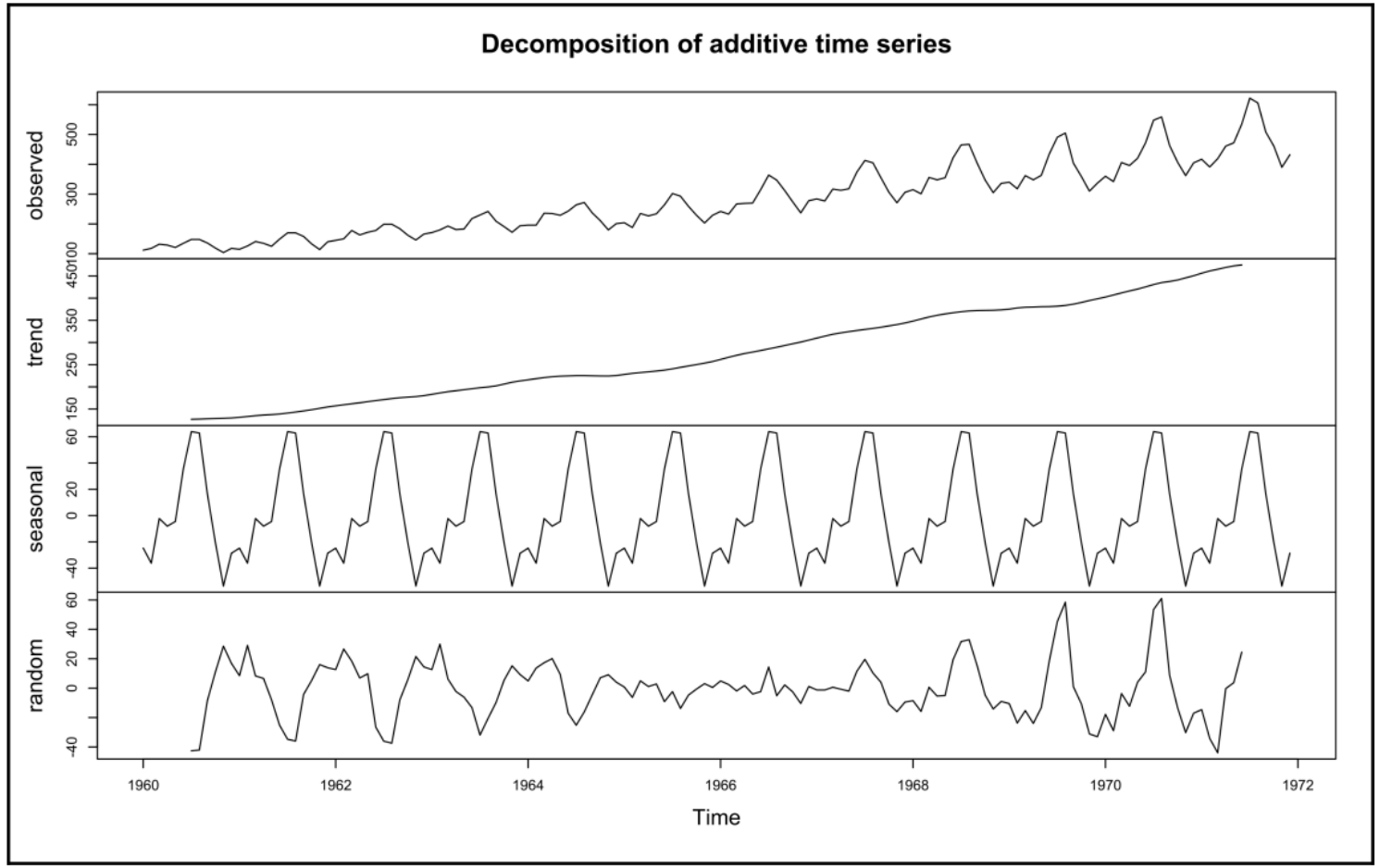
Key elements of time series analysis

You just applied a linear regression model to time series data and saw it did not work. The biggest problem was not a failure in fitting a linear model to the trend. For this well-behaved time series, the average formed a linear plot over time. *Where was the problem?*

The problem was in seasonal fluctuations. The seasonal fluctuations were one year in length and then repeated. Most of the data points existed above and below the fitted line, instead of on it or near it. As we saw, the ability to make a point estimate prediction was poor. There is an old adage that says *even a broken clock is correct twice a day*. This is a good analogy for analyzing seasonal time series data with linear regression. The fitted linear line will be a good predictor twice every cycle. You will need to do something about the seasonal fluctuations in order to make better forecasts; otherwise, they will simply be straight lines with no account of the seasonality.

With seasonality in mind, there are functions in R that can break apart the trend, seasonality, and random components of a time series. The `decompose()` function found in the `forecast` package shows how each of these three components influence the data. You can think of this technique as being similar to creating the correlogram plot during exploratory data analysis. It captures a greater understanding of the data in a single plot:

```
library(forecast)
plot(decompose(airpass))
```



- The top panel provides a view of the original data for context.
- The next panel shows the trend. It smooths the data and removes the seasonal component. In this case, you will see that over time, air passenger volume has increased steadily and in the same direction.
- The third plot shows the seasonal component. Removing the trend helps reveal any seasonal cycles. This data shows a regular and repeated seasonal cycle through the years.
- The final plot is the randomness-everything else in the data. It is like the error term in linear regression. You will see fewer errors in the middle of the series.

The stationary assumption

Stationary data exists when its mean and variance do not change as a function of time. If you decompose a time series and witness a trend, seasonal component, or both, then you have non-stationary data. You can transform them into stationary data in order to meet th required assumption.

Differencing technique

$$y\_diff\_trend = y_{(t+1)} - y_t$$

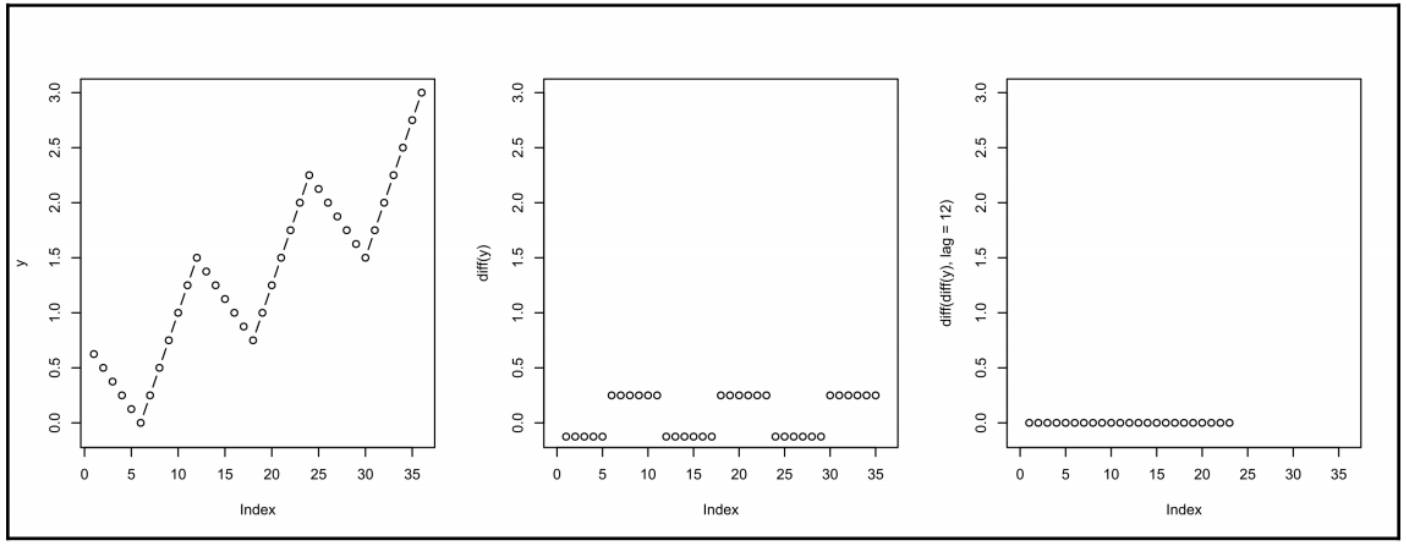
$$y\_diff\_seasonal = y_{(t+lag)} - y_t$$

Look at the results of differencing in this toy example. Build a small sample dataset of 36 data points that include an upward trend and seasonal component, as shown here:

```
seq_down <- seq(.625, .125, -0.125)
seq_up <- seq(0, 1.5, 0.25)
y <- c(seq_down, seq_up, seq_down + .75, seq_up + .75, seq_down + 1.5,
       seq_up + 1.5)
```

Then, plot the original data and the results obtained after calling the `diff()` function:

```
par(mfrow = c(1, 3))
plot(y, type = "b", ylim = c(-.1, 3))
plot(diff(y), ylim = c(-.1, 3), xlim = c(0, 36))
plot(diff(diff(y), lag = 12), ylim = c(-.1, 3), xlim = c(0, 36))
par(mfrow = c(1, 1))
detach(package:TSA, unload=TRUE)
```



These three panels are described as follows:

- The left panel shows n = 36 data points, with 12 points in each of the three cycles. It also shows a steadily increasing trend. Either of these characteristics breaks the stationary data assumption.

- The center panel shows the results of differencing. Plotting the difference between each point and its next neighbor removes the trend. Also, notice that you get one less data point. With differencing, you get $(n - 1)$ results.
- The right panel shows seasonal differencing with a lag of 12. The data is stationary. Notice that the trend differencing is now the data in the seasonal differencing. Also note that you will lose a cycle of data, getting $(n - lag)$ results.

Building ARIMA time series models

- **AR**: **Auto regressive**, specified with *p* or *P*
- **I**: **Integrated** (differencing), specified with *d* or *D*
- **MA**: **Moving average**, specified with *q* or *Q*

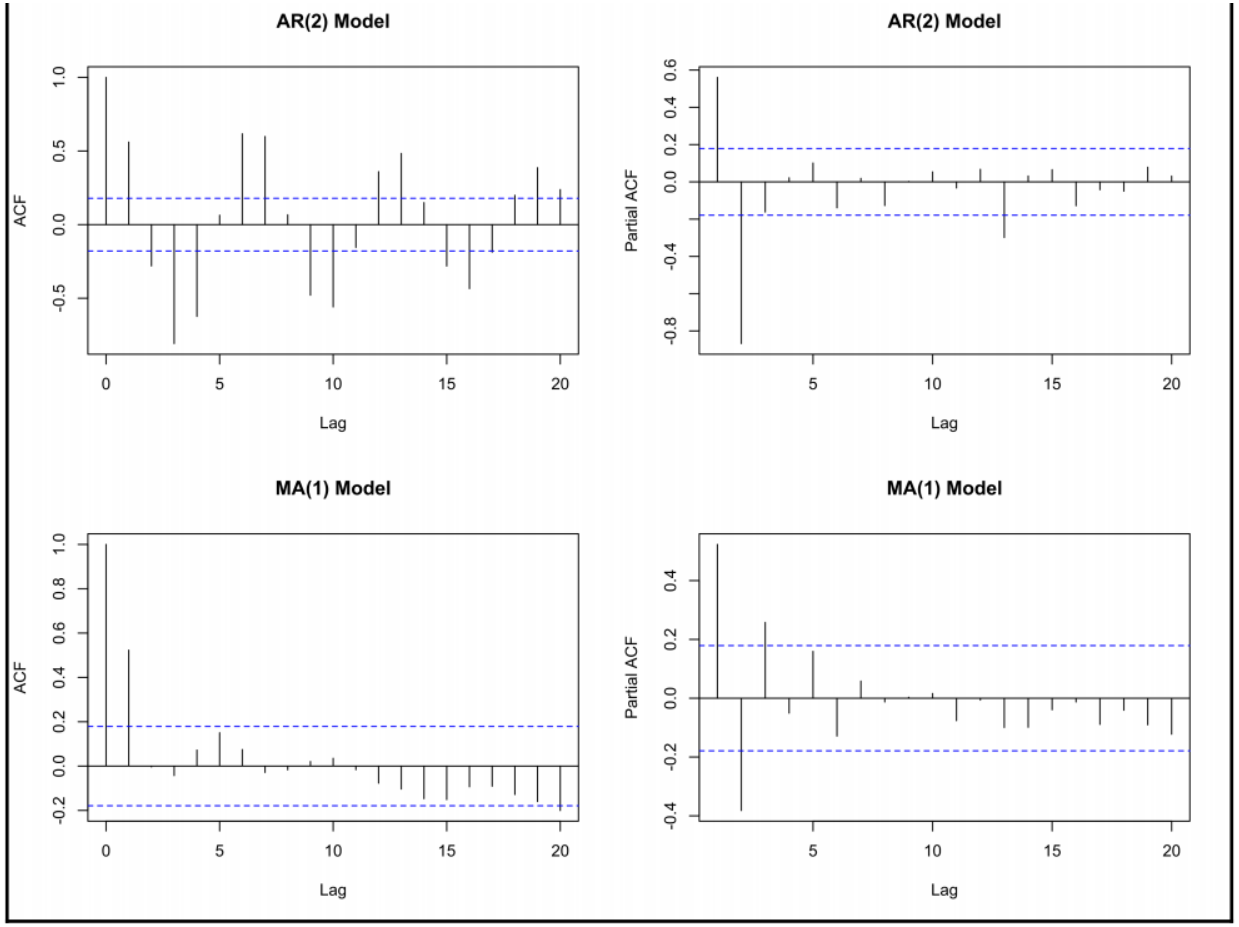
Auto regressive means that earlier lagged points in the data influence later points in the sequence. This creates a dependence condition. The type of AR model chosen is based on how many steps away (lags) the points in the past affect the points in the future. Data that has a greater lingering effect on future points has a higher lag. The higher the lag, the higher the AR number. You will see models referred to as *AR(1)*, *AR(2)*, and so forth to represent an autoregressive model of the number of *p* lags specified in the parentheses.

Integrated refers to differencing that you learned earlier. The *d* value represents the number of differences used in the model. It is typically zero or one.

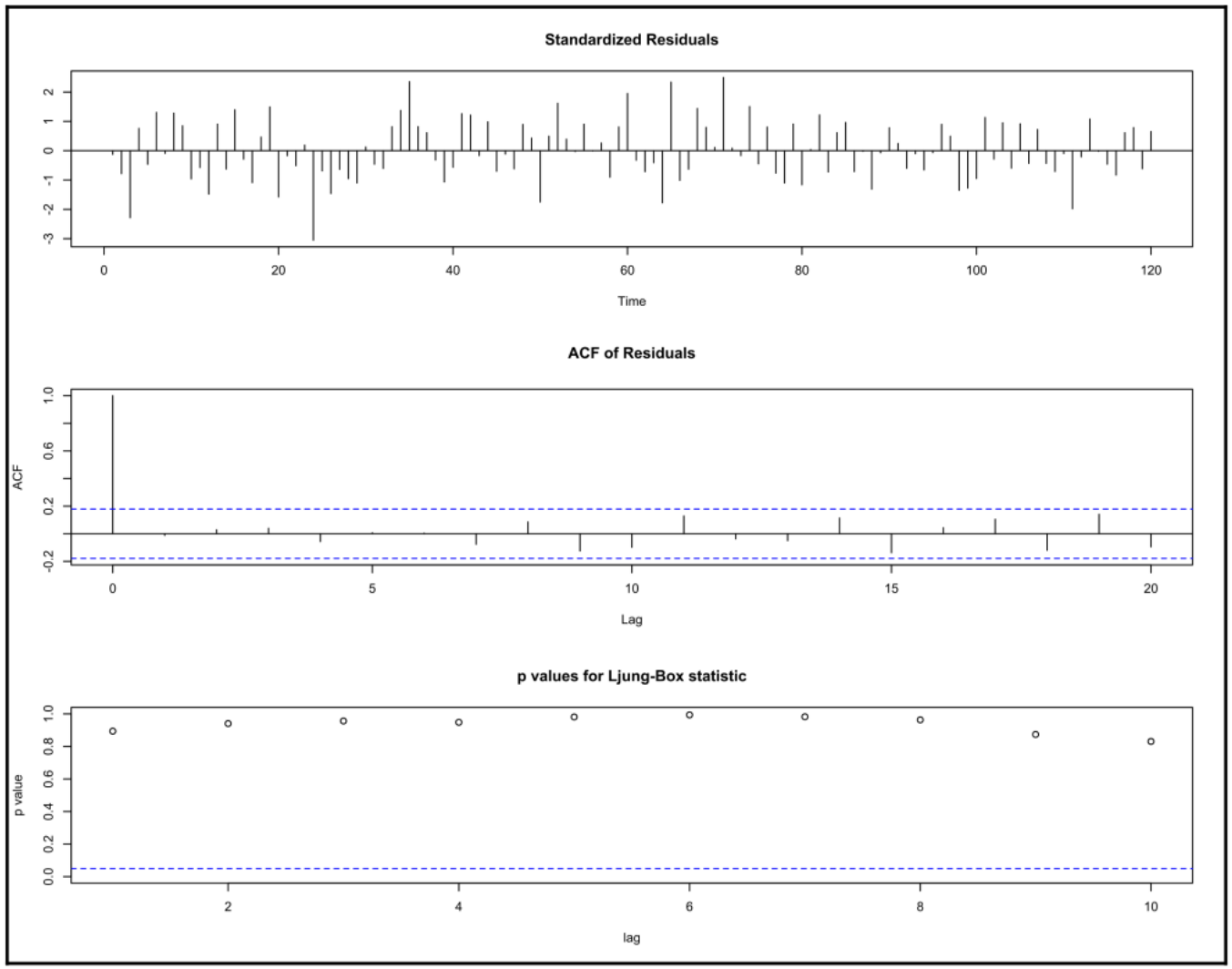
Moving average is the technique of calculating a future point based on the residuals (the error) of previous terms. You will see models referred to as *MA(1)*, *MA(2)*, and so forth to represent a moving average model with the number of *q* lagged forecast errors specified in the parentheses.

Model selection

- AR models will have an ACF that slowly diminishes or cycles, and its PACF will cut off under a significance line after a certain number of lags
- MA models will have a PACF that slowly diminishes or cycles, and its ACF will cut off under a significance line after a certain number of lags



One other useful tool is the plot generated by the `tsdiag()` function. You will apply this to an ARIMA model that already fits the data. It provides a view of standardized residuals (which should be random), an ACF plot (with a cut off at zero), and p-values (which should be above the critical point given as a dashed line). Here is a `tsdiag()` plot for the *AR(2)*:



Ridership use case data

1. Convert your data into a time series data type, adapting your data as needed.
2. Inspect the time series with the `decompose()` function.
3. If data is non-stationary, try differencing to see if it helps to make it stationary.
4. Determine the ARIMA model type and lags for the non-seasonal component.
5. Model different options and determine the best fit based on diagnostics.
6. Determine the ARIMA model type and lags for the seasonal component.
7. Model different options and determine the best fit based on diagnostics.
8. Generate a forecast for the desired number of periods into the future.

Begin by loading the data and use the `head()` function on the ridership data:

```
cycle <- read.csv("./data/Ch6_ridership_data_2011-2012.csv")
head(cycle)
```

We will get the following output:

```
              datetime count
1 2011-01-01 00:00:00    16
2 2011-01-01 01:00:00    40
3 2011-01-01 02:00:00    32
4 2011-01-01 03:00:00    13
5 2011-01-01 04:00:00     1
6 2011-01-01 05:00:00     1
```

The finance group asked you to aggregate the data on a monthly basis so that forecasts from the model provide monthly periods. The `dplyr` and `lubridate` packages are good tools to aggregate data. Using them reinforces skills you learned earlier in this book:

```
library(dplyr); library(lubridate)
monthly_ride <- as.data.frame(cycle %>%
          group_by(year = year(datetime), month = month(datetime)) %>%
          summarise(riders = sum(count)))
```

A quality check using `table(monthly_ride$year, monthly_ride$month)` shows that the aggregation resulted in a single value for each month in both years-a good output:

```
      1 2 3 4 5 6 7 8 9 10 11 12
2011 1 1 1 1 1 1 1 1 1  1  1  1
2012 1 1 1 1 1 1 1 1 1  1  1  1
```

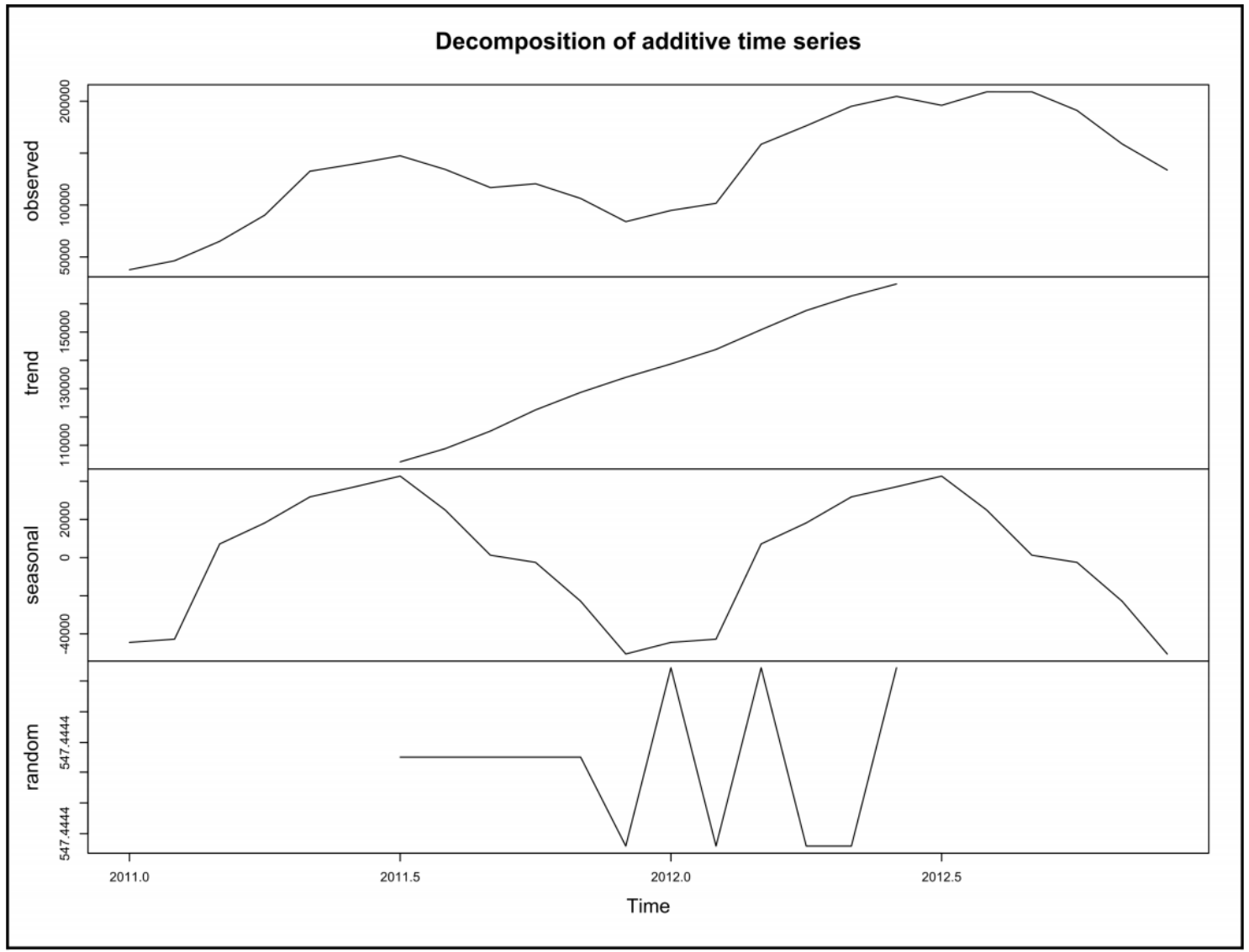
This data is still not a time series until you convert it. You have converted data before, for instance, from characters into time. The data is first subset to get a single column from the data frame. Then, you will create the time series using the `ts()` function:

```
riders <- monthly_ride[ ,3]
monthly <- ts(riders, frequency = 12, start = c(2011, 1))
```

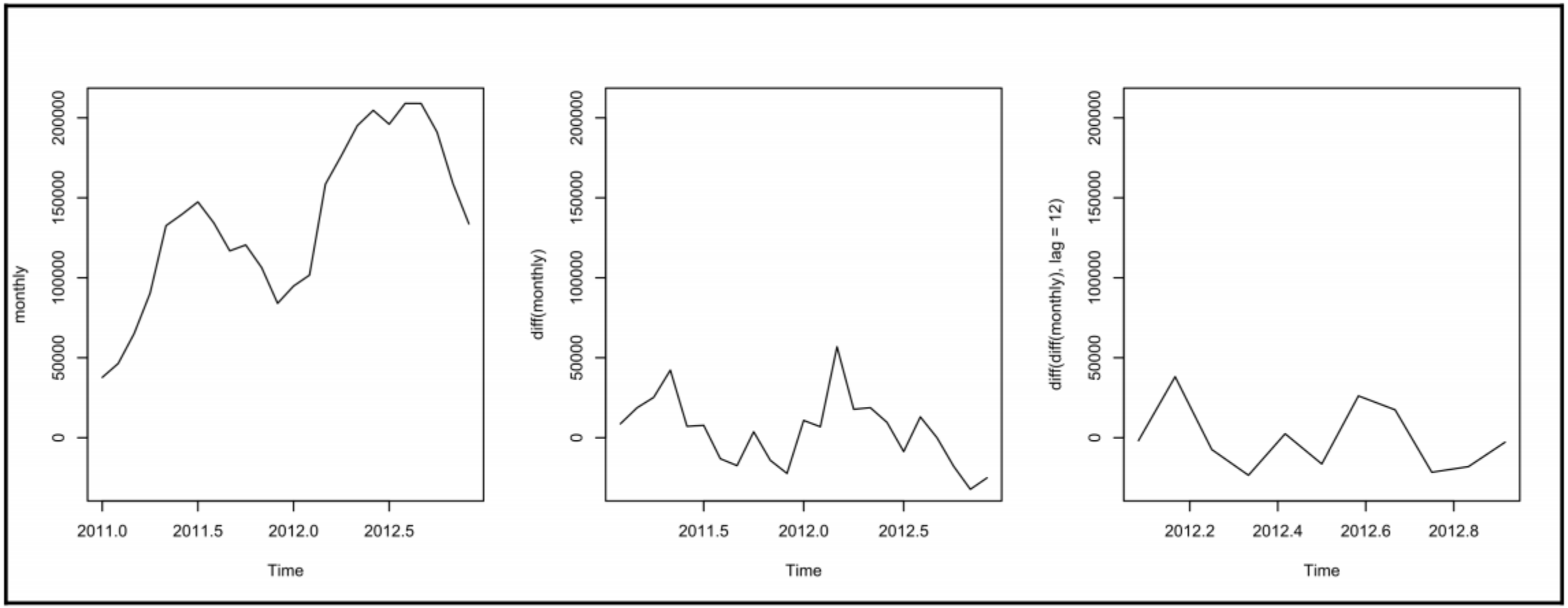
The `frequency` parameter is set to `12` as you have twelve data points (months) in every period (one year). The `start` parameter indicates that the first data point is the first month of 2011. View the time series to see that the data is loaded properly:

```
> monthly
      Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct    Nov    Dec
2011 37727  46396  65109  90332 132580 139674 147426 134280 116825 120535 106361  84025
2012 94832 101668 158535 176349 195114 204683 196014 209024 208995 191108 158855 133735
```

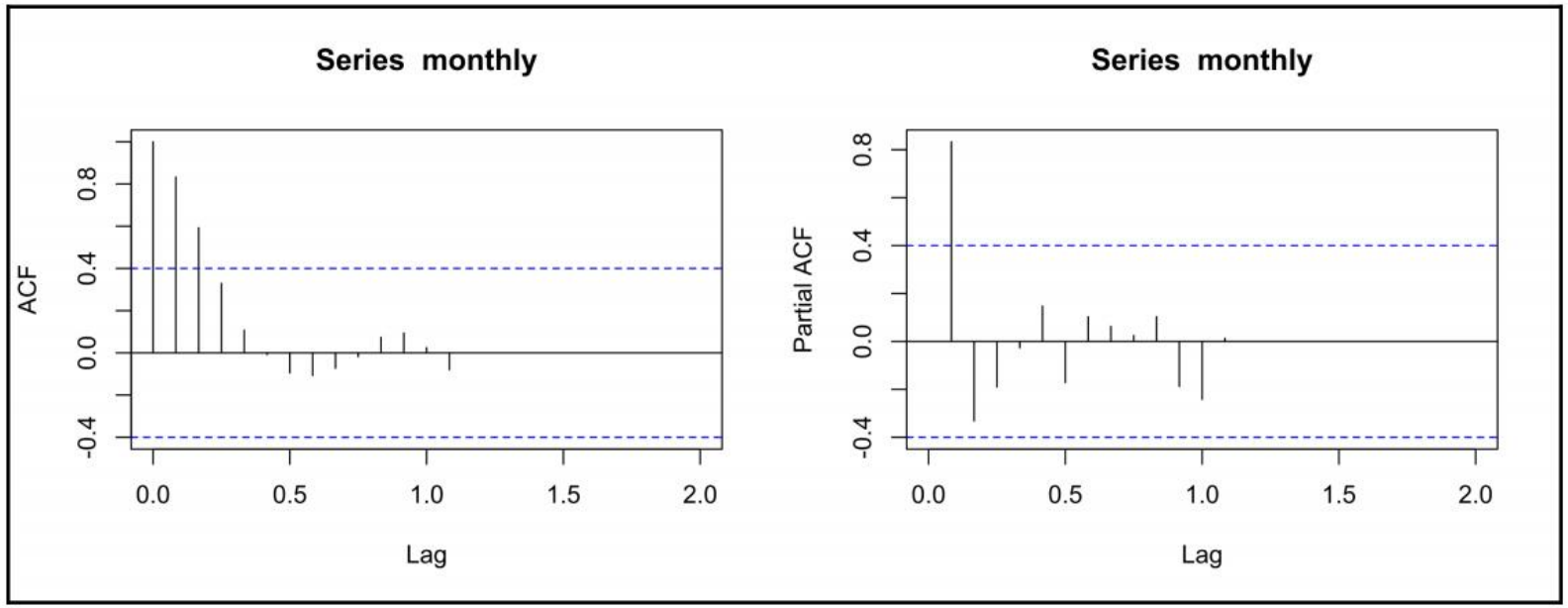
This finishes Step 1. Now, run `plot(decompose(monthly))` to inspect the time series. You will see the original data at the top, an increasing trend line, and some regular seasonality. The bottom panel shows leftover errors, as follows:



Step 3 is to run the differencing to see how to make the data stationary. Here are plots showing the original data, differencing, and seasonal differencing:



Moving to Step 4, generate ACF and PACF plots to get a sense of types and levels of models you may want to fit in this monthly data:



These plots look different from the earlier plots. There are very few lags. Yes, you are seeing them all, but there are only a few as the dataset contains only two years of data. Two data points from each cycle are typically the minimum amount of data to fit a time series model properly. Think about this, would you try to predict a linear model based on two data points? In time series, two complete periods only provide two observations for each part of the cycle. Both points better be perfect, otherwise your model will be very poor in forecasting the next point. We will continue, but be aware, you may see poor forecasting.

Step 5 is quite easy. Fit a number of models and then compare their results, including **Akaike Information Criterion** (**AIC**) (lower is better), and diagnostic plots. You may decide to create five models: two *AR(1)* models with and without differencing, one *AR(2)* with differencing, one *AR(1)* with differencing and seasonal differencing, and one *MA(1)* with differencing. You will fit ARIMA models using this general form, replacing the letters with values:

```
model_name <- arima(data, c(p, d, q),
                    seasonal = list(order = c(P, D, Q)))
```

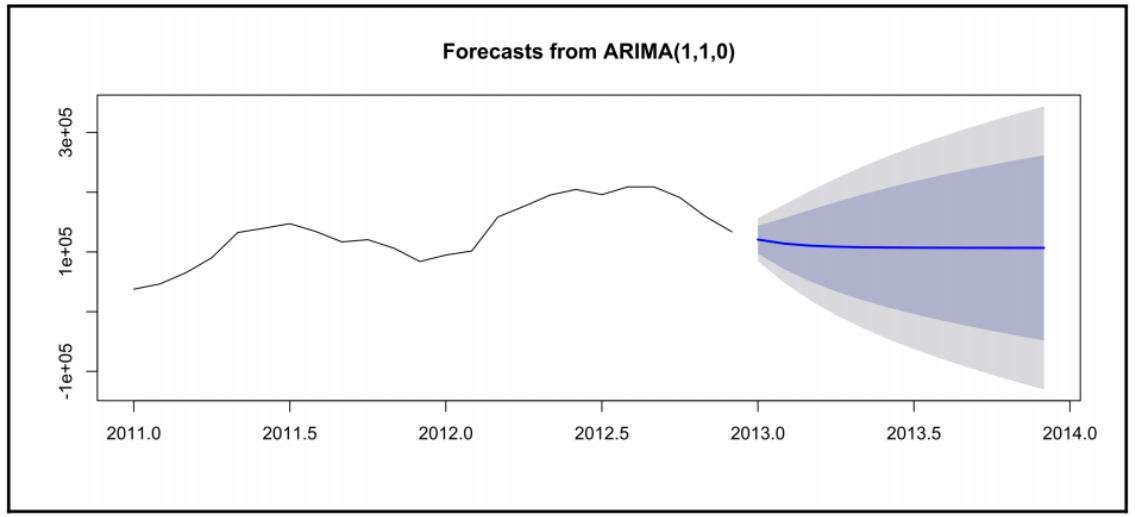
Here are the results of five different models to compare AIC and diagnostics:

Model	**AIC**	**Diagnostics**
ARIMA(1,0,0)(0,0,0)	553.74	Poor residuals and poor p-values
ARIMA(1,1,0)(0,0,0)	520.96	Good diagnostics
ARIMA(2,1,0)(0,0,0)	522.63	Good diagnostics
ARIMA(1,1,0)(0,1,0)	252.38	Poor p-values
ARIMA(0,1,1)(0,0,0)	523.30	Poor p-values

The second model is *AR(1)* with differencing. It is simpler than *AR(2)* and provides slightly better AIC. Now, you will run the `forecast()` function found in the `forecast` package to use the fitted *AR(1)* model and `h = 12` for the next 12 data points (months):

```
library(forecast)
yr_forecast <- forecast(fit2, h = 12)
plot(yr_forecast)
```

The output we get is shown in the following diagram:



The blue line in the center represents the mean forecast. The dark inner cone represents the 80% interval. The light outer cone represents the 90% forecast interval. Not a pretty forecast. It communicates the message that *anything is possible: growth, loss, or flat year*. This is too general to be of much use to the business. The trouble with this forecast is that the methods do not work on small trends such as this. Advanced techniques can help.

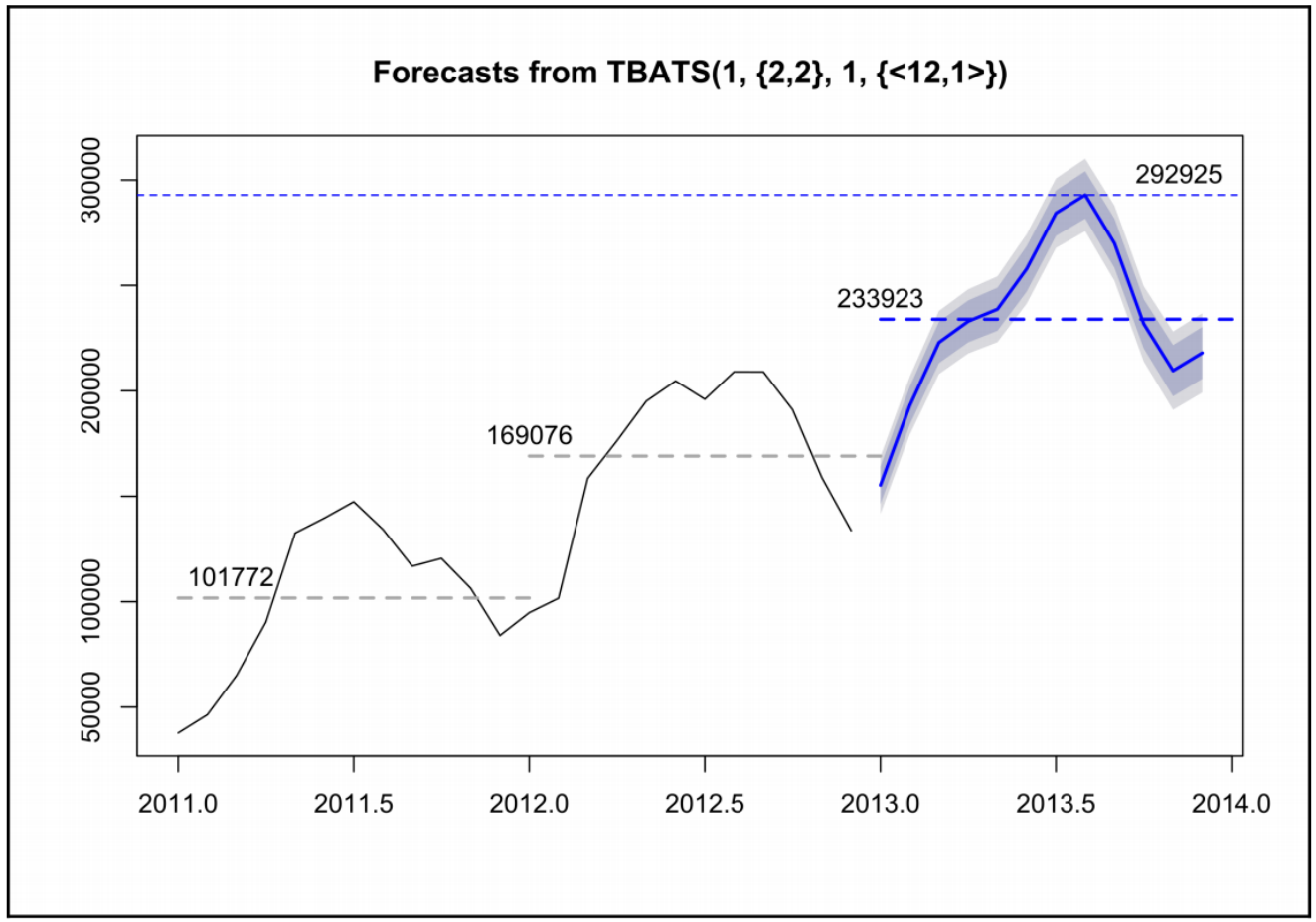
Advanced functionality for time series modelling

Advanced functions exist that work with smaller data sets such as your ridership data. The `tbats` package performs high-level mathematics, such as Fourier transforms, to fit models with smaller data. This technique also works well when your `frequency` parameter exceeds 24 (used for hourly aggregations). The `arima()` function cannot work with frequencies such as 52, which you may want if your model is based on weekly aggregations over a year. To run the advanced function, you call the `tbats()` function on the data:

```
monthly_data <- tbats(monthly)
```

Yes, it is very simple to run the `tbats()` function. You now have a fitted model. Use it to create a forecast for the next year:

```
year_forecast <- forecast(monthly_data, h = 12)
plot(year_forecast)
```



R tip: Do not allow the simplicity of this technique to fool you. Be respectful of its power and use it wisely while making forecasts. This technique appears in this book only after investigating a number of methods. TBATS worked with this particular dataset, but it may not work well with other short time series.

There are many other advanced techniques. You can read about them in Rob Hyndman's blog. He does a very good job at explaining R techniques for time series analysis in understandable terms. You can find many approaches at `http://robjhyndman.com/hyndsight/`.

The forecast in the plot looks reasonable. The middle parameters in the title, **{2,2}**, indicate this has an *AR(2)* and *MA(2)* component. The other parameters are specific to the method. The forecast shows a trend and seasonality comparable to the data you have. From the business perspective, you can expect questions about this forecast. These may include:

- What is the maximum ridership level for next year?
- What is the average ridership level for next year?
- How do both values compare to the historical data?

This plot anticipated these questions and added information to the basic plot. Like other modeling objects, you can extract this information from the fitted model. Begin with summaries to get an idea of the available information.

First, get a summary of the average:

```
summary(year_forecast$mean)
```

The output is as follows:

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 155300  215900  232300  233900  260900  292900
```

Then, summarize the upper end of the forecast interval:

```
summary(year_forecast$upper)
```

The following is the output:

```
       V1              V2
 Min.   :164108   Min.   :168788
 1st Qu.:228134   1st Qu.:234617
 Median :243134   Median :248848
 Mean   :244606   Mean   :250262
 3rd Qu.:271888   3rd Qu.:277705
 Max.   :304163   Max.   :310112
```

Lastly, take a look at the lower end of the forecast interval:

```
summary(year_forecast$lower)
```

The output is as shown here:

```
       V1                V2
Min.   :146425   Min.   :141745
1st Qu.:203639   1st Qu.:197155
Median :221544   Median :215830
Mean   :223240   Mean   :217585
3rd Qu.:249914   3rd Qu.:244098
Max.   :281688   Max.   :275739
```

These summaries show you the data available to extract in order to accent the plot. The following is the code snippet used to create the annotations. Begin by getting the yearly means and maximum ridership:

```
mean_2011 <- round(as.numeric(filter(monthly_ride, year == 2011) %>%
             summarise(mean = mean(riders))), 0)
mean_2012 <- round(as.numeric(filter(monthly_ride, year == 2012) %>%
             summarise(mean = mean(riders))), 0)
mean_2013 <- round(mean(year_forecast$mean), 0)
max_mean_2013 <- round(max(year_forecast$mean), 0)
```

Now use the variables to create a top line representing the maximum of the mean forecast, as well as segments that span across only the individual years to show the mean for each year:

```
abline(h = max(year_forecast$mean), lty = 2, col = "blue")
segments(2011, mean_2011, x1 = 2012, y1 = mean_2011,
         col = "darkgray", lty = 2, lwd = 2)
segments(2012, mean_2012, x1 = 2013, y1 = mean_2012,
         col = "darkgray", lty = 2, lwd = 2)
segments(2013, mean_2013, x1 = 2014, y1 = mean_2013,
         col = "blue", lty = 2, lwd = 2)
```

Finally, label each of the lines and segments with the mean values that you can extract from the fitted model. You use the `text()` function and pass it the `x` and `y` coordinates, followed by the text you would like on the plot. In this case, you will also pass the text as a variable. Adding `10000` to the `y` coordinate lifts it off the line for readability:

```
text(2011.15, mean_2011 + 10000, mean_2011)
text(2012, mean_2012 + 10000, mean_2012)
text(2013, mean_2013 + 10000, mean_2013)
text(2013.85, max_mean_2013 + 10000, max_mean_2013)
```

Things are looking good at Bike Sharing LLC. The company can anticipate a solid year with over 35% growth in average ridership, and the peak monthly ridership could grow 40% over last year.

Assignment 2 – due 17th Dec 2020

- Produce the output in Slide 19 for the five chosen models. Show how the diagnostics in this slide are concluded based on the outputs.