TimeSeries Analysis - Trigonometric and Binary Modeling

writer : Joohyoung Jeon (jeonjoohyoung@gmail.com)

전체 소스 코드 : https://github.com/Joohyoung/R-usages/blob/master/TimeSeriesAnalysisCase1/TutorialCode.R

1. 분석 목적

시계열 데이터 중 Seasonal Variational을 갖는 데이터를 찾고 해당 데이터에 대한 기초 통계 분석, Trigonometric Modeling, Binary Modeling을 수행하고 각 모델에 대한 예측력을 평가한다

2. 기초통계 분석

2.1 데이터 개요

본 데이터는 '구글 트렌드'(https://www.google.com/trends/?hl=ko) 에서 검색어 '비염(rhinitis)'에 대한 주차별 검색횟수 합을 나타낸 그래프이다. 데이터 기간은 2015년 1월부터 2019년 9월까지이며 관측치(Observations)는 총 247개(247주) 이다. 분석은 R에서 수행하였다.

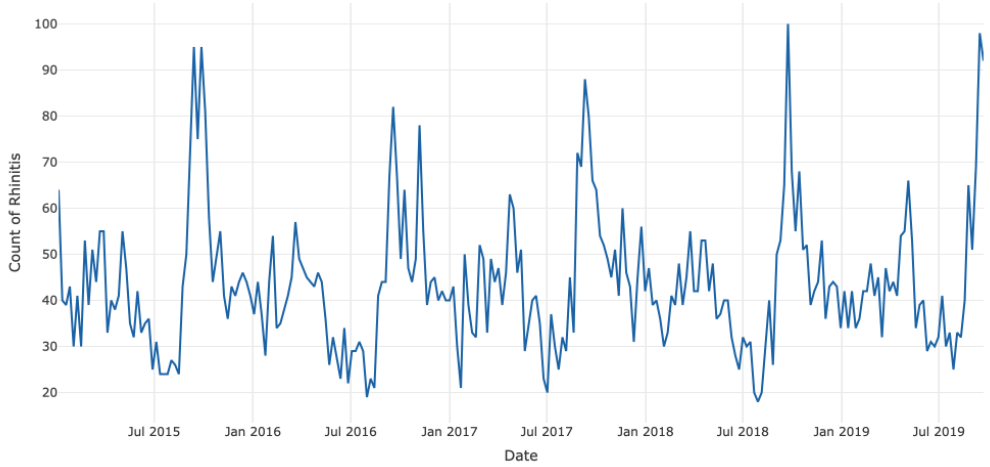
2.2 데이터 적재 및 시각화

- **데이터 적재**

```
# 분석에 필요한 패키지를 설치한다.
install.packages(c("dplyr","plotly", "anytime", "lubridate", "dummy", "lmtest"))
# 1. dplyr : 데이터 집계를 위한 패키지
# 2. plotly : 시각화 툴
# 3. anytime :Character 형태의 Date 변수를 Date 유형의 변수로 전환해주는 패키지. 일자 관련 처리할 수 있는 다양한 패키지가 존재한다.
# 4. lubridate : Date변수에서 월, 주차, 년주차 등 일자정보를 추출하는 패키지
# 5, dummy : One-Hot Encoding 변수를 만들어 주는 패키지
# 6. lmtest : Durbin-Watson 검증을 위한 패키지

# 패키지 로드
require(dplyr)
require(plotly)
require(anytime)
require(lubridate)
require(dummy)
require(lmtest)

# 작업환경 설정
setwd(paste(getwd(), 'TimeSeriesAnalysisCase1', sep = '/'))

# 데이터셋을 읽는다. Header가 존재하며, CSV파일은 ',' 을 기준으로 Column 을 구분하고 있다.
# colClass를 사전에 선언함으로써 각 Column별 데이터 Type을 사전에 지정 가능하다.
data <- read.csv('search_rhinitis.csv',
                 header = T, sep = ',', strip.white = T, colClasses = c('character', 'numeric'))
```

```
# 위에서 부터 10줄을 확인한다.
# 다음과 같이 위에서 부터 데이터를 확인한다. time은 집계된 날짜, x는 검색 횟수 집계이다.
> head(data, 10)
         time  x
1  2015-01-04 64
2  2015-01-11 40
3  2015-01-18 39
4  2015-01-25 43
5  2015-02-01 30
6  2015-02-08 41
7  2015-02-15 30
8  2015-02-22 53
9  2015-03-01 39
10 2015-03-08 51
```

- **데이터 시각화**



시각화 결과는 위의 그래프와 같다.

'비염'은 환절기에 자주 발생하는 질환 중 하나이다. 5년간의 시계열 자료를 살펴 보았을 때, 검색어가 증가하는 시점은 9월과 3월에 집중되어 있다. 해당 반복성이 2년 이상을 띄고 있으므로 우리는 본 데이터에 Seasonal Variational을 갖는다고 추정 할 수 있다.

```
# 시각화 plot 출력 코드
plotly::plot_ly( x = ~ anytime::anydate(data$time),
                 y = ~data$x, mode = 'lines') %>%
  plotly::layout(xaxis = list(title = 'Date'),
                 yaxis = list(title = 'Count of Rhinitis'))
```

- plotly는 다양한 그래프를 출력 하는데 사용 되는 매우 유용한 library 이다. python, js, r 을 지원한다.
  참조 : https://plot.ly/graphing-libraries/

2.3 기초통계량 분석

- **전체 데이터 셋에 대한 기초통계값 확인**

summary 명령어를 통해 독립변수 X의 기초통계를 확인한다. 최소, 최대값, 1, 3분위수, 중위수, 평균을 확인 할 수 있다.

최소, 최대값은 말 그대로 데이터 중에서 가장 작은 값과 가장 큰 값을 말한다. 1분위, 중위, 3분위수는 데이터를 낮은 값에서 높은 값까지 순서대로 정렬하였을 때 각각의 위치값을 말한다.

데이터를 4등분 한다고 가정하자. 4개의 조각을 얻기 위해서는 3개를 기점으로 나눠야 한다. 즉 ——|——|——|—— 이런 식이다. 이 때 각각의 세로선 (등분하는 선)을 분위수라 말한다. ——1분위수——중위수——3분위수—— 가 된다. 이를 위해서는 데이터를 있는 그대로 자르는 것이 아니라, 낮은 순서에서 높은 순서대로 정렬해야 한다. 중위수를 기준으로 좌측 데이터는 항상 우측 데이터들보다 낮은 값으로 구성된다. 이처럼 데이터를 조각내는 것을 분위수라 말한다. 분위수 개념은 나중에 데이터 특성을 파악하여 분류하는 클러스터링 방법에도 활용될 수 있기 때문에 잘 알아 두는 것이 좋다.

```
> summary(data)
     time                x
 Length:247         Min.   : 18.00
 Class :character   1st Qu.: 33.00
 Mode  :character   Median : 42.00
                    Mean   : 43.49
                    3rd Qu.: 49.00
                    Max.   :100.00
> dim(data) # Num of Observations 247, Feature 2 (Time, X)
[1] 247   2
```

- **월별 기초통계량 확인**

```
# 다음과 같이 집계(Group_by) 기준을 Month로 하게 되면, Date변수인 Time에서 매월 기준으로 데이터를 집계한다.
data %>% dplyr::group_by(Month = lubridate::floor_date(time, "1 month")) %>%
  dplyr::summarise(mean_x = mean(x), sd_x = sd(x),
                   q1_x = quantile(x, 0.25),
                   median_x = median(x),
                   q3_x = quantile(x, 0.75)) %>% print(n=100)
```

```
    Month      mean_x  sd_x  q1_x median_x  q3_x
   <date>      <dbl> <dbl> <dbl>    <dbl> <dbl>
 1 2015-01-01   46.5 11.8   39.8     41.5  48.2
 2 2015-02-01   38.5 11.0   30       35.5  44
 3 2015-03-01   48.8  7.09  44       51    55
 4 2015-04-01   38    3.56  36.8     39    40.2
 5 2015-05-01   42.2  9.26  35       42    47
 6 2015-06-01   32.2  4.99  31       34    35.2
 7 2015-07-01   25.8  3.5   24       24    25.8
 8 2015-08-01   34   11.7   26       27    43
 9 2015-09-01   84.8 11.8   74.8     85    95
10 2015-10-01   58.2 16.2   48.5     54    63.8
11 2015-11-01   43.2  7.09  41       41    43
12 2015-12-01   43.8  2.06  43.2     44    44.5
13 2016-01-01   38    6.60  37       37    44
14 2016-02-01   40.2  9.32  34.8     36.5  42
15 2016-03-01   48    6.83  44       47    51
16 2016-04-01   44.8  1.71  43.8     44.5  45.5
17 2016-05-01   36.8  8.32  32       36    44
18 2016-06-01   26.5  5.45  22.8     25    28.8
19 2016-07-01   27.4  4.77  29       29    29
20 2016-08-01   32.2 11.9   22.5     32    41.8
21 2016-09-01   65   15.7   61.2     67    70.8
22 2016-10-01   50.6  7.77  47       49    49
23 2016-11-01   54   17.3   42.8     49.5  60.8
24 2016-12-01   41.8  2.36  40       41    42.8
.... 중략 ....
```

데이터를 확인해 보면 9월과 3월의 평균값이 다른 월에 비해 비교적 높은 것을 확인할 수 있다.

2.4 데이터의 자기상관성 분석

자기상관성(autocorrelation)이란 데이터 자기 자신과 N 시점 뒤로 미룬(lag) 자기 자신과 데이터와의 상관성을 말한다. 본 과정에서는 Durbin-Watson 검정을 통해 오차항의 자기상관성을 검정 해보도록 한다. 본 검정법은 오직 1차수의 자기상관성만 검정할 수 있는 단점이 있다. 만약, 1차수 이상의 자기상관성을 확인해보기 위해서는 편자기상관계수(PACF : Partial Autocorrelation Function)과 자기상관함수(ACF : AutoCorrelation Function)을 통해 확인할 수 있으나, 본 방법은 추수 수행하도록 하고 본 분석에서는 Durbin-Watson 검정만 수행 한다.

- **Durbin-Watson 테스트**

Durbin-Watson테스트에서 종속변수(y)는 검색횟수로 독립변수(x)는 시간으로 설정한다.

이를 위해서는 기존 time 변수를 Date타입이 아닌 Sequence 데이터로 바꿔 주어야 한다. 기존 time 변수는 Date 타입으로 Formula에 넣으면 해석되지 않기 때문이다. Date 타입 변수에 대한 차(diff)를 구할 수 없기 때문이다. 따라서, 시간 변수를 1, 2, 3, ... N (# of observations) 으로 새롭게 생성한다.

```
# dim(data)[1] = Number of Obersvations
# Sequence형태의 Time Variable을 새롭게 만든다.
data$seq_time <- seq(1, dim(data)[1])
# [1]   1   2   3   4   5   6   7   8   9  10 ....... 246 247

# Durbin-Watson 통계량을 확인하는데, 이때 Formula는 회귀모형 객체(lm)이 그대로 들어갈 수도 있다.
# lm(formula = x ~ seq_time)는 least square 방식으로 독립변수가 시간, 종속변수가 검색횟수인 선형회귀모델을 생성한다.
lmtest::dwtest(lm(formula = x ~ seq_time, data = data), alternative = 'two.sided')
```

```
          Durbin-Watson test

data:  lm(formula = x ~ seq_time, data = data)
DW = 0.5837, p-value < 2.2e-16
alternative hypothesis: true autocorrelation is not 0
```

Durbin-Watson 테스트 결과, DW가 0.5837 이고 P-value가 매우 낮은 값이므로 귀무가설(H0 : 잔차들 사이에 자기상관관계가 없다)를 기각하고 대립가설(H1 : 잔차가 자기상관관계가 있다) 를 채택하게 된다. (일반적으로 DW값이 2 근처의 값이 나와야 자기상관관계가 없다고 볼 수 있다.)

즉, 본 데이터는 자기상관성이 존재한다.

3. 모델링

주어진 데이터에 대하여 Trigonometric, Binary 모델링을 수행한다. 본 모델링의 핵심은 시간 변수(t)를 다양한 방법으로 가공하는 것이다.

3.1 Trigonometic Modeling

$$Model_1 : y_t = \beta_0 + \beta_1 t + \beta_2 \sin(\frac{2\pi t}{L}) + \beta_3 \cos(\frac{2\pi t}{L}) + \epsilon_t$$

$$Model_2 : y_t = \beta_0 + \beta_1 t + \beta_2 \sin(\frac{2\pi t}{L}) + \beta_3 \cos(\frac{2\pi t}{L}) + \beta_4 \sin(\frac{4\pi t}{L}) + \beta_5 \cos(\frac{4\pi t}{L}) + \epsilon_t$$

위의 수식을 보면 종속변수(y)는 검색횟수를 의미하고, 독립변수는 t 하나이다. 시간변수인 t에 대하여 삼각함수(sin, cos)의 값을 대입하여 새로운 유형의 변수들을 생성한다. 또한 Model Parameter로서 L값을 부여하게 된다.

이러한 수식으로 데이터를 변환 시켜주는 작업이 필요하다. 다음 코드는 Trigonometric modeling을 위해 데이터를 Trigometric 모델에 맞게 변환시켜 주는 작업을 하는 함수이다.

```
# 본 함수는 관측치의 갯수와 모델 파라메터 L, Model Type 1 or 2 가 주어지면 해당 조건에 맞는 trigonometric data를 생성한다.
transform_trigonometric_data <- function(observations, L, model_type = '1'){
  time <- seq(1, observations)

  common_value <- (2*base::pi*time)/L
  var_1 <- sin(common_value)
  var_2 <- cos(common_value)

  data = data.frame(time, var_1, var_2)

  if(model_type == '2'){
    var_3 <- sin(common_value*2)
    var_4 <- cos(common_value*2)
    data$var_3 <- var_3
    data$var_4 <- var_4
  }
  return(data)
}

# trigonometric data는 다음과 같이 생성한다. 종속변수(클릭횟수)를 log를 취해준다.
trigonometic_data <- data.frame(y=log(data$x), transform_trigonometric_data(dim(data)[1], L = 4, '1'))
```

종속변수에 자연로그를 취해주었다. (r에서 log함수의 기본 base로 exp(1) 값이 주어진다)

- **모델링 데이터 확인 (model 1)**

```
> trigonometic_data <- data.frame(y=log(data$x), transform_trigonometric_data(dim(data)[1], L = 12, '1'))
> head(trigonometic_data)
   y time        var_1         var_2
1 4.158883    1 5.000000e-01  8.660254e-01
2 3.688879    2 8.660254e-01  5.000000e-01
3 3.663562    3 1.000000e+00  6.123234e-17
4 3.761200    4 8.660254e-01 -5.000000e-01
5 3.401197    5 5.000000e-01 -8.660254e-01
6 3.713572    6 1.224647e-16 -1.000000e+00
```

- **모델링 데이터 확인 (model2)**

```
> trigonometic_data <- data.frame(y=log(data$x), transform_trigonometric_data(dim(data)[1], L = 12, '2'))
> head(trigonometic_data)
```

```
   y time        var_1         var_2         var_3 var_4
1 4.158883    1 5.000000e-01  8.660254e-01  8.660254e-01   0.5
2 3.688879    2 8.660254e-01  5.000000e-01  8.660254e-01  -0.5
3 3.663562    3 1.000000e+00  6.123234e-17  1.224647e-16  -1.0
4 3.761200    4 8.660254e-01 -5.000000e-01 -8.660254e-01  -0.5
5 3.401197    5 5.000000e-01 -8.660254e-01 -8.660254e-01   0.5
6 3.713572    6 1.224647e-16 -1.000000e+00 -2.449294e-16   1.0
```

- **데이터셋 분할 (90% train, 10% valid) 및 모델 생성**

전체 데이터 중 90%를 Train 데이터로 활용하고, 10%를 Validation으로 활용한다.

Trigonometic의 parameters로 L은 12로 주어졌다. 모델 1, 2에 대하여 각각 모델을 생성하고, 예측결과를 검증한다.

```
# data split
# 1 to 222 (90% of Obersvations) -> train
# 223 to remainer (10% of Observations) -> valid
train_idx <- seq(1, round(dim(trigonometic_data)[1]*0.9))
valid_idx <- seq(round(dim(trigonometic_data)[1]*0.9)+1, dim(trigonometic_data)[1])

train_trigono <- trigonometic_data[train_idx,]
valid_trigono <- trigonometic_data[valid_idx,]

# 모델 정보를 가져온다.
summary(lm(formula = y ~ ., data = train_trigono))
```

Trigonometric Model 1 Result

```
> summary(trigono_model1)

Call:
lm(formula = y ~ ., data = train_trigono)

Residuals:
     Min       1Q   Median       3Q      Max
-0.80221 -0.18620  0.02574  0.17412  0.84890

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.7038410  0.0426303  86.883   <2e-16 ***
time        0.0001001  0.0003315   0.302    0.763
var_1       0.0329015  0.0300486   1.095    0.275
var_2       0.0156270  0.0300408   0.520    0.603
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3165 on 218 degrees of freedom
Multiple R-squared:  0.007108,  Adjusted R-squared:  -0.006556
F-statistic: 0.5202 on 3 and 218 DF,  p-value: 0.6688
```

최소제곱법(Least Square) 방식으로 위의 Trigonometric Model1에 대하여 계수를 추정한 값이다. 절편을 제외한 나머지 계수에 대한 유효성은 존재하지 않다. 또한, R Sqaure 값이 매우 낮으므로 해당 모델이 본 데이터에 대하여 적합하지 않은 것으로 판단된다. 위의 결과는 다음과 같은 식으로 표현 가능하다.

$$Model_1 : \log(y_t) = 3.7038410 + 0.001001 * t + 0.0329015 \sin(\frac{2\pi t}{L}) + 0.0156270 \cos(\frac{2\pi t}{L}) + \epsilon_t$$

위에서 생성한 모델을 **trigono_model1** 이라는 변수에 저장하였다.

Trigonometric Model 2 Result

```
Call:
lm(formula = y ~ ., data = train_trigono)

Residuals:
     Min       1Q   Median       3Q      Max
-0.81710 -0.18362  0.02642  0.17632  0.84422
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.7033380  0.0427943  86.538   <2e-16 ***
time         0.0001046  0.0003328   0.314    0.753
var_1        0.0328479  0.0301610   1.089    0.277
var_2        0.0151862  0.0301589   0.504    0.615
var_3        0.0188485  0.0301626   0.625    0.533
var_4       -0.0043585  0.0301555  -0.145    0.885
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3177 on 216 degrees of freedom
Multiple R-squared:  0.008995,  Adjusted R-squared:  -0.01395
F-statistic: 0.3921 on 5 and 216 DF,  p-value: 0.8539
```

두번째 모델에 대한 결과는 위와 같으며, 위의 결과는 다음과 같은 식으로 표현 가능하다.

$$Model_2 : \log(y_t) = 3.7033380 + 0.0001046 * t + 0.0328479 \sin(\frac{2\pi t}{L}) + 0.0151862 \cos(\frac{2\pi t}{L}) + 0.0188485 \sin(\frac{4}{}$$

위에서 생성한 모델을 **trigono_model2** 라는 변수에 저장하였다.

3.2 Binary Modeling

$$y_t = \beta_0 + \beta_1 t + \beta_2 M_1 + \beta_3 M_2 + \cdots + \beta_{11} M_{10} + \beta_{12} M_{11} + \epsilon_t$$

바이너리 모델링은 월별 Dummy 변수를 생성한다. 월별 Dummy 변수는 M1 ~ M11로써 해당 변수가 1월에 해당되면 M1 = 1 이고, 나머지 M2 ~ M11은 전부 0이 된다. 전구에 불이 켜지고 꺼지고를 생각하면 된다.

1월에 대한 표현 : 1 0 0 0 0 0 0 0 0 0 0

2월에 대한 표현 : 0 1 0 0 0 0 0 0 0 0 0

...

11월에 대한 표현 : 0 0 0 0 0 0 0 0 0 0 1

12월에 대한 표현 : 0 0 0 0 0 0 0 0 0 0 0

이렇게 나타내는 변수들이다. 이러한 변수들을 우리는 **One-Hot Variable** 이라고 한다. 이는 Machine Learning이나 Deep Learning 분석 시 Categorical 변수들을 처리하거나 자연어 처리에서 단어 또는 문자열을 수치로 표현하기 위해 활용한다. Binary Modeling은 데이터 관측치 중 Time 정보를 통해 각 월 변수로 모델링 하는 것을 말한다. (자세한 내용은 아래 코드들을 참고하면 된다)

바이너리 모델링을 위해 Date (YYYY-MM-DD) 유형의 데이터를 사용하지 않고, MM 데이터만 활용한다. 즉 2015-01-01 ~ 2015-01-31은 모두 1월 이므로 M =1 이 된다. Binary 모델링을 위한 데이터를 생성하는 코드는 다음과 같다.

- **Binary Modeling을 위한 Dataset 생성방법**

```
# lubridate::month 함수를 활용하면, YYYY-MM-DD 유형의 데이터에서 Month 만을 추출한다.
> data$M <-lubridate::month(data$time, abbr = T)
# 데이터를 확인하면 다음과 같다. 1월에는 1, 2월에는 2, ... 12월에는 12가 입력된다.
> head(data)
        time  x seq_time M
1 2015-01-04 64        1 1
2 2015-01-11 40        2 1
3 2015-01-18 39        3 1
4 2015-01-25 43        4 1
5 2015-02-01 30        5 2
6 2015-02-08 41        6 2

# Month 변수는 숫자로서 의미를 가지지 않고, Factor 변수이기 때문에 Factor 변환을 수행한다.
data$M <- as.factor(data$M)

# 분석에 필요한 검색횟수(x), 시간(time), 월(M) 컬럼을 선택하여 m_biary_data 라는 변수에 담는다.
```

```
m_binary_data <- data[,c("x","seq_time", "M")]

# 데이터는 다음과 같이 구성된다.
> head(m_binary_data)
   x seq_time M
1 64        1 1
2 40        2 1
3 39        3 1
4 43        4 1
5 30        5 2
6 41        6 2

# 변수M에 대해서는 One-Hot Encoding으로 변환하고, 나머지 값들은 그대로 담는다.
binary_data_frame <- data.frame(
  time = m_binary_data$seq_time, # 1,2,3, .... 246, 247로 구성된 time 변수
  dummy::dummy(m_binary_data, p = 'all')[c(1:11)],   # 각 월 데이터를 0, 1 변수로 생성한다. 이후 1 ~ 11까지만 선택한다.
  y=log(m_binary_data$x) ) # 종속변수(y) 검색횟수 이다.
```

- **Binary Data 생성 결과**

```
> head(binary_data_frame, 20)
   time M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_10 M_11        y
1     1   1   0   0   0   0   0   0   0   0    0    0 4.158883
2     2   1   0   0   0   0   0   0   0   0    0    0 3.688879
3     3   1   0   0   0   0   0   0   0   0    0    0 3.663562
4     4   1   0   0   0   0   0   0   0   0    0    0 3.761200
5     5   0   1   0   0   0   0   0   0   0    0    0 3.401197
6     6   0   1   0   0   0   0   0   0   0    0    0 3.713572
7     7   0   1   0   0   0   0   0   0   0    0    0 3.401197
8     8   0   1   0   0   0   0   0   0   0    0    0 3.970292
9     9   0   0   1   0   0   0   0   0   0    0    0 3.663562
10   10   0   0   1   0   0   0   0   0   0    0    0 3.931826
11   11   0   0   1   0   0   0   0   0   0    0    0 3.784190
12   12   0   0   1   0   0   0   0   0   0    0    0 4.007333
13   13   0   0   1   0   0   0   0   0   0    0    0 4.007333
14   14   0   0   0   1   0   0   0   0   0    0    0 3.496508
15   15   0   0   0   1   0   0   0   0   0    0    0 3.688879
16   16   0   0   0   1   0   0   0   0   0    0    0 3.637586
17   17   0   0   0   1   0   0   0   0   0    0    0 3.713572
18   18   0   0   0   0   1   0   0   0   0    0    0 4.007333
19   19   0   0   0   0   1   0   0   0   0    0    0 3.850148
20   20   0   0   0   0   1   0   0   0   0    0    0 3.555348
..... 중략 .....
```

특정 월에 해당되는 데이터에만 M 값이 존재한다.

- **Binary Data를 Least Square 방식으로 Fit 해보기**

```
# train, valid data split
> binary_train_data <- binary_data_frame[train_idx,]
> binary_valid_data <- binary_data_frame[valid_idx,]

# least square 방식으로 적합시킴
> binary_model <- lm(formula = y ~ ., data = binary_train_data)

# 모델 적합 결과 확인
> summary(binary_model)

Call:
lm(formula = y ~ ., data = binary_train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.61484 -0.12096  0.00135  0.12434  0.78903

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.7548001  0.0569568  65.924  < 2e-16 ***
time        -0.0001537  0.0002195  -0.700 0.484397
M_1         -0.0788378  0.0659954  -1.195 0.233599
M_2         -0.0888637  0.0673249  -1.320 0.188305
M_3          0.0736957  0.0658029   1.120 0.264022
M_4          0.0645157  0.0693022   0.931 0.352962
M_5         -0.0753622  0.0694925  -1.084 0.279408
M_6         -0.3201676  0.0713426  -4.488 1.19e-05 ***
M_7         -0.4573158  0.0681592  -6.710 1.80e-10 ***
M_8         -0.2457954  0.0701464  -3.504 0.000561 ***
```

```
M_9          0.5280450  0.0699513   7.549 1.34e-12 ***
M_10         0.2433980  0.0689492   3.530 0.000511 ***
M_11         0.1056774  0.0699560   1.511 0.132394
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2066 on 209 degrees of freedom
Multiple R-squared:  0.5943,  Adjusted R-squared:  0.571
F-statistic: 25.51 on 12 and 209 DF,  p-value: < 2.2e-16
```

Binary 방식으로 모델을 적합시킨 결과는 다음과 같다. 절편, M6, M7, M8, M9, M10의 계수값의 P-value가 낮으므로 유효하다고 볼 수 있다. 본 모델을 수식으로 나타내면 다음과 같다.

$$\log(y_t) = 3.7548001 - 0.0001537 * t - 0.0788378M_1 - 0.0888637M_2 + \cdots + 0.2433980M_{10} + 0.1056774M_{11}$$

위에서 생성된 Trigonometric Model1, Trigonometric Model2, Binary Model에 대하여 **MAE, MSE, RMSE**를 비교한다.

4. Validation Result

4.1 각 모델링 결과에 대한 Prediction 출력

```
# predict(model_result, new_data) 를 통해 예측 변수를 출력한다. 학습시 y 변수에 log 취하였으므로 밑수를 지수로 하여 다시 변환한다.
trigono_pred1 <- exp(predict(trigono_model1,
                             newdata = valid_trigono1[,c("time", "var_1", "var_2")]))

trigono_pred2 <- exp(predict(trigono_model2,
                             newdata = valid_trigono2[,c("time", "var_1", "var_2", "var_3", "var_4")]))

binary_pred <- exp(predict(binary_model,
                             newdata =binary_valid_data[,c(1:12)]))

# 각 예측결과와 실제값을 저장하는 Data Frame를 새롭게 생성한다.
ts_result_table <- data.frame(real = data$x[valid_idx],
                              tri_pred1 = trigono_pred1,
                              tri_pred2 = trigono_pred2,
                              bin_pred = binary_pred)
# 결과 테이블은 다음과 같다.
> head(ts_result_table, 20)
real tri_pred1 tri_pred2 bin_pred
223   44  40.29337  40.90451 44.03699
224   41  40.04341  40.82272 44.03022
225   54  40.18404  40.38230 44.02345
226   55  40.68223  40.12452 44.01669
227   66  41.41546  40.66288 38.26506
228   53  42.19487  42.01484 38.25917
229   34  42.80932  43.42431 38.25329
230   39  43.08517  43.90144 38.24741
231   40  42.94300  43.15142 29.93765
232   29  42.42561  41.85970 29.93304
233   31  41.68285  40.95558 29.92844
234   30  40.92109  40.78352 29.92384
235   32  40.34182  40.95590 29.91924
236   41  40.09156  40.87401 26.08081
237   30  40.23235  40.43304 26.07680
238   33  40.73115  40.17494 26.07279
239   25  41.46525  40.71397 26.06878
240   33  42.24560  42.06763 32.20447
241   32  42.86080  43.47887 32.19952
242   40  43.13698  43.95660 32.19457
```

4.2 평가기준의 정의

평가기준은 MAE, MSE, RMSE를 활용하도록 한다. 각 평가기준 함수는 다음과 같다.

```
mean_absolute_error <- function(pred, real){
  e = abs(pred - real)
  return(mean(e))
}
```

```
mean_squared_error <- function(pred, real){
  e = pred - real
  return(mean(e^2))
}

root_mean_squared_error <- function(pred, real){
  e <- (real - pred)
  return(sqrt(mean(e^2)))
}
```

4.3 평가결과

```
# 위에서 각 모델의 예측값과 실제값을 저장한 테이블을 통해, 3개 평가기준 결과를 출력한다.
ts_result_table %>% dplyr::summarise(
  MAE_tri1 = mean_absolute_error(tri_pred1, real),
  MAE_tri2 = mean_absolute_error(tri_pred2, real),
  MAE_bin  = mean_absolute_error(bin_pred, real),
  MSE_tri1 = mean_squared_error(tri_pred1, real),
  MSE_tri2 = mean_squared_error(tri_pred2, real),
  MSE_bin  = mean_squared_error(bin_pred, real),
  RMSE_tri1 = root_mean_squared_error(tri_pred1, real),
  RMSE_tri2 = root_mean_squared_error(tri_pred2, real),
  RMSE_bin  = root_mean_squared_error(bin_pred, real))
```

```
              [MAE]           |              [MSE]             |             [RMSE]
 MAE_tri1 | MAE_tri2 | MAE_bin | MSE_tri1 | MSE_tri2 | MSE_bin  | RMSE_tri1 | RMSE_tri2 | RMSE_bin
 14.09645 | 14.12416 | 8.96899 | 383.3547 | 384.3577 | 176.5377 | 19.57945  | 19.60504  | 13.28675
```

MAE, MSE, RMSE 전체 기준으로 보았을 때 Binary Model의 예측력이 우수함을 알 수 있다.

[End of Documnet]