Intro to Analytics Modeling HW 3

2024-06-05

Question 7.1:

Describe a situation or problem from your job, everyday life, current events, etc., for which exponential smoothing would be appropriate. What data would you need? Would you expect the value of (the first smoothing parameter) to be closer to 0 or 1, and why?.

Answer: In my current company, we conduct R&D on lithium metal batteries. When we test our battery cells, we charge and discharge them, where one cycle consists of one charging and one discharging step. While there are several metrics to observe, such as voltage, current, capacity, and temperature, using voltage for exponential smoothing can be particularly helpful for monitoring changes over cycles. This can indirectly indicate capacity fade and identify anomalies. Especially, it can sometimes detect safety-critical failures that have a high potential to lead to thermal runaway. Since the time series data using voltage is normally cyclic, it is not expected to have high randomness; therefore, the value of alpha tends to be closer to 1.

Importing Libraries

```
# library(ggplot2)
library(knitr)
library(tidyr)
library(stats)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

Question 7.2:

Using the 20 years of daily high temperature data for Atlanta (July through October) from Question 6.2 (file temps.txt), build and use an exponential smoothing model to help make a judgment of whether the unofficial end of summer has gotten later over the 20 years. (Part of the point of this assignment is for you to think about how you might use exponential smoothing to answer this question. Feel free to combine it with other models if you'd like to. There's certainly more than one reasonable approach.)

Note: in R, you can use either HoltWinters (simpler to use) or the smooth package's es function (harder to use, but more general). If you use es, the Holt-Winters model uses model="AAM" in the function call (the first and second constants are used "A"dditively, and the third (seasonality) is used "M"ultiplicatively; the documentation doesn't make that clear).

Read Data

```
temp_data <- read.table(
  "~/Desktop/ISYE-6501/week 3 Homework-Summer24/week 3 data-summer/temps.txt",
  stringsAsFactors = FALSE,
  header=TRUE
)
head(temp_data)
```

```
##     DAY X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003 X2004 X2005 X2006 X2007
## 1 1-Jul    98    86    91    84    89    84    90    73    82    91    93    95
## 2 2-Jul    97    90    88    82    91    87    90    81    81    89    93    85
## 3 3-Jul    97    93    91    87    93    87    87    87    86    86    93    82
## 4 4-Jul    90    91    91    88    95    84    89    86    88    86    91    86
## 5 5-Jul    89    84    91    90    96    86    93    80    90    89    90    88
## 6 6-Jul    93    84    89    91    96    87    93    84    90    82    81    87
##   X2008 X2009 X2010 X2011 X2012 X2013 X2014 X2015
## 1    85    95    87    92   105    82    90    85
## 2    87    90    84    94    93    85    93    87
## 3    91    89    83    95    99    76    87    79
## 4    90    91    85    92    98    77    84    85
## 5    88    80    88    90   100    83    86    84
## 6    82    87    89    90    98    83    87    84
```

```
years <- names(temp_data[,2:ncol(temp_data)])
df_temps <- pivot_longer(temp_data, cols=years, names_to="Year", values_to="Temperature")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(years)
##
##   # Now:
##   data %>% select(all_of(years))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
# Data cleansing
df_temps$Year <- as.integer(gsub("X", "", df_temps$Year))
df_temps$ts <- as.Date(paste(df_temps$DAY, df_temps$Year), format = "%d-%b %Y")
df_temps$Month <- as.integer(format(df_temps$ts, "%m"))
df_temps$Temperature <- as.numeric(df_temps$Temperature)

df_temps <- df_temps[order(df_temps$ts), ]
df_temps <- df_temps[, c("ts", "Temperature", "Year", "Month")]
df_temps
```

```
## # A tibble: 2,460 x 4
```

```
##    ts         Temperature  Year Month
##    <date>           <dbl> <int> <int>
##  1 1996-07-01          98  1996     7
##  2 1996-07-02          97  1996     7
##  3 1996-07-03          97  1996     7
##  4 1996-07-04          90  1996     7
##  5 1996-07-05          89  1996     7
##  6 1996-07-06          93  1996     7
##  7 1996-07-07          93  1996     7
##  8 1996-07-08          91  1996     7
##  9 1996-07-09          93  1996     7
## 10 1996-07-10          93  1996     7
## # i 2,450 more rows
```

CUSUM Recap

```r
years <- unique(df_temps$Year)
end_summer_dates <- as.Date(character(0))
for (year in years) {
  temporary_table <- df_temps[df_temps$Year == year,]
  mean_overall <- mean(temporary_table$Temperature)
  std_overall <- sd(temporary_table$Temperature)
  C <- std_overall
  T <- 3 * std_overall #threshold set to 3 sigma

  temporary_table[1,"St"] <- 0 # Assuming S_0 is 0

  for(t in 2:nrow(temporary_table)) {
    temporary_table[t, "St"] <-
      max(0, (temporary_table[t-1, "St"] + mean_overall - temporary_table[t, "Temperature"] - C)$St)
  }

  end_summer_dates <- c(
    end_summer_dates,
    as.Date(temporary_table[which(temporary_table$St>T),][1,]$ts)
  )
}
end_summer_dates <- df_temps[df_temps$ts %in% end_summer_dates, ][, c("ts", "Temperature")]

end_summer_dates$Year <- as.integer(format(end_summer_dates$ts, "%Y"))
end_summer_dates$month_date <- as.integer(format(end_summer_dates$ts, "%m%d"))
end_summer_dates$Year_norm <- end_summer_dates$Year / max(end_summer_dates$Year)
end_summer_dates$month_date_norm <- end_summer_dates$month_date / max(end_summer_dates$month_date)
end_summer_dates
```

```
## # A tibble: 20 x 6
##    ts         Temperature  Year month_date Year_norm month_date_norm
##    <date>           <dbl> <int>      <int>     <dbl>           <dbl>
##  1 1996-10-02          72  1996       1002     0.991           0.978
##  2 1997-10-16          57  1997       1016     0.991           0.991
##  3 1998-10-10          73  1998       1010     0.992           0.985
##  4 1999-10-20          60  1999       1020     0.992           0.995
```

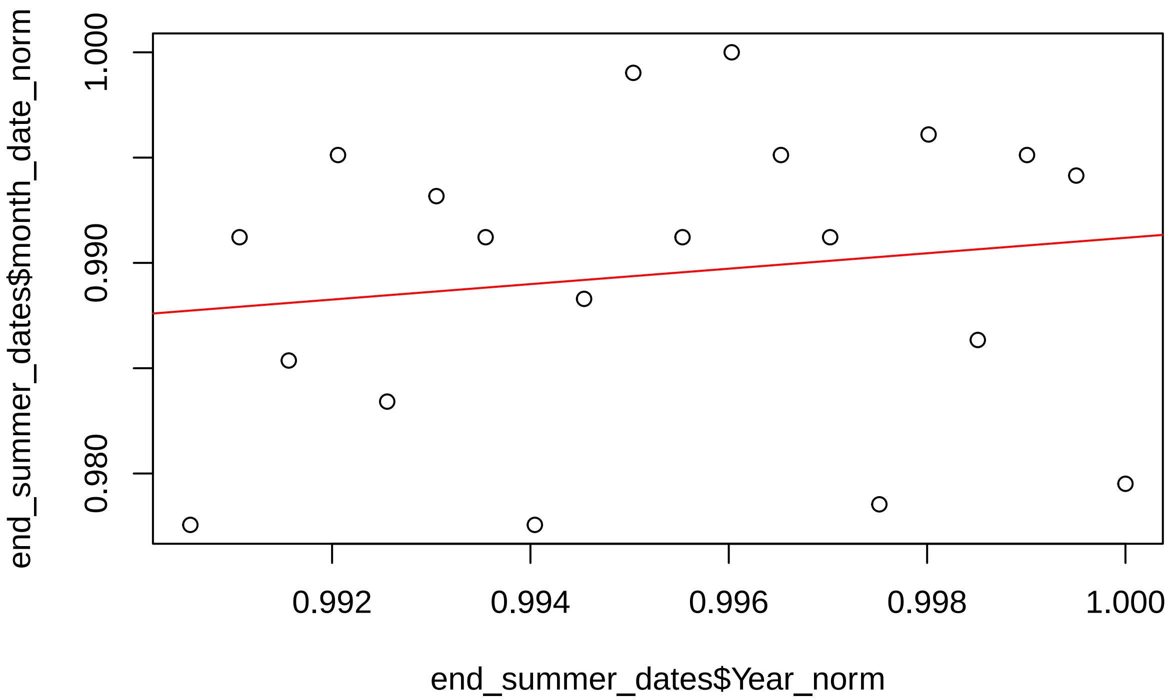
```
##   5 2000-10-08          55  2000       1008      0.993            0.983
##   6 2001-10-18          64  2001       1018      0.993            0.993
##   7 2002-10-16          66  2002       1016      0.994            0.991
##   8 2003-10-02          68  2003       1002      0.994            0.978
##   9 2004-10-13          64  2004       1013      0.995            0.988
##  10 2005-10-24          56  2005       1024      0.995            0.999
##  11 2006-10-16          59  2006       1016      0.996            0.991
##  12 2007-10-25          61  2007       1025      0.996            1
##  13 2008-10-20          66  2008       1020      0.997            0.995
##  14 2009-10-16          61  2009       1016      0.997            0.991
##  15 2010-10-03          68  2010       1003      0.998            0.979
##  16 2011-10-21          63  2011       1021      0.998            0.996
##  17 2012-10-11          75  2012       1011      0.999            0.986
##  18 2013-10-20          70  2013       1020      0.999            0.995
##  19 2014-10-19          73  2014       1019      1.00             0.994
##  20 2015-10-04          70  2015       1004      1                0.980
```

Explanation: This is a list of unofficial summer end dates for each year based on CUSUM algorithm. Note that mean and standard deviation for C and T are applied dynamically for each year separately as it's described in the for loop.

```r
x <- as.vector(end_summer_dates$Year_norm)
y <- as.vector(end_summer_dates$month_date_norm)

linear_model <- lm(y ~ x)
intercept <- linear_model$coefficients[1]
slope <- linear_model$coefficients[2]

plot(end_summer_dates$Year_norm, end_summer_dates$month_date_norm)
# Add linear line
abline(a = intercept, b = slope, col = "red")
```



Explanation: The X-axis represents normalized years, and the Y-axis represents normalized month_date.

```
linear_model
```

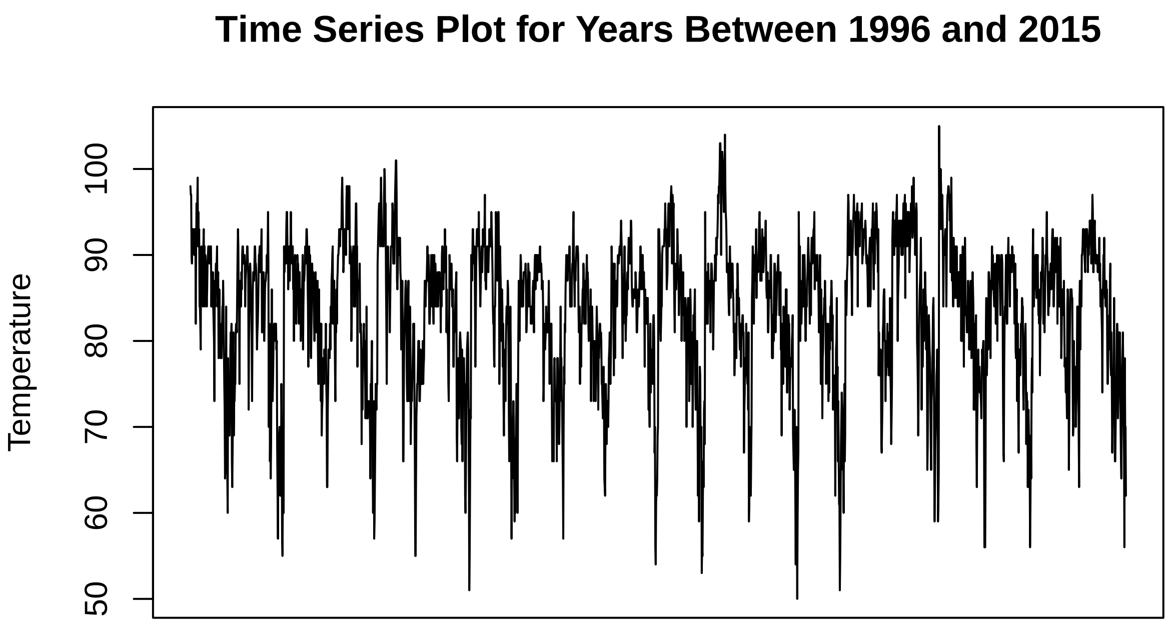
```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      0.6246       0.3666
```

Observation: Based on how the CUSUM algorithm is applied (mean and standard deviation for C and T are applied dynamically for each year separately), it produces unofficial summer end dates for each year. When the result is displayed in a scatter plot along with a linear regression line, it indicates that the unofficial end of summer has gotten later over the 20 years.

Exponential Smoothing

```
# 123 days from July to October for the frequency
ts_data_all <- ts(df_temps$Temperature, start = min(df_temps$ts), frequency = 123)
# Plot the time series with continuous date x-axis
plot(
  ts_data_all,
```

```r
  xlab = "",
  ylab = "Temperature",
  main = "Time Series Plot for Years Between 1996 and 2015",
  xaxt="n"
)
```

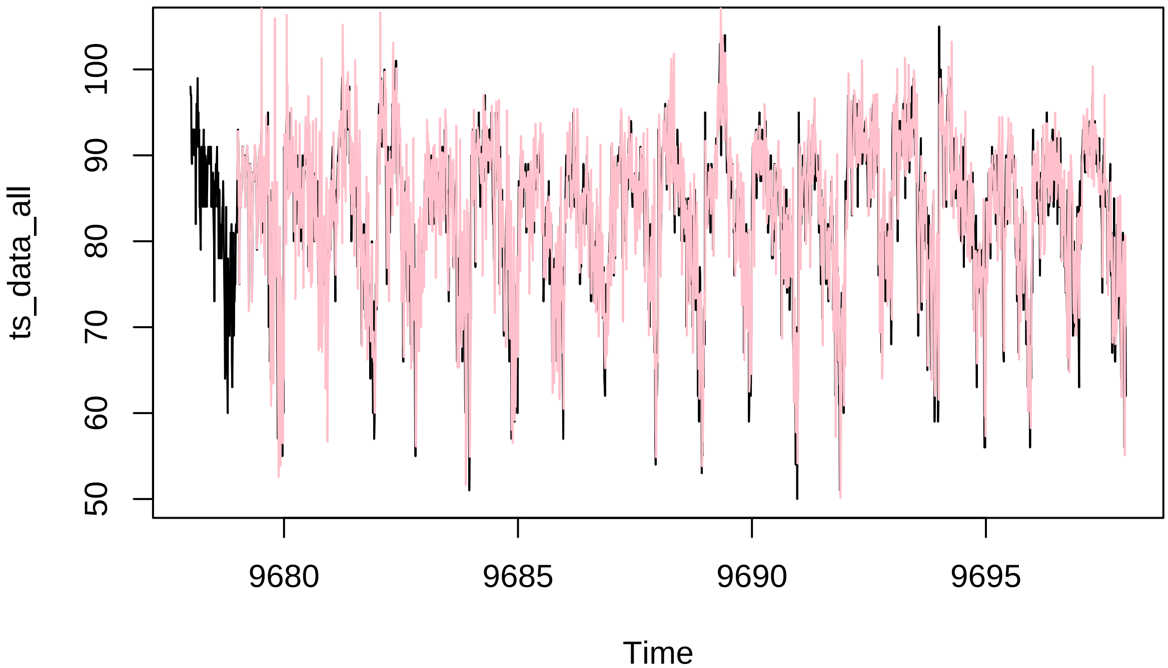


Observation: There appears to be a cyclic pattern over the years, but it's not clear whether the overall trend is increasing or decreasing. Additionally, there is some level of randomness in daily high temperatures. Therefore, it makes sense to utilize triple exponential smoothing, specifically the Holt-Winters method.

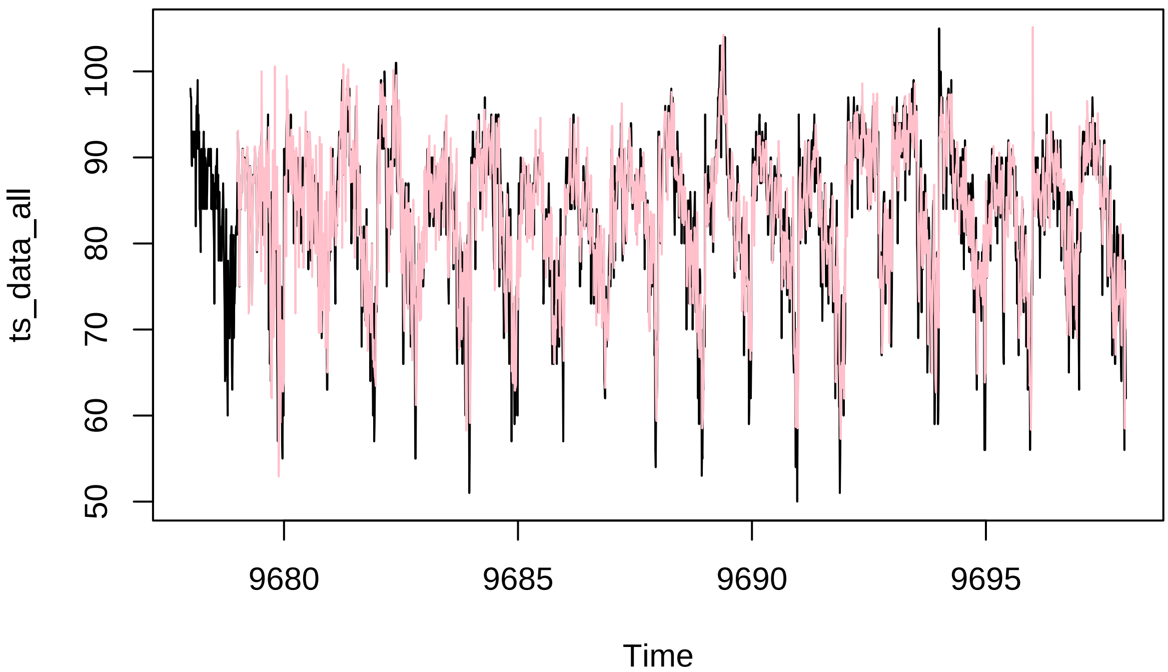
```r
# Define a function for an exponential smoothing model
build_hw_model = function(data, a, b, g, s) {
  hw_model = HoltWinters(
    data,
    alpha=a,
    beta=b,
    gamma=g,
    seasonal=s,
  )
  return(hw_model)
}
```

```r
# A model that an exponential smoothing is applied
hw_model = build_hw_model(ts_data_all, 0.9, NULL, NULL, "m")
smoothed_values <- fitted(hw_model)
```

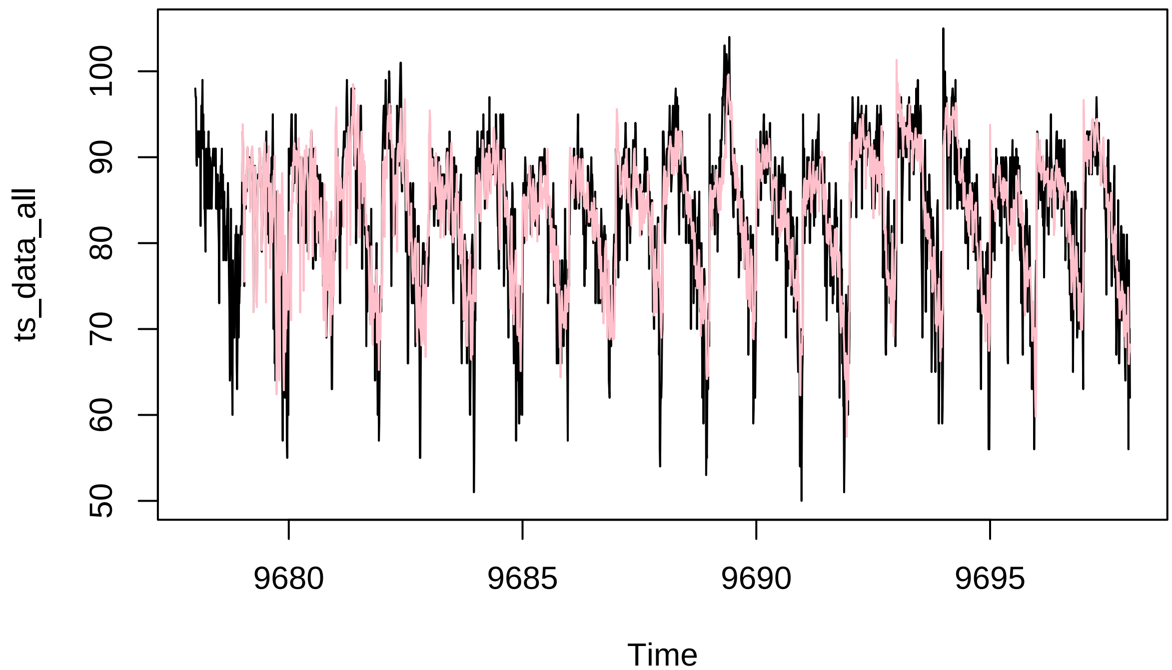
```r
plot(ts_data_all)
lines(smoothed_values[,1], col="pink")
```




```r
# A model that an exponential smoothing is applied
hw_model = build_hw_model(ts_data_all, 0.5, NULL, NULL, "m")
smoothed_values <- fitted(hw_model)
plot(ts_data_all)
lines(smoothed_values[,1], col="pink")
```




```r
# A model that an exponential smoothing is applied
hw_model = build_hw_model(ts_data_all, 0.1, NULL, NULL, "m")
smoothed_values <- fitted(hw_model)
plot(ts_data_all)
lines(smoothed_values[,1], col="pink")
```



Observation: By applying different values for alpha (0.1, 0.5, and 0.9), it appears that 0.1 is the optimal value for exponential smoothing of the time series data.

```
print(paste("alpha:", hw_model$alpha, " beta:", hw_model$beta, "gamma:", hw_model$gamma))
```

```
## [1] "alpha: 0.1  beta: 0 gamma: 0.279818248767374"
```

Observation: With the model, there's no trend identified but a bit of seasonality identified.

```
summary(hw_model)
```

```
##              Length Class  Mode
## fitted       9348   mts    numeric
## x            2460   ts     numeric
## alpha           1   -none- numeric
## beta            1   -none- numeric
## gamma           1   -none- numeric
## coefficients  125   -none- numeric
## seasonal        1   -none- character
## SSE             1   -none- numeric
## call            6   -none- call
```

```
hw_model$coefficients[1:2]
```

```
##            a            b
## 85.544930095 -0.004362918
```

Observation: Based on the current estimate of the baseline coefficient, which can be interpreted in a linear model, 'b' indicates the slope of the fitted curve. It appears to be close to zero, suggesting that there may be no significant increase or decrease in daily high temperatures over the 20-year period after the data is smoothed by exponential smoothing. However, it's important to note that this observation captures only the overall trend. We are interested in determining if there exists a trend specifically over the unofficial end-of-summer dates during the 20-year period. Let's recompute the unofficial summer end dates against the smoothed data by applying CUSUM algorithm.

CUSUM using Smoothed temperature

```r
# exclude 1996 data since there's no smooth data for the first year because the seasonality coefficient
df_temps_new <- df_temps[df_temps$Year > 1996,]

# down sample the smoothed temperature by applying a frequency of 123
# smoothed_values comes from hw_model with an alpha of 0.1
downsampled_smoothed_data <- downSample(x=smoothed_values, factor(df_temps_new$ts))
df_temps_new$t_hat <- downsampled_smoothed_data$xhat
```

```r
# Compute CUSUM for each year against the smoothed data
years <- unique(df_temps_new$Year)
end_summer_dates <- as.Date(character(0))
for (year in years) {
  temporary_table <- df_temps_new[df_temps_new$Year == year,]
  mean_overall <- mean(temporary_table$t_hat)
  std_overall <- sd(temporary_table$t_hat)
  C <- std_overall
  T <- 3 * std_overall #threshold set to 3 sigma

  temporary_table[1,"St"] <- 0 # Assuming S_0 is 0

  for(t in 2:nrow(temporary_table)) {
    temporary_table[t, "St"] <-
      max(0, (temporary_table[t-1, "St"] + mean_overall - temporary_table[t, "t_hat"] - C)$St)
  }

  end_summer_dates <- c(
    end_summer_dates,
    as.Date(temporary_table[which(temporary_table$St>T),][1,]$ts)
  )
}
end_summer_dates <- df_temps_new[df_temps_new$ts %in% end_summer_dates, ][, c("ts", "t_hat")]

end_summer_dates$Year <- as.integer(format(end_summer_dates$ts, "%Y"))
end_summer_dates$month_date <- as.integer(format(end_summer_dates$ts, "%m%d"))
end_summer_dates$Year_norm <- end_summer_dates$Year / max(end_summer_dates$Year)
end_summer_dates$month_date_norm <- end_summer_dates$month_date / max(end_summer_dates$month_date)
end_summer_dates
```

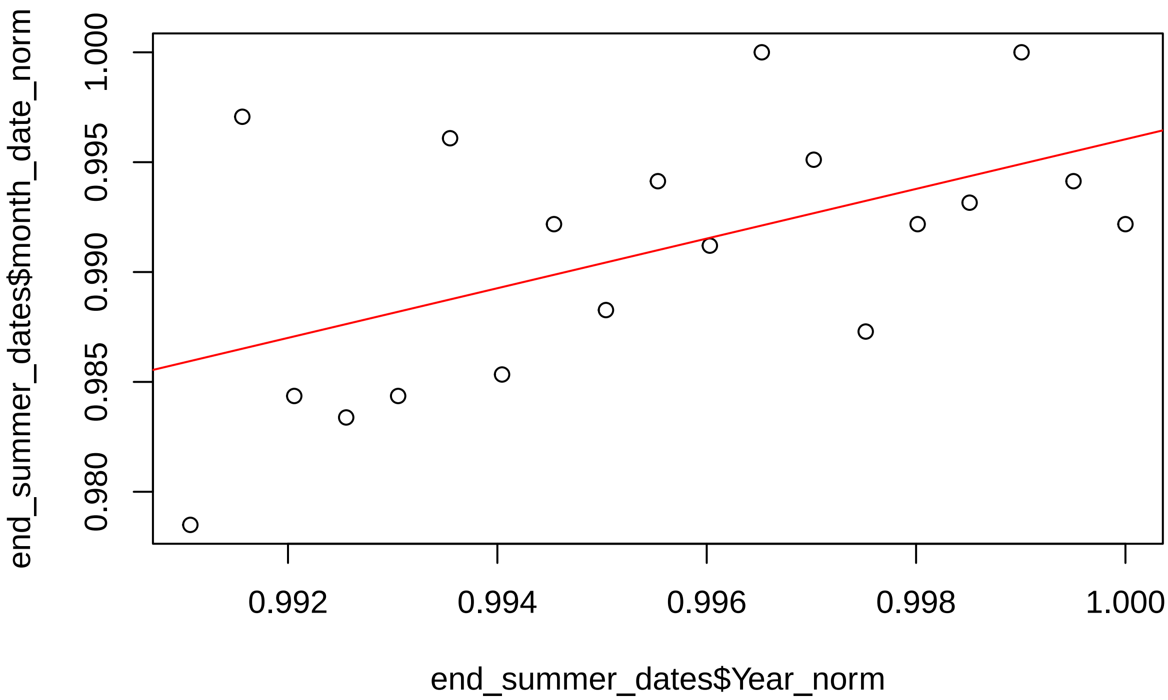
```
## # A tibble: 19 x 6
```

```
##    ts          t_hat  Year month_date Year_norm month_date_norm
##    <date>      <dbl> <int>      <int>     <dbl>           <dbl>
##  1 1997-10-01  66.9  1997       1001     0.991           0.978
##  2 1998-10-20  74.7  1998       1020     0.992           0.997
##  3 1999-10-07  68.2  1999       1007     0.992           0.984
##  4 2000-10-06  70.2  2000       1006     0.993           0.983
##  5 2001-10-07  68.1  2001       1007     0.993           0.984
##  6 2002-10-19  69.3  2002       1019     0.994           0.996
##  7 2003-10-08  68.5  2003       1008     0.994           0.985
##  8 2004-10-15  68.8  2004       1015     0.995           0.992
##  9 2005-10-11  76.3  2005       1011     0.995           0.988
## 10 2006-10-17  71.2  2006       1017     0.996           0.994
## 11 2007-10-14  74.4  2007       1014     0.996           0.991
## 12 2008-10-23  68.8  2008       1023     0.997           1
## 13 2009-10-18  66.3  2009       1018     0.997           0.995
## 14 2010-10-10  75.7  2010       1010     0.998           0.987
## 15 2011-10-15  70.5  2011       1015     0.998           0.992
## 16 2012-10-16  73.9  2012       1016     0.999           0.993
## 17 2013-10-23  71.3  2013       1023     0.999           1
## 18 2014-10-17  74.9  2014       1017     1.00            0.994
## 19 2015-10-15  69.2  2015       1015     1               0.992
```

```r
x <- as.vector(end_summer_dates$Year_norm)
y <- as.vector(end_summer_dates$month_date_norm)

linear_model <- lm(y ~ x)
intercept <- linear_model$coefficients[1]
slope <- linear_model$coefficients[2]

plot(end_summer_dates$Year_norm, end_summer_dates$month_date_norm)
# Add linear line
abline(a = intercept, b = slope, col = "red")
```



Explanation: The X-axis represents normalized years, and the Y-axis represents normalized month_date.

```
linear_model
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##     -0.1339       1.1300
```

Observation: Based on how the CUSUM algorithm is applied (mean and standard deviation for C and T are applied dynamically for each year separately) against the smoothed data, it produces unofficial summer end dates for each year again. When the result is displayed in a scatter plot along with a linear regression line, it clearly indicates that the unofficial end of summer has gotten later over the 20 years.

Question 8.1:

Describe a situation or problem from your job, everyday life, current events, etc., for which a linear regression model would be appropriate. List some (up to 5) predictors that you might use.

I have a small rechargeable desk fan, and I'm interested in assessing its battery health, specifically in terms of the capacity for a full charge. To achieve this, I believe that applying a linear regression model would be appropriate, considering several potential predictors:

- daily temperature
- hours of desk usage
- hours of daily charging time
- hours of daily operating time
- average speed of the fan (1 through 4)
- cumulative hours of usage

Question 8.2:

Using crime data from http://www.statsci.org/data/general/uscrime.txt (file uscrime.txt, description at http://www.statsci.org/data/general/uscrime.html), use regression (a useful R function is lm or glm) to predict the observed crime rate in a city with the following data:

- M = 14.0
- So = 0
- Ed = 10.0
- Po1 = 12.0
- Po2 = 15.5
- LF = 0.640
- M.F = 94.0
- Pop = 150
- NW = 1.1
- U1 = 0.120
- U2 = 3.6
- Wealth = 3200
- Ineq = 20.1
- Prob = 0.04
- Time = 39.0

Show your model (factors used and their coefficients), the software output, and the quality of fit.

Note *that because there are only 47 data points and 15 predictors, you'll probably notice some overfitting. We'll see ways of dealing with this sort of problem later in the course.*

Read Data

```
crime_data <- read.table("~/Desktop/ISYE-6501/week 3 Homework-Summer24/week 3 data-summer/uscrime.txt",
crime_data
```

```
##         M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1  15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2  14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3  14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4  13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5  14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6  12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
## 7  12.7  1 11.1  8.2  7.9 0.519  98.2   4 13.9 0.097 3.8   6200 16.8 0.042100
## 8  13.1  1 10.9 11.5 10.9 0.542  96.9  50 17.9 0.079 3.5   4720 20.6 0.040099
## 9  15.7  1  9.0  6.5  6.2 0.553  95.5  39 28.6 0.081 2.8   4210 23.9 0.071697
```

```
## 10 14.0  0 11.8  7.1  6.8 0.632 102.9   7  1.5 0.100 2.4   5260 17.4 0.044498
## 11 12.4  0 10.5 12.1 11.6 0.580  96.6 101 10.6 0.077 3.5   6570 17.0 0.016201
## 12 13.4  0 10.8  7.5  7.1 0.595  97.2  47  5.9 0.083 3.1   5800 17.2 0.031201
## 13 12.8  0 11.3  6.7  6.0 0.624  97.2  28  1.0 0.077 2.5   5070 20.6 0.045302
## 14 13.5  0 11.7  6.2  6.1 0.595  98.6  22  4.6 0.077 2.7   5290 19.0 0.053200
## 15 15.2  1  8.7  5.7  5.3 0.530  98.6  30  7.2 0.092 4.3   4050 26.4 0.069100
## 16 14.2  1  8.8  8.1  7.7 0.497  95.6  33 32.1 0.116 4.7   4270 24.7 0.052099
## 17 14.3  0 11.0  6.6  6.3 0.537  97.7  10  0.6 0.114 3.5   4870 16.6 0.076299
## 18 13.5  1 10.4 12.3 11.5 0.537  97.8  31 17.0 0.089 3.4   6310 16.5 0.119804
## 19 13.0  0 11.6 12.8 12.8 0.536  93.4  51  2.4 0.078 3.4   6270 13.5 0.019099
## 20 12.5  0 10.8 11.3 10.5 0.567  98.5  78  9.4 0.130 5.8   6260 16.6 0.034801
## 21 12.6  0 10.8  7.4  6.7 0.602  98.4  34  1.2 0.102 3.3   5570 19.5 0.022800
## 22 15.7  1  8.9  4.7  4.4 0.512  96.2  22 42.3 0.097 3.4   2880 27.6 0.089502
## 23 13.2  0  9.6  8.7  8.3 0.564  95.3  43  9.2 0.083 3.2   5130 22.7 0.030700
## 24 13.1  0 11.6  7.8  7.3 0.574 103.8   7  3.6 0.142 4.2   5400 17.6 0.041598
## 25 13.0  0 11.6  6.3  5.7 0.641  98.4  14  2.6 0.070 2.1   4860 19.6 0.069197
## 26 13.1  0 12.1 16.0 14.3 0.631 107.1   3  7.7 0.102 4.1   6740 15.2 0.041698
## 27 13.5  0 10.9  6.9  7.1 0.540  96.5   6  0.4 0.080 2.2   5640 13.9 0.036099
## 28 15.2  0 11.2  8.2  7.6 0.571 101.8  10  7.9 0.103 2.8   5370 21.5 0.038201
## 29 11.9  0 10.7 16.6 15.7 0.521  93.8 168  8.9 0.092 3.6   6370 15.4 0.023400
## 30 16.6  1  8.9  5.8  5.4 0.521  97.3  46 25.4 0.072 2.6   3960 23.7 0.075298
## 31 14.0  0  9.3  5.5  5.4 0.535 104.5   6  2.0 0.135 4.0   4530 20.0 0.041999
## 32 12.5  0 10.9  9.0  8.1 0.586  96.4  97  8.2 0.105 4.3   6170 16.3 0.042698
## 33 14.7  1 10.4  6.3  6.4 0.560  97.2  23  9.5 0.076 2.4   4620 23.3 0.049499
## 34 12.6  0 11.8  9.7  9.7 0.542  99.0  18  2.1 0.102 3.5   5890 16.6 0.040799
## 35 12.3  0 10.2  9.7  8.7 0.526  94.8 113  7.6 0.124 5.0   5720 15.8 0.020700
## 36 15.0  0 10.0 10.9  9.8 0.531  96.4   9  2.4 0.087 3.8   5590 15.3 0.006900
## 37 17.7  1  8.7  5.8  5.6 0.638  97.4  24 34.9 0.076 2.8   3820 25.4 0.045198
## 38 13.3  0 10.4  5.1  4.7 0.599 102.4   7  4.0 0.099 2.7   4250 22.5 0.053998
## 39 14.9  1  8.8  6.1  5.4 0.515  95.3  36 16.5 0.086 3.5   3950 25.1 0.047099
## 40 14.5  1 10.4  8.2  7.4 0.560  98.1  96 12.6 0.088 3.1   4880 22.8 0.038801
## 41 14.8  0 12.2  7.2  6.6 0.601  99.8   9  1.9 0.084 2.0   5900 14.4 0.025100
## 42 14.1  0 10.9  5.6  5.4 0.523  96.8   4  0.2 0.107 3.7   4890 17.0 0.088904
## 43 16.2  1  9.9  7.5  7.0 0.522  99.6  40 20.8 0.073 2.7   4960 22.4 0.054902
## 44 13.6  0 12.1  9.5  9.6 0.574 101.2  29  3.6 0.111 3.7   6220 16.2 0.028100
## 45 13.9  1  8.8  4.6  4.1 0.480  96.8  19  4.9 0.135 5.3   4570 24.9 0.056202
## 46 12.6  0 10.4 10.6  9.7 0.599  98.9  40  2.4 0.078 2.5   5930 17.1 0.046598
## 47 13.0  0 12.1  9.0  9.1 0.623 104.9   3  2.2 0.113 4.0   5880 16.0 0.052802
##       Time Crime
## 1  26.2011   791
## 2  25.2999  1635
## 3  24.3006   578
## 4  29.9012  1969
## 5  21.2998  1234
## 6  20.9995   682
## 7  20.6993   963
## 8  24.5988  1555
## 9  29.4001   856
## 10 19.5994   705
## 11 41.6000  1674
## 12 34.2984   849
## 13 36.2993   511
## 14 21.5010   664
## 15 22.7008   798
```

```
## 16 26.0991   946
## 17 19.1002   539
## 18 18.1996   929
## 19 24.9008   750
## 20 26.4010  1225
## 21 37.5998   742
## 22 37.0994   439
## 23 25.1989  1216
## 24 17.6000   968
## 25 21.9003   523
## 26 22.1005  1993
## 27 28.4999   342
## 28 25.8006  1216
## 29 36.7009  1043
## 30 28.3011   696
## 31 21.7998   373
## 32 30.9014   754
## 33 25.5005  1072
## 34 21.6997   923
## 35 37.4011   653
## 36 44.0004  1272
## 37 31.6995   831
## 38 16.6999   566
## 39 27.3004   826
## 40 29.3004  1151
## 41 30.0001   880
## 42 12.1996   542
## 43 31.9989   823
## 44 30.0001  1030
## 45 32.5996   455
## 46 16.6999   508
## 47 16.0997   849
```

```
linear_model_crime <- lm(Crime~., data=crime_data)
summary(linear_model_crime)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = crime_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -395.74  -98.09   -6.69  112.99  512.67
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.984e+03  1.628e+03  -3.675 0.000893 ***
## M            8.783e+01  4.171e+01   2.106 0.043443 *
## So          -3.803e+00  1.488e+02  -0.026 0.979765
## Ed           1.883e+02  6.209e+01   3.033 0.004861 **
## Po1          1.928e+02  1.061e+02   1.817 0.078892 .
## Po2         -1.094e+02  1.175e+02  -0.931 0.358830
## LF          -6.638e+02  1.470e+03  -0.452 0.654654
## M.F          1.741e+01  2.035e+01   0.855 0.398995
```

```
## Pop         -7.330e-01  1.290e+00  -0.568 0.573845
## NW           4.204e+00  6.481e+00   0.649 0.521279
## U1          -5.827e+03  4.210e+03  -1.384 0.176238
## U2           1.678e+02  8.234e+01   2.038 0.050161 .
## Wealth       9.617e-02  1.037e-01   0.928 0.360754
## Ineq         7.067e+01  2.272e+01   3.111 0.003983 **
## Prob        -4.855e+03  2.272e+03  -2.137 0.040627 *
## Time        -3.479e+00  7.165e+00  -0.486 0.630708
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 209.1 on 31 degrees of freedom
## Multiple R-squared:  0.8031, Adjusted R-squared:  0.7078
## F-statistic: 8.429 on 15 and 31 DF,  p-value: 3.539e-07
```

Interpretation of the summary information for the linear regression model *I asked ChatGPT the following question to understand the information available in the summary function output for my linear regression model. "How to interpret summary(my linear regression model using lm() function in R)?"*

- Coefficients: This section displays the estimated coefficients for each predictor variable in the model. It includes columns for the estimated coefficient, standard error, t-value, and p-value. The coefficient represents the change in the response variable for a one-unit change in the predictor, holding all other predictors constant.
- Residuals: This section provides information about the residuals of the model, including the minimum, 1st quartile, median, mean, 3rd quartile, and maximum values. Residuals are the differences between the observed and predicted values of the response variable.
- Coefficients' significance: The p-values associated with each coefficient test the null hypothesis that the corresponding coefficient is zero (i.e., the predictor has no effect on the response). Lower p-values suggest stronger evidence against the null hypothesis and indicate that the predictor is likely to be significant.
- Multiple R-squared and adjusted R-squared: These metrics quantify the goodness-of-fit of the model. Multiple R-squared represents the proportion of variance in the response variable explained by the predictors, while adjusted R-squared adjusts for the number of predictors in the model.
- F-statistic: This statistic tests the overall significance of the model by comparing the variance explained by the model to the variance not explained. A low p-value for the F-statistic indicates that the model is a significant improvement over a null model with no predictors.
- Residual standard error: This is an estimate of the standard deviation of the errors in predicting the response variable. It provides a measure of the model's accuracy in predicting new observations.
- Degrees of Freedom: These values represent the degrees of freedom associated with the model, residual, and total.
- Significance stars: Some versions of the summary output use stars to indicate the level of significance of the coefficients, with more stars indicating higher significance.

```r
# create a new data set to predict the observed crime rate in a city with the following data
M <- c(14.0)
So <- c(0)
Ed <- c(10.0)
Po1 <- c(12.0)
```

```r
Po2 <- c(15.5)
LF <- c(0.640)
M.F <- c(94.0)
Pop <- c(150)
NW <- c(1.1)
U1 <- c(0.120)
U2 <- c(3.6)
Wealth <- c(3200)
Ineq <- c(20.1)
Prob <- c(0.04)
Time <- c(39.0)
new_data <- data.frame(
  M = M,
  So = So,
  Ed = Ed,
  Po1 = Po1,
  Po2 = Po2,
  LF = LF,
  M.F = M.F,
  Pop = Pop,
  NW = NW,
  U1 = U1,
  U2 = U2,
  Wealth = Wealth,
  Ineq = Ineq,
  Prob = Prob,
  Time = Time
)
new_data
```

```
##    M So Ed Po1  Po2   LF M.F Pop  NW   U1  U2 Wealth Ineq Prob Time
## 1 14  0 10  12 15.5 0.64  94 150 1.1 0.12 3.6   3200 20.1 0.04   39
```

```r
predict(linear_model_crime, newdata=new_data)
```

Prediction for the given data

```
##        1
## 155.4349
```

Observation: There are only five predictors obtaining star(s) while there are 15 predictors. This means that the remaining predictors with high p-value are likely to have no effect on the response variable and this potentially indicates that the model is overfitted.