ISL Assignment 3

3_1 Question:

You're to use the KC Weather Data ("kc_weather_srt.csv", available here: kc_weather_srt.csv). The data has categorized the weather for each day into three categories ("Events": Rain, Rain_Thunderstorm, Snow) over the three years 2014, 2015, and 2016. You'll note that not all dates are listed because it's a filtered subset where other categories or no events are deleted to have a more manageable subset. The entire dataset has 366 entries. The column labels indicate the units as well such as Temp.F means temperature in Fahrenheit, Visibility.mi means Visibility in miles, etc.

You're to do two level of analysis

Consider next the entire dataset consisting of 366 entries. Now logistics regression cannot be applied, but you can apply the rest of them. Repeat the above studies in i) and ii) with LDA, QDA, and knn on the entire data set (using 290 of them in a training set). Do not forget randomization and 100 replications for this study."

Solution:

Linear Kernel :

```
library('e1071')
kcweather<-read.csv(file="C:\\Users\\praga\\Downloads\\kc_weather_srt.csv",head=T,sep=",")
kcweather<-kcweather[,2:9]
n=366
nt=290
neval=n-nt
rep=100
accuracy=dim(rep)
precision_Rain=dim(rep)
Recall_Rain=dim(rep)
precision_Rain_Thunderstorm=dim(rep)
Recall_Rain_Thunderstorm=dim(rep)
precision_Snow=dim(rep)
Recall_Snow=dim(rep)

for(k in 1:rep)
{
  Tkcweather = sample(1:n,nt)
  kcweather.Train = kcweather[Tkcweather,]
  kcweather.Test = kcweather[-Tkcweather,]
  svmfit = svm(Events~.,data=kcweather.Train,kernel='linear',cost=1)
  p=predict(svmfit,kcweather.Test)
  cmatrix = table(p,kcweather.Test$Events)
  accuracy[k] = sum(diag(cmatrix))/sum(cmatrix)
  precision_Rain[k] = cmatrix[1,1]/(cmatrix[1,1]+cmatrix[2,1]+cmatrix[3,1])
  precision_Rain_Thunderstorm[k] = cmatrix[2,2]/(cmatrix[1,2]+cmatrix[2,2]+cmatrix[3,2])
  precision_Snow[k] = cmatrix[3,3]/(cmatrix[1,3]+cmatrix[2,3]+cmatrix[3,3])
  Recall_Rain[k] = cmatrix[1,1]/(cmatrix[1,1]+cmatrix[1,2]+cmatrix[1,3])
  Recall_Rain_Thunderstorm[k] = cmatrix[2,2]/(cmatrix[2,1]+cmatrix[2,2]+cmatrix[2,3])
  Recall_Snow[k] = cmatrix[3,3]/(cmatrix[3,1]+cmatrix[3,2]+cmatrix[3,3])
}
  summary(svmfit)
  # calculate mean
  mean(accuracy)
  mean(precision_Rain)
  mean(Recall_Rain)
  mean(precision_Rain_Thunderstorm)
  mean(Recall_Rain_Thunderstorm)
  mean(precision_Snow)
  mean(Recall_Snow)

  # Compute 95% confidence interval on Accuracy using t-distribution
  AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
  # show the confidence interval [for Accuracy]
  mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err

  # summaries
  summary(accuracy)
  summary(precision_Rain)
  summary(Recall_Rain)
  summary(precision_Rain_Thunderstorm)
  summary(Recall_Rain_Thunderstorm)
  summary(precision_Snow)
  summary(Recall_Snow)
  svm_tune <- tune(svm,
        Events~.,data=kcweather.Train,kernel='linear',ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))
  summary(svm_tune)
```

Execution:

```
> kcweather<-kcweather[,2:9]
> n=366
> nt=290
> neval=n-nt
> rep=100
> accuracy=dim(rep)
> precision_Rain=dim(rep)
> Recall_Rain=dim(rep)
> precision_Rain_Thunderstorm=dim(rep)
> Recall_Rain_Thunderstorm=dim(rep)
> precision_Snow=dim(rep)
> Recall_Snow=dim(rep)
>
> for(k in 1:rep)
+ {
+     Tkcweather = sample(1:n,nt)
+     kcweather.Train = kcweather[Tkcweather,]
+     kcweather.Test = kcweather[-Tkcweather,]
+     svmfit = svm(Events~.,data=kcweather.Train,kernel='linear',cost=1)
+     p=predict(svmfit,kcweather.Test)
+     cmatrix = table(p,kcweather.Test$Events)
+     accuracy[k] = sum(diag(cmatrix))/sum(cmatrix)
+     precision_Rain[k] = cmatrix[1,1]/(cmatrix[1,1]+cmatrix[2,1]+cmatrix[3,1])
+     precision_Rain_Thunderstorm[k] = cmatrix[2,2]/(cmatrix[1,2]+cmatrix[2,2]+cmatrix[3,2])
+     precision_Snow[k] = cmatrix[3,3]/(cmatrix[1,3]+cmatrix[2,3]+cmatrix[3,3])
+     Recall_Rain[k] = cmatrix[1,1]/(cmatrix[1,1]+cmatrix[1,2]+cmatrix[1,3])
+     Recall_Rain_Thunderstorm[k] = cmatrix[2,2]/(cmatrix[2,1]+cmatrix[2,2]+cmatrix[2,3])
+     Recall_Snow[k] = cmatrix[3,3]/(cmatrix[3,1]+cmatrix[3,2]+cmatrix[3,3])
+ }
> summary(svmfit)

Call:
svm(formula = Events ~ ., data = kcweather.Train, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.1428571

Number of Support Vectors:  156

 ( 65 79 12 )


Number of Classes:  3

Levels:
 Rain Rain_Thunderstorm Snow
```

Output Results:

```
Call:
svm(formula = Events ~ ., data = kcweather.Train, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.1428571

Number of Support Vectors:  156

 ( 65 79 12 )


Number of Classes:  3

Levels:
 Rain Rain_Thunderstorm Snow



> # calculate mean
> mean(accuracy)
[1] 0.7576316
> mean(precision_Rain)
[1] 0.7102237
> mean(Recall_Rain)
[1] 0.7729309
> mean(precision_Rain_Thunderstorm)
[1] 0.7763678
> mean(Recall_Rain_Thunderstorm)
[1] 0.7050854
> mean(precision_Snow)
[1] 0.8934333
> mean(Recall_Snow)
[1] 0.8886146
>
> # Compute 95% confidence interval on Accuracy using t-distribution
> AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
> # show the confidence interval [for Accuracy]
> mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err
[1] 0.7500795
[1] 0.7651837
>
```

```
> # summaries
> summary(accuracy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6316  0.7368  0.7632  0.7576  0.7763  0.8421 
> summary(precision_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5161  0.6544  0.7104  0.7102  0.7778  0.8788 
> summary(Recall_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6098  0.7273  0.7647  0.7729  0.8121  0.9677 
> summary(precision_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6053  0.7197  0.7690  0.7764  0.8278  0.9615 
> summary(Recall_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.4828  0.6535  0.7029  0.7051  0.7667  0.8621 
> summary(precision_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5455  0.8333  0.9000  0.8934  1.0000  1.0000 
> summary(Recall_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6667  0.8333  0.9000  0.8886  1.0000  1.0000 
> svm_tune <- tune(svm,
+                  Events~.,data=kcweather.Train,kernel='linear',ranges=list(cost=10^(-1:2), gamma=c(.5,1,2
)))
> summary(svm_tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation 

- best parameters:
 cost gamma
  100   0.5

- best performance: 0.2344828 

- Detailed performance results:
    cost gamma     error dispersion
1    0.1   0.5 0.2482759 0.08415175
2    1.0   0.5 0.2517241 0.06906124
3   10.0   0.5 0.2482759 0.07413570
4  100.0   0.5 0.2344828 0.09021341
5    0.1   1.0 0.2482759 0.08415175
6    1.0   1.0 0.2517241 0.06906124
7   10.0   1.0 0.2482759 0.07413570
8  100.0   1.0 0.2344828 0.09021341
9    0.1   2.0 0.2482759 0.08415175
10   1.0   2.0 0.2517241 0.06906124
11  10.0   2.0 0.2482759 0.07413570
12 100.0   2.0 0.2344828 0.09021341
```

Radial Kernel :

```
svmfit = svm(Events~.,data=kcweather.Train,kernel='radial',cost=100)
> summary(svmfit)
```

Execution:

```
Console ~/
+ }
> summary(svmfit)

Call:
svm(formula = Events ~ ., data = kcweather.Train, kernel = "radial", cost = 100)


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  radial 
       cost:  100 
      gamma:  0.1428571 

Number of Support Vectors:  146

 ( 61 66 19 )


Number of Classes:  3 

Levels: 
 Rain Rain_Thunderstorm Snow



> # calculate mean
> mean(accuracy)
[1] 0.7527632
> mean(precision_Rain)
[1] 0.7494105
> mean(Recall_Rain)
[1] 0.7471074
> mean(precision_Rain_Thunderstorm)
[1] 0.7261495
> mean(Recall_Rain_Thunderstorm)
[1] 0.7335187
> mean(precision_Snow)
[1] 0.8498081
> mean(Recall_Snow)
[1] 0.8399129
> 
> # Compute 95% confidence interval on Accuracy using t-distribution
> AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
> # show the confidence interval [for Accuracy]
> mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err
[1] 0.7434839
[1] 0.7620425
> 
> # summaries
> summary(accuracy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
```

Output results:

```
Call:
svm(formula = Events ~ ., data = kcweather.Train, kernel = "radial", cost = 100)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  100
      gamma:  0.1428571

Number of Support Vectors:  146

 ( 61 66 19 )


Number of Classes:  3

Levels:
 Rain Rain_Thunderstorm Snow



> # calculate mean
> mean(accuracy)
[1] 0.7527632
> mean(precision_Rain)
[1] 0.7494105
> mean(Recall_Rain)
[1] 0.7471074
> mean(precision_Rain_Thunderstorm)
[1] 0.7261495
> mean(Recall_Rain_Thunderstorm)
[1] 0.7335187
> mean(precision_Snow)
[1] 0.8498081
> mean(Recall_Snow)
[1] 0.8399129
>
> # Compute 95% confidence interval on Accuracy using t-distribution
> AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
> # show the confidence interval [for Accuracy]
> mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err
[1] 0.7434839
[1] 0.7620425
>
> # summaries
> summary(accuracy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.6316  0.7204  0.7500  0.7528  0.7796  0.8947
> summary(precision_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5714  0.7051  0.7534  0.7494  0.7876  0.9091
> summary(Recall_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.6047  0.6901  0.7432  0.7471  0.8000  0.9375
> summary(precision_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.4231  0.6756  0.7333  0.7261  0.8068  0.9615
> summary(Recall_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5312  0.6794  0.7333  0.7335  0.7863  0.9048
> summary(precision_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5000  0.7778  0.8750  0.8498  0.9245  1.0000
> summary(Recall_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5556  0.7500  0.8571  0.8399  0.9091  1.0000
> svm_tune <- tune(svm,
+                   Events~.,data=kcweather.Train,kernel='radial', ranges=list(cost=10^(-1:2), gamma=c(.5,1,
2)))
> summary(svm_tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.2413793

- Detailed performance results:
    cost gamma     error dispersion
1    0.1   0.5 0.3965517 0.11972797
2    1.0   0.5 0.2413793 0.08127664
3   10.0   0.5 0.2758621 0.06896552
4  100.0   0.5 0.3172414 0.06858131
5    0.1   1.0 0.5068966 0.12060753
6    1.0   1.0 0.2862069 0.07457990
7   10.0   1.0 0.3000000 0.07094850
8  100.0   1.0 0.3413793 0.06592934
9    0.1   2.0 0.5068966 0.12060753
10   1.0   2.0 0.3517241 0.09450485
```

```
11   10.0    2.0 0.3586207 0.10046792
12  100.0    2.0 0.3586207 0.08933039
```

Analysis Summary:

The number of support vectors is 146 for the cost function is 100.
The svm tune function results in best cost value as 1.

3. SVM chooses appropriate kernel (Default Kernel)

```
svmfit = svm(Events~.,data=kcweather.Train)
```

Execution :

```
Console ~/
> summary(svmfit)

Call:
svm(formula = Events ~ ., data = kcweather.Train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1428571

Number of Support Vectors:  183

 ( 91 22 70 )


Number of Classes:  3

Levels:
 Rain Rain_Thunderstorm Snow



> # calculate mean
> mean(accuracy)
[1] 0.7896053
> mean(precision_Rain)
[1] 0.7865571
> mean(Recall_Rain)
[1] 0.7877644
> mean(precision_Rain_Thunderstorm)
[1] 0.7620164
> mean(Recall_Rain_Thunderstorm)
[1] 0.765396
> mean(precision_Snow)
[1] 0.8858419
> mean(Recall_Snow)
[1] 0.8703902
>
> # Compute 95% confidence interval on Accuracy using t-distribution
> AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
> # show the confidence interval [for Accuracy]
> mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err
[1] 0.781539
[1] 0.7976715
>
> # summaries
> summary(accuracy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.6974  0.7632  0.7895  0.7896  0.8289  0.8947
> summary(precision_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5789  0.7440  0.7838  0.7866  0.8239  0.9394
```

Output Results:

```
>summary(svmfit)

Call:
svm(formula = Events ~ ., data = kcweather.Train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1428571

Number of Support Vectors:  183

 ( 91 22 70 )


Number of Classes:  3

Levels:
 Rain Rain_Thunderstorm Snow



> # calculate mean
> mean(accuracy)
[1] 0.7896053
> mean(precision_Rain)
[1] 0.7865571
```

```
> mean(Recall_Rain)
[1] 0.7877644
> mean(precision_Rain_Thunderstorm)
[1] 0.7620164
> mean(Recall_Rain_Thunderstorm)
[1] 0.765396
> mean(precision_Snow)
[1] 0.8858419
> mean(Recall_Snow)
[1] 0.8703902
> 
> # Compute 95% confidence interval on Accuracy using t-distribution
> AccySR_svm_err = qt(0.975, df = rep-1) * sd(accuracy) / sqrt(rep)
> # show the confidence interval [for Accuracy]
> mean(accuracy) - AccySR_svm_err; mean(accuracy) + AccySR_svm_err
[1] 0.781539
[1] 0.7976715
> 
> # summaries
> summary(accuracy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6974  0.7632  0.7895  0.7896  0.8289  0.8947 
> summary(precision_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5789  0.7440  0.7838  0.7866  0.8239  0.9394 
> summary(Recall_Rain)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6154  0.7500  0.7936  0.7878  0.8287  0.9429 
> summary(precision_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5938  0.7097  0.7692  0.7620  0.8157  0.9130 
> summary(Recall_Rain_Thunderstorm)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5714  0.7143  0.7742  0.7654  0.8276  0.9259 
> summary(precision_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5000  0.8333  0.9091  0.8858  1.0000  1.0000 
> summary(Recall_Snow)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.6000  0.8136  0.8889  0.8704  1.0000  1.0000 
> svm_tune <- tune(svm,
+                  Events~.,data=kcweather.Train, ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))
> summary(svm_tune)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation 

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.2172414 

- Detailed performance results:
    cost gamma     error dispersion
1    0.1   0.5 0.3793103 0.07795782
2    1.0   0.5 0.2172414 0.05403522
3   10.0   0.5 0.2275862 0.04047544
4  100.0   0.5 0.2448276 0.04128341
5    0.1   1.0 0.5000000 0.06944279
6    1.0   1.0 0.2620690 0.04654818
7   10.0   1.0 0.2517241 0.03998282
8  100.0   1.0 0.2517241 0.04612047
9    0.1   2.0 0.5034483 0.06934760
10   1.0   2.0 0.3275862 0.05917029
11  10.0   2.0 0.3379310 0.06858131
12 100.0   2.0 0.3310345 0.06337494
```

SVM Results:

Model	Kernel	Accuracy	Precision			Recall		
			Rain	Rain_Thunderstorm	Snow	Rain	Rain_Thunderstorm	Snow
SVM	Linear	0.7576316	0.7102237	0.7763678	0.8934333	0.7729309	0.7050854	0.8886146
	Radial	0.7527632	0.7494105	0.7261495	0.8498081	0.7471074	0.7335187	0.8399129
	Default	0.7896053	0.7865571	0.7620164	0.8858419	0.7877644	0.765396	0.8703902

SVM Analysis summary:

By the above results the SVM default (appropriate) model has higher accuracy when compared to other kernels.

LDA, QDA and KNN comparison with the SVM for the data set consisting 366 entries:

Model	Error	Accuracy	Precision			Recall		
			Rain	Rain_Thunderstorm	Snow	Rain	Rain_Thunderstorm	Snow
LDA	0.2519737	0.7480263	0.7523023	0.7237916	0.8021281	0.7087325	0.7499575	0.8945227
QDA	0.255	0.745	0.7498849	0.7184631	0.7964458	0.7016425	0.7264472	0.9515166

KNN(k=3)	0.847623	0.733815	0.7262863	0.6946528	0.8897852	0.7250768	0.6982034	0.8713335

Analysis Summary:

Hence, from above results when compared LDA, QDA, KNN to SVM the SVM has more accuracy.

3_2 Question:

Consider the time series on Milk production data milk-production(1).csv
it shows cow milk production per pound from 1962 to 1975.

a. Try at least three different values for window size with simple moving average (SMA) for forecasting

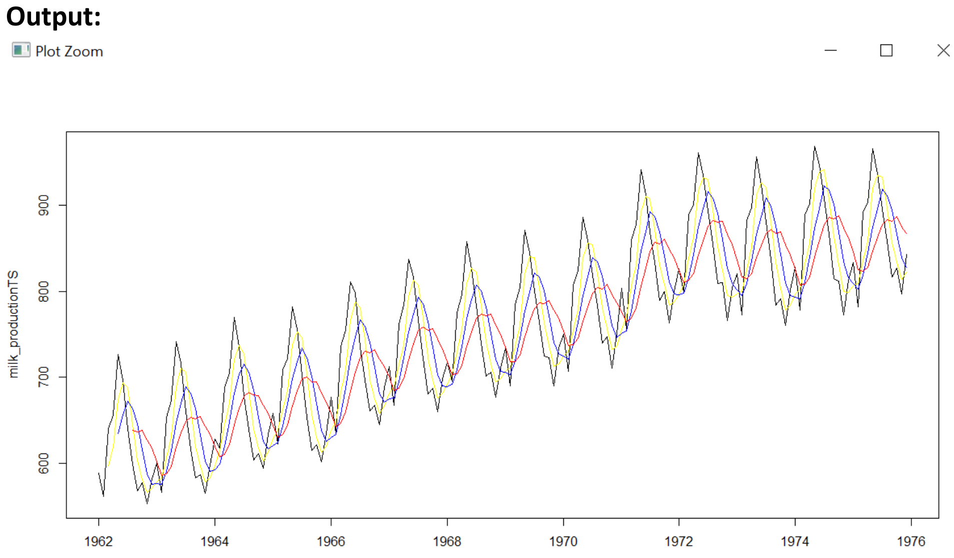
Solution:

```
> library(TTR)
> library(readr)
> milk_production <- read_csv("R/milk-production.csv")
Parsed with column specification:
cols (
Month = col_character(),
Pounds_per_Cow = col_integer()
)
> View(milk_production)
> milk_productionTS<-ts(milk_production$Pounds_per_Cow,frequency = 12,start=c(1962,1))
> plot.ts(milk_productionTS)
> sTS3 = SMA(milk_productionTS,n=3)
> sTS5 = SMA(milk_productionTS,n=5)
> sTS8 = SMA(milk_productionTS,n=8)
> lines(milk_productionTS, col="black")
> lines(sTS3, col="yellow")
> lines(sTS5, col="blue")
> lines (sTS8, col="red")
```

Execution:

```
>
> library(TTR)
> library(readr)
> milk_production <- read_csv("R/milk-production.csv")
Parsed with column specification:
cols(
  Month = col_character(),
  Pounds_per_Cow = col_integer()
)
> View(milk_production)
> milk_productionTS<-ts(milk_production$Pounds_per_Cow,frequency = 12,start=c(1962,1))
> plot.ts(milk_productionTS)
> sTS3 = SMA(milk_productionTS,n=3)
> sTS5 = SMA(milk_productionTS,n=5)
> sTS8 = SMA(milk_productionTS,n=8)
> lines(milk_productionTS, col="black")
> lines(sTS3, col="yellow")
> lines(sTS5, col="blue")
> lines(sTS8, col="red")
```

Output:



b. Apply exponential moving average using HoltWinters for forecasting

Solution:

```
> library(TTR)
```

> library(readr)
> milk_production <- read_csv("R/milk-production.csv")
Parsed with column specification:
cols (
Month = col_character(),
Pounds_per_Cow = col_integer ()
)
> View(milk_production)
> milk_productionTS<-ts(milk_production$Pounds_per_Cow,frequency = 12,start=c(1962,1))
> plot.ts(milk_productionTS)

Simple exponential smoothing without seasonal and trend component

> sHW_without = HoltWinters(milk_productionTS,beta=FALSE, gamma=FALSE)
> sHW_without
Holt-Winters exponential smoothing without trend and without seasonal component.

Call:
HoltWinters(x = milk_productionTS, beta = FALSE, gamma = FALSE)

Smoothing parameters:
alpha: 0.9999339
beta : FALSE
gamma: FALSE

Coefficients:
[,1]
a 842.997

Execution:

```
> sHW_without = HoltWinters(milk_productionTS,beta=FALSE, gamma=FALSE)
> sHW_without
Holt-Winters exponential smoothing without trend and without seasonal component.

Call:
HoltWinters(x = milk_productionTS, beta = FALSE, gamma = FALSE)

Smoothing parameters:
 alpha: 0.9999339
 beta : FALSE
 gamma: FALSE

Coefficients:
     [,1]
a 842.997
> |
```

- **Holt-winters exponential smoothing with trend and additive seasonal component**

> sHW_with = HoltWinters(milk_productionTS)
> sHW_with
Holt-Winters exponential smoothing with trend and additive seasonal component.

Call:
HoltWinters(x = milk_productionTS)

Smoothing parameters:
alpha: 0.68933
beta: 0
gamma: 0.8362592

Coefficients:

```
        [,1]
a   885.775547
b     1.278118
s1  -16.743296
s2  -59.730034
s3   47.492731
s4   56.203890
s5  115.537545
s6   84.554817
s7   39.580306
s8   -4.702033
s9  -54.554684
s10 -51.582594
s11 -85.953466
s12 -42.907363
```

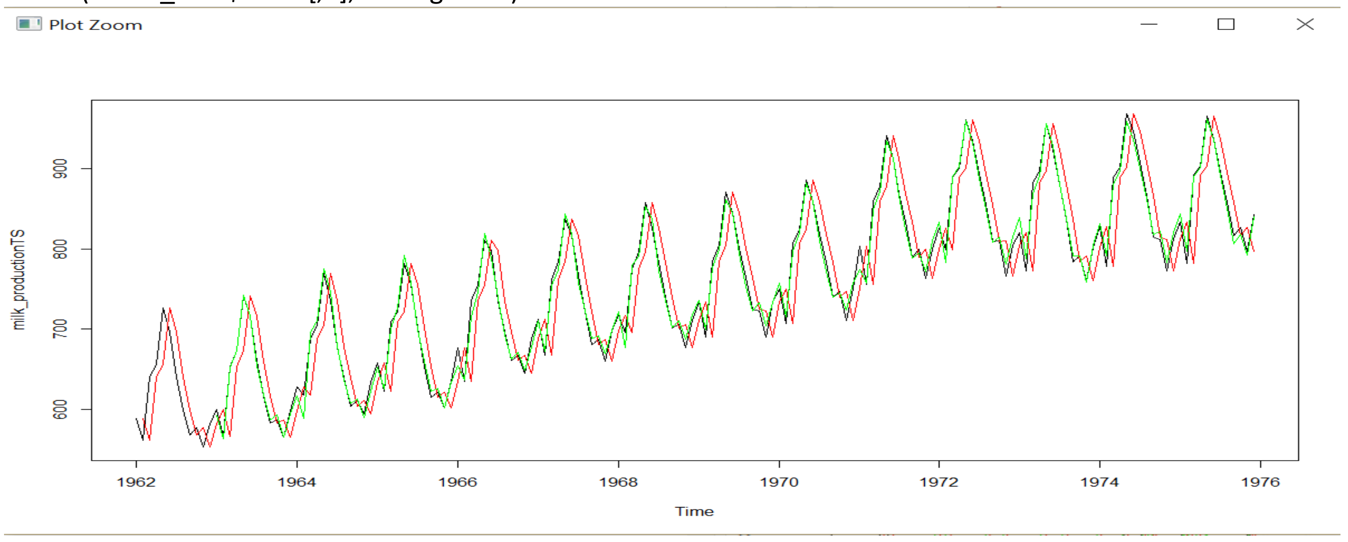
Execution:

```
> sHW_with = HoltWinters(milk_productionTS)
> sHW_with
Holt-Winters exponential smoothing with trend and additive seasonal component.

Call:
HoltWinters(x = milk_productionTS)

Smoothing parameters:
 alpha: 0.68933
 beta : 0
 gamma: 0.8362592

Coefficients:
          [,1]
a    885.775547
b      1.278118
s1   -16.743296
s2   -59.730034
s3    47.492731
s4    56.203890
s5   115.537545
s6    84.554817
s7    39.580306
s8    -4.702033
s9   -54.554684
s10  -51.582594
s11  -85.953466
s12  -42.907363
> |
```

Output Plot :

>lines(sHW_without$fitted[,1], col= "red")

> lines(sHW1_with$fitted[,1], col= "green")



c. For the above, discuss how the forecasting differs in terms of MAD and MFE and why one approach or the other is better.

Solution:

MAD:

MAD is Mean absolute deviation which is an indication for the absolute size of the errors.

For any n time periods where we have actual demand and forecast values

$$MAD = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

MFE:

MFE is Mean forecast error which is a measure of forecast model bias.

For any n time periods where we have actual demand and forecast values.

Initially the ideal value would be =0 and

the model tends to Under - forecast when MFE>0

the model tends to over -forecast when MFE<0

$$MFE = \frac{\sum_{i=1}^{n} (e_i)}{n}$$

Thus, the MAD is used to indicate the absolute size of the error and MFE measures of forecast model bias.

Source:

```
> plot.ts(milk_productionTS)
> sTS3 = SMA(milk_productionTS,n=3)
> error = milk_productionTS -  sTS3
> MFE3 = (sum(error,na.rm = TRUE)/length(error))
> MFE3
[1] 1.531746
> MAD3 = (sum(abs(error),na.rm = TRUE)/length(error))
> MAD3
[1] 29.69841
> sTS5 = SMA(milk_productionTS,n=5)
> error = milk_productionTS -  sTS5
> MAD5 = (sum(abs(error),na.rm = TRUE)/length(error))
> MFE5 = (sum(error,na.rm = TRUE)/length(error))
> MAD5
[1] 46.67976
```

> MFE5
[1] 2.355952
> sTS8 = SMA(milk_productionTS,n=8)
> error = milk_productionTS - sTS8
> MAD8 = (sum(abs(error),na.rm = TRUE)/length(error))
> MFE8 = (sum(error,na.rm = TRUE)/length(error))
> MFE8
[1] 3.598958
> MAD8
[1] 55.39062

Execution :

```
> plot.ts(milk_productionTS)
> sTS3 = SMA(milk_productionTS,n=3)
> error = milk_productionTS -  sTS3
> MFE3 = (sum(error,na.rm = TRUE)/length(error))
> MFE3
[1] 1.531746
> MAD3 = (sum(abs(error),na.rm = TRUE)/length(error))
> MAD3
[1] 29.69841
> sTS5 = SMA(milk_productionTS,n=5)
> error = milk_productionTS -  sTS5
> MAD5 = (sum(abs(error),na.rm = TRUE)/length(error))
> MFE5 = (sum(error,na.rm = TRUE)/length(error))
> MAD5
[1] 46.67976
> MFE5
[1] 2.355952
> sTS8 = SMA(milk_productionTS,n=8)
> error = milk_productionTS -  sTS8
> MAD8 = (sum(abs(error),na.rm = TRUE)/length(error))
> MFE8 = (sum(error,na.rm = TRUE)/length(error))
> MFE8
[1] 3.598958
> MAD8
[1] 55.39062
```

Analysis summary:

	MFE	MAD
N=3	1.531746	29.69841
N=5	2.355952	46.67976
N=8	3.598958	55.39062

For N=3, Model watches out for marginally under-forecast, with a normal absolute error of 29.69841units
For N=5, Model watches out for marginally under-forecast, with a normal absolute error of 46.67976units
For N=8, Model watches out for marginally under-forecast, with a normal absolute error of 55.39062units