Homework 2: Data Pre-processing & Exponential Smoothing

Group 4 - Subhalaxmi Rout, Kenan Sooklall, Devin Teran, Christian Thieme, Leo Yi

6/12/2021

## Applied Predictive Modeling Chapter 3 - Data Pre-processing

### 3.1

**The UC Irvine Machine Learning Repository contains a data set related to glass identification. The data consist of 214 glass samples labeled as one of seven class categories. There are nine predictors, including the refractive index and percentages of eight elements: Na, Mg, Al, Si, K, Ca, Ba, and Fe. The data can be accessed via:**

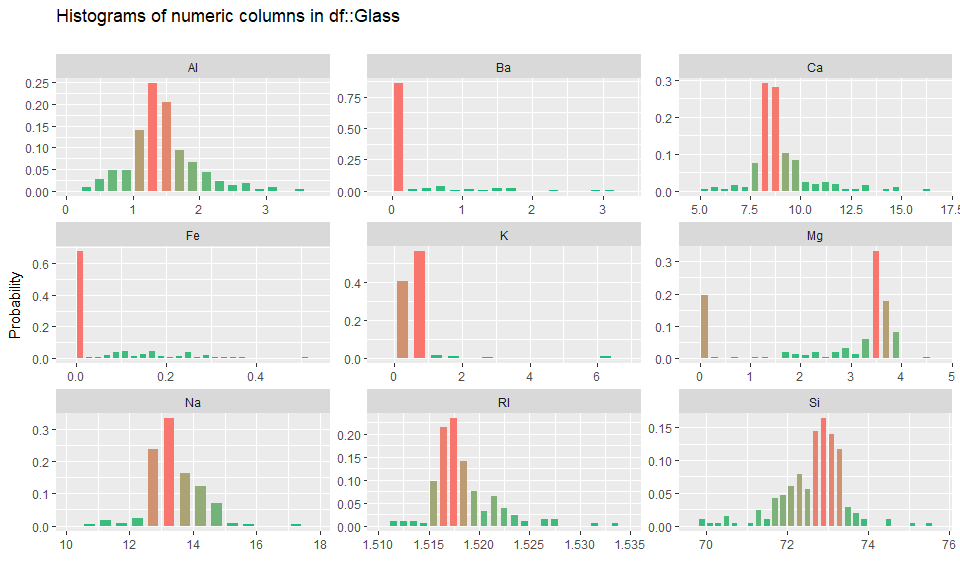
data(Glass)  
glimpse(Glass)

## Rows: 214  
## Columns: 10  
## $ RI <dbl> 1.52101, 1.51761, 1.51618, 1.51766, 1.51742, 1.51596, 1.51743,...  
## $ Na <dbl> 13.64, 13.89, 13.53, 13.21, 13.27, 12.79, 13.30, 13.15, 14.04,...  
## $ Mg <dbl> 4.49, 3.60, 3.55, 3.69, 3.62, 3.61, 3.60, 3.61, 3.58, 3.60, 3....  
## $ Al <dbl> 1.10, 1.36, 1.54, 1.29, 1.24, 1.62, 1.14, 1.05, 1.37, 1.36, 1....  
## $ Si <dbl> 71.78, 72.73, 72.99, 72.61, 73.08, 72.97, 73.09, 73.24, 72.08,...  
## $ K <dbl> 0.06, 0.48, 0.39, 0.57, 0.55, 0.64, 0.58, 0.57, 0.56, 0.57, 0....  
## $ Ca <dbl> 8.75, 7.83, 7.78, 8.22, 8.07, 8.07, 8.17, 8.24, 8.30, 8.40, 8....  
## $ Ba <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ Fe <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.26, 0.00, 0.00, 0.00, 0.11, 0....  
## $ Type <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...

**Using visualizations, explore the predictor variables to understand their distributions as well as the relationships between predictors.**

Since our predictors are continuous variables, we’ll use histograms to understand their distributions:

inspectdf::inspect\_num(Glass) %>%   
 show\_plot()

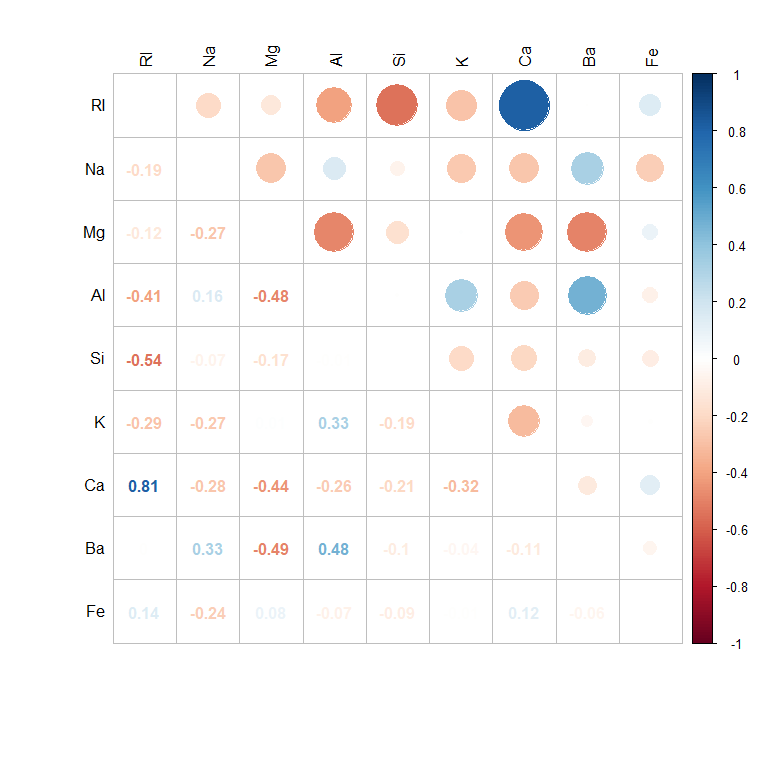


We note the following:

* Al: Appears to have a fairly normal distribution, slightly right skewed
* Ba: Over 75% of this distribution is 0 values. We note some potential outliers that will need to be investigated.
* Ca: The distribution has two values that make up over 60% of the distribution. The distribution is right skewed.
* Fe: Similar to Ba, 65% of the values are 0. We note the presence of potential outliers.
* K: Two values make up ~90% of the distribution. We note the presence of potential outliers.
* Mg: The distribution is bi-modal, which may indicate sub-populations. 20% of the distribution is 0s. The remainder of the distribution is left skewed.
* Na: Right skewed.Values before 12.5 are rare.
* Rl: The distribution is right skewed. Values before 1.515 are rare.
* Si: The distribution is left skewed. Values after 74 are rare.

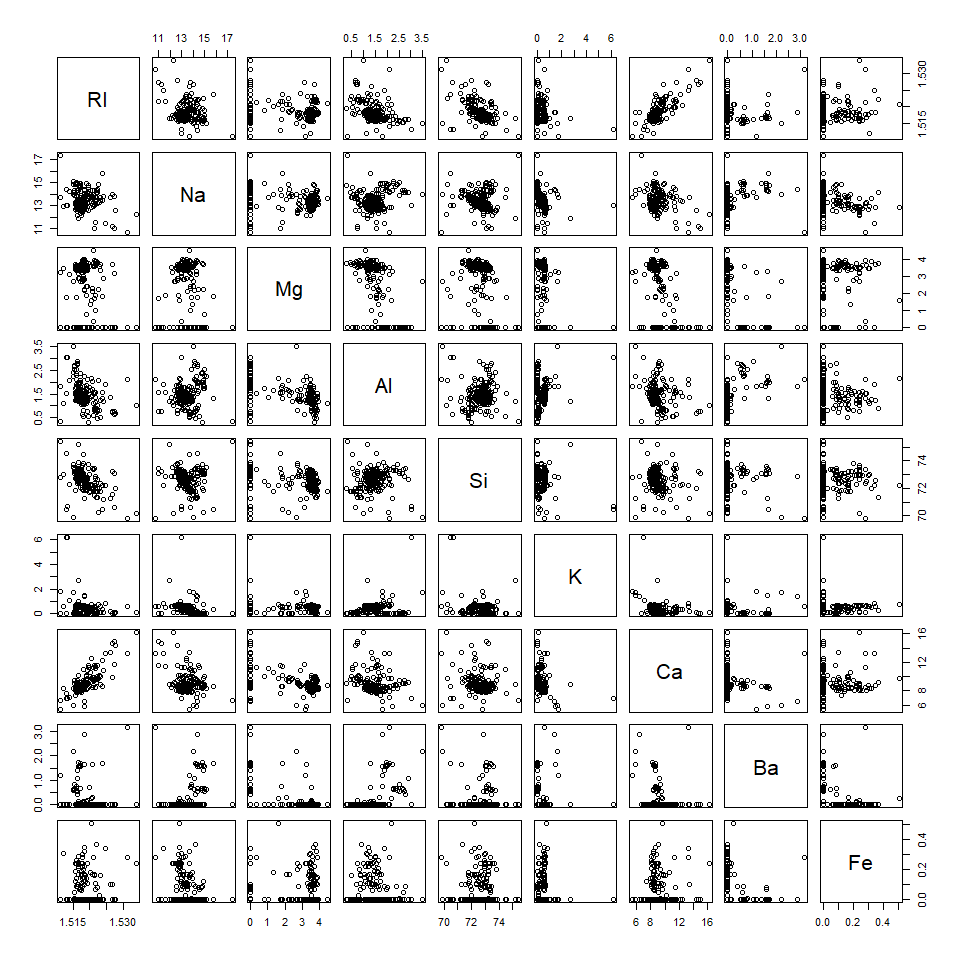
Now we turn our attention to the relationship between predictors. We’ll review a correlation plot to assess the relationship between predictors:

correlation <- cor(Glass %>% select(-c('Type')))  
corrplot.mixed(correlation, tl.col = 'black', tl.pos = 'lt')



Correlation plots can be deceiving if used without looking at the visual relationship between the data. Let’s create some scatter plots of the relationships as well.

pairs(Glass %>% dplyr::select\_if(is.numeric))



We note very few meaningful relationships between the predictors despite what the correlation plot shows:

* There appears to be a loosely negative relationship between Ri and Al and Ri and Si. There appears to be a positive relationship between Ri and Ca.

**Do there appear to be any outliers in the data? Are any predictors skewed?**

We noted skew and outliers in our discussion above of the histograms.

**Are there any relevant transformations of one or more predictors that might improve the classification model?**

This data would benefit from centering and scaling as they are all on different scales. Additionally, several of the skewed variables may benefit from a log or similar transformation. We recommend using the Box-Cox approach to determine the necessary transformation for each of the skewed variables.

### 3.2

**The soybean data can also be found at the UC Irvine Machine Learning Repository. Data were collected to predict disease in 683 soybeans. The 35 predictors are mostly categorical and include information on the environmental conditions (e.g., temperature, precipitation) and plant conditions (e.g., left spots, mold growth). The outcome labels consist of 19 distinct classes. The data can be loaded via:**

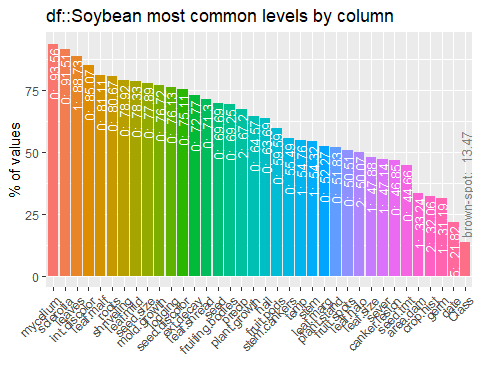
data("Soybean")  
glimpse(Soybean)

## Rows: 683  
## Columns: 36  
## $ Class <fct> diaporthe-stem-canker, diaporthe-stem-canker, diapo...  
## $ date <fct> 6, 4, 3, 3, 6, 5, 5, 4, 6, 4, 6, 4, 3, 6, 6, 5, 6, ...  
## $ plant.stand <ord> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ precip <ord> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, ...  
## $ temp <ord> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, ...  
## $ hail <fct> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, ...  
## $ crop.hist <fct> 1, 2, 1, 1, 2, 3, 2, 1, 3, 2, 1, 1, 1, 3, 1, 3, 0, ...  
## $ area.dam <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 2, 3, 3, 3, 2, ...  
## $ sever <fct> 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, ...  
## $ seed.tmt <fct> 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, ...  
## $ germ <ord> 0, 1, 2, 1, 2, 1, 0, 2, 1, 2, 0, 1, 0, 0, 1, 2, 0, ...  
## $ plant.growth <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...  
## $ leaves <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...  
## $ leaf.halo <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ leaf.marg <fct> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...  
## $ leaf.size <ord> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...  
## $ leaf.shread <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ leaf.malf <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ leaf.mild <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ stem <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...  
## $ lodging <fct> 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, ...  
## $ stem.cankers <fct> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, ...  
## $ canker.lesion <fct> 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, ...  
## $ fruiting.bodies <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...  
## $ ext.decay <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...  
## $ mycelium <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ int.discolor <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, ...  
## $ sclerotia <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ...  
## $ fruit.pods <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ fruit.spots <fct> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ...  
## $ seed <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ mold.growth <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ seed.discolor <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ seed.size <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ shriveling <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  
## $ roots <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

**Investigate the frequency distributions for the categorical predictors. Are any of the distributions degenerate in the ways discussed earlier in this chapter?**

We’ll look at a chart for all of the factor variables to see the % of the most common level within the factor:

# factor analysis. visualize most common factors  
inspectdf::inspect\_imb(Soybean) %>% show\_plot()



In looking at the chart, we can see that near zero variance may be an issue for several features within our dataset. For example, mycelium, sclerotia, leaves, and int.discolor all have 85%+ of their values with the same level. As discussed in the reading, this can be problematic. Let’s run the nearZeroVar function from the caret package to see which variables meet the criteria:

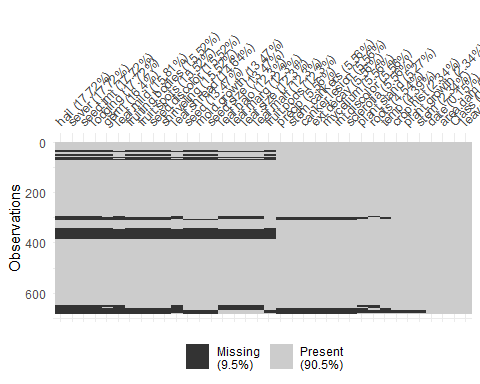
nzv <- caret::nearZeroVar(Soybean, names = TRUE, saveMetrics = TRUE)  
nzv %>% filter(nzv == TRUE)

## freqRatio percentUnique zeroVar nzv  
## leaf.mild 26.75 0.4392387 FALSE TRUE  
## mycelium 106.50 0.2928258 FALSE TRUE  
## sclerotia 31.25 0.2928258 FALSE TRUE

We should consider removing the 3 variables above before modeling.

**Roughly 18% of the data are missing. Are there particular predictors that are more likely to be missing? Is the pattern of missing data related to the classes?**

visdat::vis\_miss(Soybean, sort\_miss = TRUE)



9.5% of the data have NAs. Several of the predictors have more than 15% of their values missing such as hail, sever, seed.tmt, and lodging. Additionally, the missing values appear to have some pattern as the nulls are not random.

Soybean %>%  
 filter(!complete.cases(.)) %>%   
 count(Class) %>%   
 mutate('% missing' = n/dim(Soybean)[1]) %>%   
 arrange(desc(`% missing`))

## Class n % missing  
## 1 phytophthora-rot 68 0.09956076  
## 2 2-4-d-injury 16 0.02342606  
## 3 diaporthe-pod-&-stem-blight 15 0.02196193  
## 4 cyst-nematode 14 0.02049780  
## 5 herbicide-injury 8 0.01171303

From the missing 18% of our data, we can see that more than half of it is related to the phytophthora-rot class. The percentages show us what percent of the total data is missing, but lets investigate how much of each class is missing:

complete <- Soybean %>%  
 count(Class) %>%  
 rename(total = n)  
  
missing <- Soybean %>%  
 filter(!complete.cases(.)) %>%   
 count(Class) %>%  
 rename(missing = n)  
  
final <- missing %>%   
 left\_join(complete) %>%   
 mutate('% missing' = missing/total)

## Joining, by = "Class"

final

## Class missing total % missing  
## 1 2-4-d-injury 16 16 1.0000000  
## 2 cyst-nematode 14 14 1.0000000  
## 3 diaporthe-pod-&-stem-blight 15 15 1.0000000  
## 4 herbicide-injury 8 8 1.0000000  
## 5 phytophthora-rot 68 88 0.7727273

It appears that several classes are missing at least one cell in each observation (Albeit, they may be a fairly small part of the population).Phytophthora-rot is missing data in 77% of its observations and is a larger percentage of the total population.

**Develop a strategy for handling missing data, either by eliminating predictors or imputation.**

In looking at the the proportions and combinations of missing values as well as understanding that half of our missing values come from one class, we have chosen to impute missing values using predictive mean matching (pmm). Predictive mean matching calculates the predicted value for our target variable, and, for missing values, forms a small set of “candidate donors” from the complete cases that are closest to the predicted value for our missing entry. Donors are then randomly chosen from candidates and imputed where values were once missing. This method is similar in theory to knn, however, it is often better at imputation. We can utilize pmm through the mice library.

Sb <- mice(data = Soybean, m = 1, method = "pmm", printFlag=F, seed = 500)  
Sb <- mice::complete(Sb, 1)  
  
colSums(is.na(Sb))

## Class date plant.stand precip temp   
## 0 0 0 0 0   
## hail crop.hist area.dam sever seed.tmt   
## 0 0 0 0 0   
## germ plant.growth leaves leaf.halo leaf.marg   
## 0 0 0 0 0   
## leaf.size leaf.shread leaf.malf leaf.mild stem   
## 0 0 0 0 0   
## lodging stem.cankers canker.lesion fruiting.bodies ext.decay   
## 0 0 0 0 0   
## mycelium int.discolor sclerotia fruit.pods fruit.spots   
## 0 0 0 0 0   
## seed mold.growth seed.discolor seed.size shriveling   
## 0 0 0 0 0   
## roots   
## 0

In the output above, we can see that there are no longer missing values in our dataset.

## Forecasting: Principles and Practice - Exponential smoothing

### 7.1

**Consider the pigs series — the number of pigs slaughtered in Victoria each month.**

pigs <- fma::pigs

**Use the ses() function in R to find the optimal values of and , and generate forecasts for the next four months.**

Optimal values can be found by looking at the model element of the forecast object.

simple <- ses(pigs, h = 4)  
simple$model

## Simple exponential smoothing   
##   
## Call:  
## ses(y = pigs, h = 4)   
##   
## Smoothing parameters:  
## alpha = 0.2971   
##   
## Initial states:  
## l = 77260.0561   
##   
## sigma: 10308.58  
##   
## AIC AICc BIC   
## 4462.955 4463.086 4472.665

In the code above, we generated the next 4 months of predictions. We can access them by calling mean on our forecast object simple.

simple$mean

## Sep Oct Nov Dec  
## 1995 98816.41 98816.41 98816.41 98816.41

**Compute a 95% prediction interval for the first forecast using where s is the standard deviation of the residuals. Compare your interval with the interval produced by R.**

prediction <- simple$mean[1]   
pred\_interval <- 1.96 \* (sd(simple$residuals))  
upper <- prediction + pred\_interval  
lower <- prediction - pred\_interval  
  
print(c(lower, upper))

## [1] 78679.97 118952.84

We calculated the prediction interval in the code chunk above and got an interval of (78,679.97, 118,952.84). We’ll now look at the interval calculated by the forecast object:

Upper bound:

simple$upper[1,2]

## 95%   
## 119020.8

Lower bound:

simple$lower[1,2]

## 95%   
## 78611.97

We can see that the interval is very close. It’s within ~68 on each side.

### 7.2

**Write your own function to implement simple exponential smoothing. The function should take arguments y (the time series), alpha (the smoothing parameter alpha) and level (the initial level lambda). It should return the forecast of the next observation in the series. Does it give the same forecast as ses()?**

simple\_exponential\_smoothing <- function(y, alpha, level) {  
   
 current\_level <- level  
   
 for (i in 1:length(y)){  
 current\_level <- (alpha \* y[i]) + (1 - alpha) \* current\_level  
 }  
   
 return(current\_level)  
   
}  
  
simple\_exponential\_smoothing(pigs, simple$model$par[[1]], simple$model$par[[2]])

## [1] 98816.41

Our function returns 98,816.41. Now let’s see what R returns:

simple$mean[1]

## [1] 98816.41

The values are identical.

### 7.3

**Modify your function from the previous exercise to return the sum of squared errors rather than the forecast of the next observation. Then use the optim() function to find the optimal values of alpha and lambda 0. Do you get the same values as the ses() function?**

*This* [*article*](https://www.magesblog.com/post/2013-03-12-how-to-use-optim-in-r/) *was helpful in solving this question.*

We’ll reconfigure our function to calculate the sum of squared errors and then use the optim function to find the optimal values of alpha and lambda:

sum\_of\_squared\_error <- function(y, par = c(alpha, level)){  
 e <- 0  
 SSE <- 0  
 alpha <- par[1]  
 current\_level <- par[2]  
 return\_value <- current\_level  
   
 for (i in 1:length(y)){  
 e <- y[i] - current\_level  
 SSE <- SSE + e^2  
   
 current\_level <- alpha \* y[i] + (1 - alpha) \* current\_level  
 }  
 return(SSE)  
}  
  
results <- optim(y = pigs, par = c(0.5, pigs[1]), fn = sum\_of\_squared\_error)  
results$par

## [1] 2.990081e-01 7.637927e+04

Now, that we have our optimized values, let’s compare these to what the ses function returns:

simple$model$par

## alpha l   
## 2.971488e-01 7.726006e+04

Our values are not exactly the same, but they are very, very close.