

IS624 - Assignment 1

James Quacinella

06/13/2015

Code

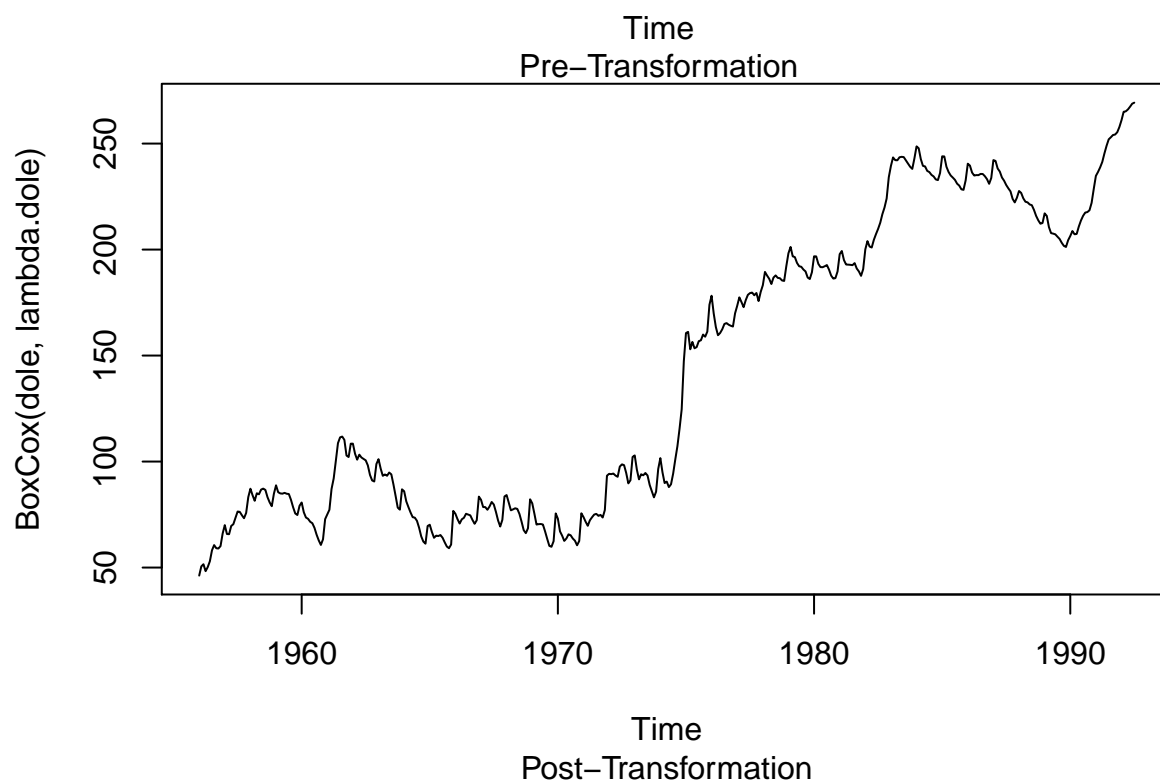
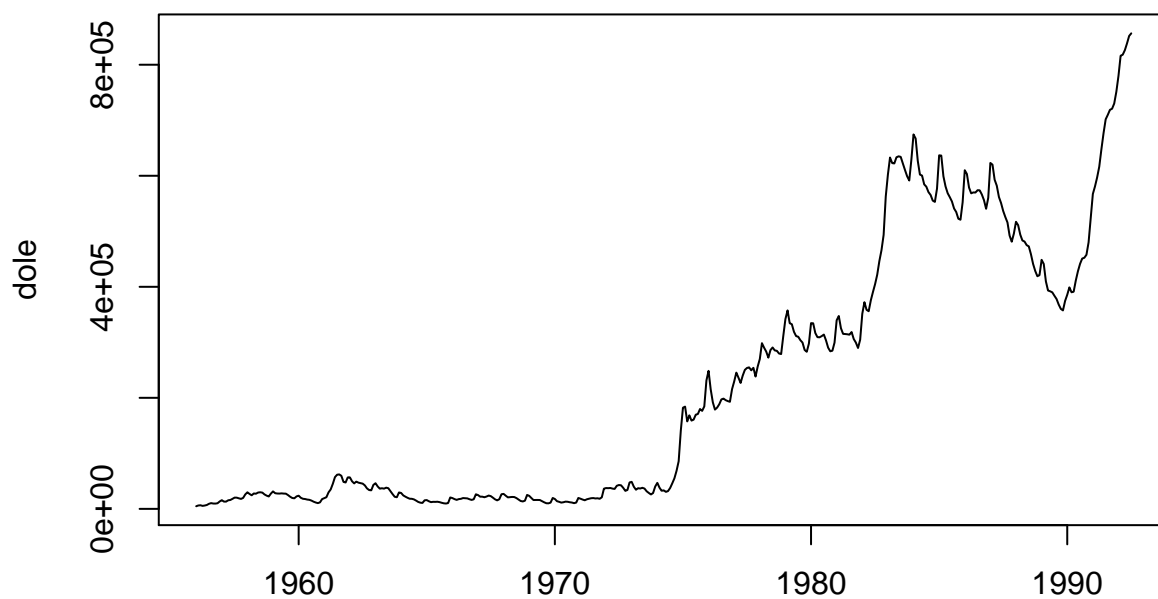
The official R code for this homework is located on github.com. All the code below is copied from the problem functions in that file.

Question 2.1a)

The following code produces two graphs of the pre and post transformation of the monthly total of people on unemployed benefits in Australia. The data looks skewed to me, and by recommendation of the first textbook, I would apply this transformation. Note that the y-scale is a lot smaller with the transform, which is true across all problems in this question.

```
#' Plot the Monthly total of people on unemployed benefits in Australia (January 1956-July 1992).
lambda.dole <- BoxCox.lambda(dole) # 0.3290922
plot(dole, main="Monthly total of people on unemployed benefits in Australia", sub="Pre-Transformation",
plot(BoxCox(dole, lambda.dole), sub="Post-Transformation")
```

Monthly total of people on unemployed benefits in Australia

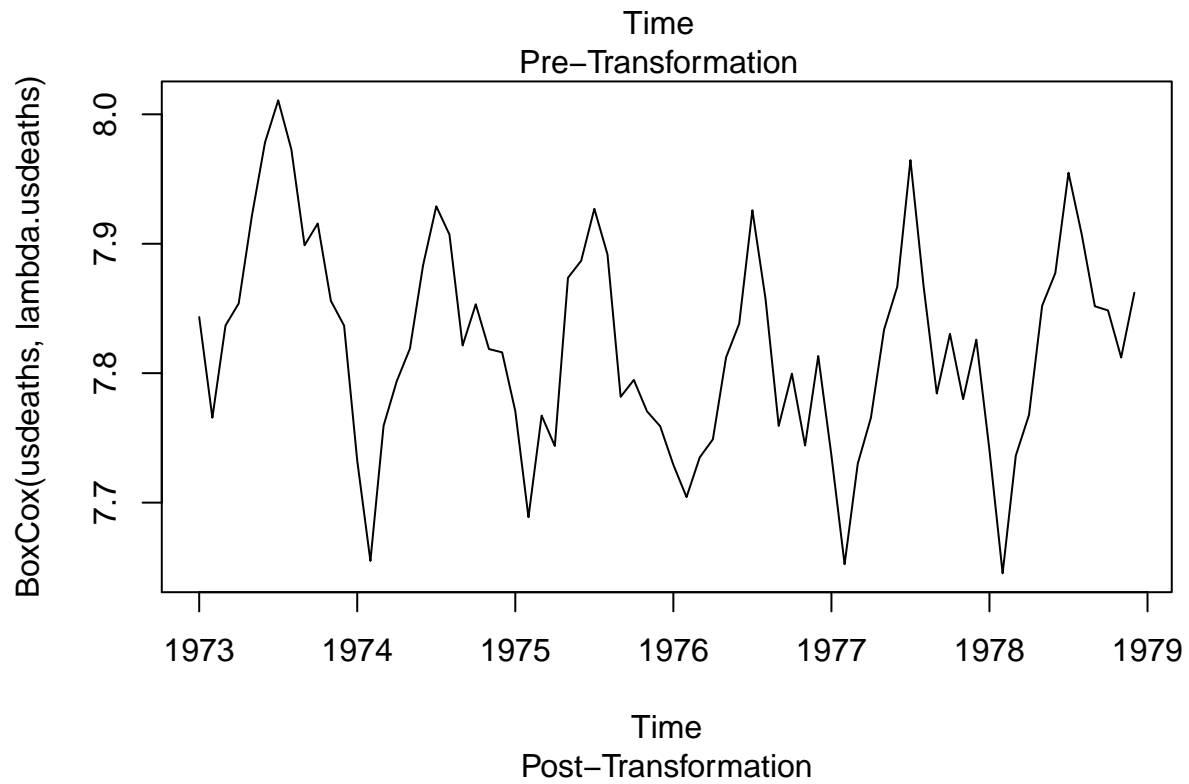
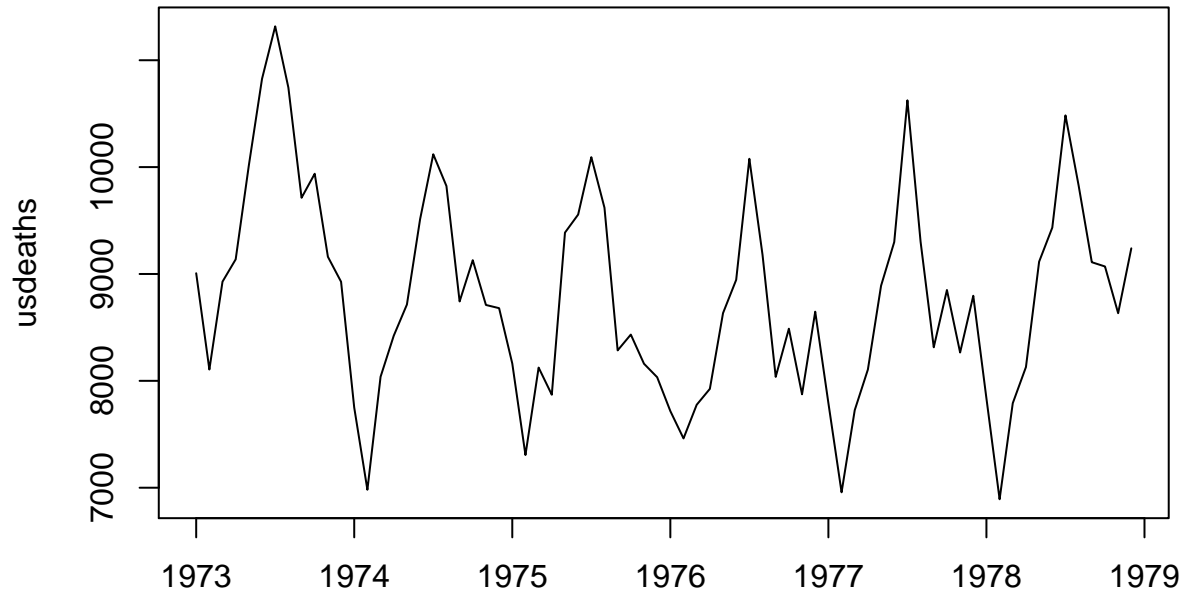


Question 2.1b)

The following code produces two graphs of the pre and post transformation of the monthly total of accidental deaths in the United States. The lambda value is small, and according to the text, since the seasonality already looks to be the same across the data, this transformation is probably not needed (“A good value of lambda is one which makes the size of the seasonal variation about the same across the whole series”). I have plotted both either way.

```
#' Monthly total of accidental deaths in the United States (January 1973-December 1978).  
lambda.usdeaths <- BoxCox.lambda(usdeaths) # -0.03363775  
plot(usdeaths, main="Monthly total of accidental deaths in the United States", sub="Pre-Transformation")  
plot(BoxCox(usdeaths, lambda.usdeaths), sub="Post-Transformation")
```

Monthly total of accidental deaths in the United States

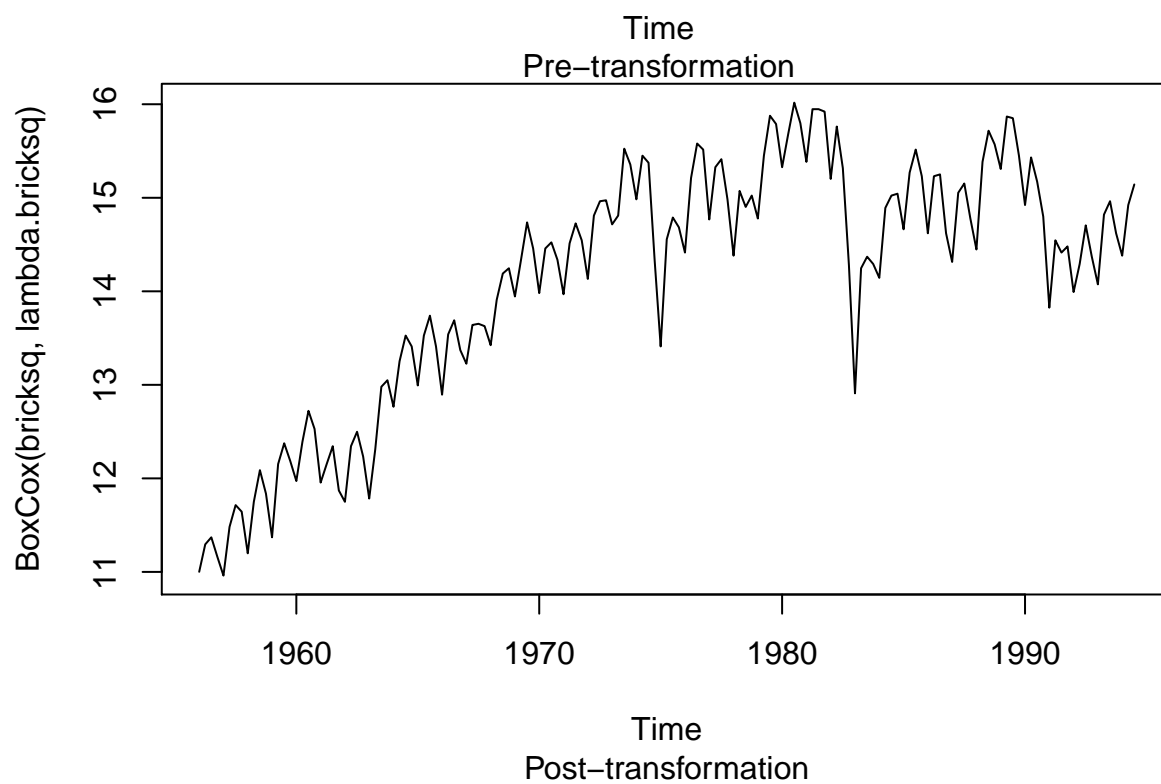
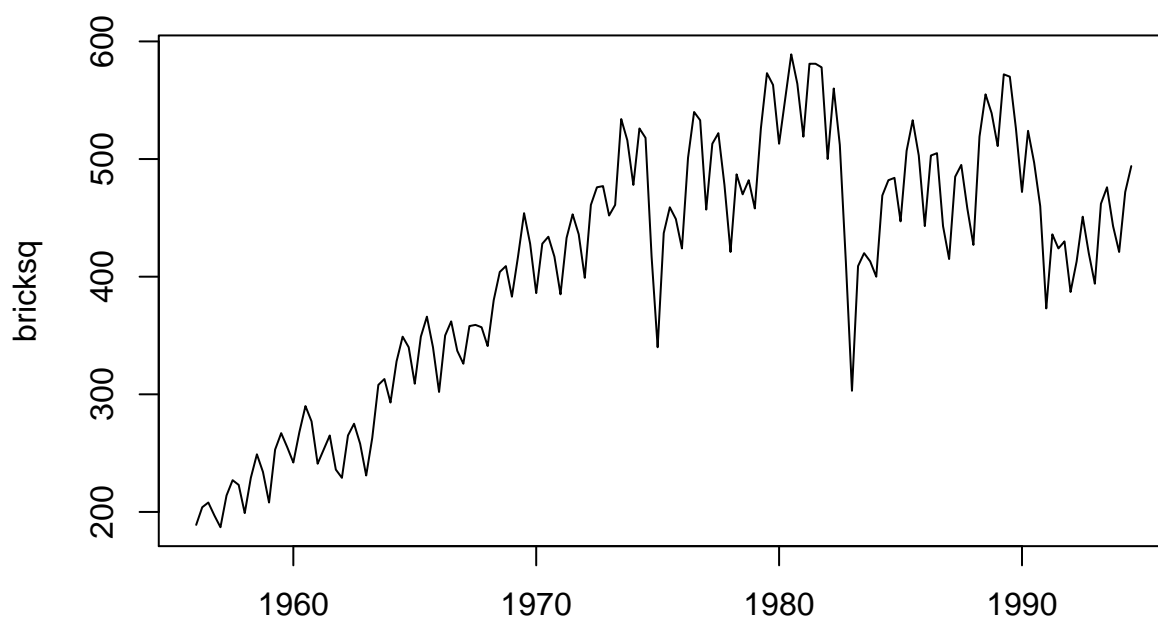


Question 2.1c)

The following code produces two graphs of the pre and post transformation of the Quarterly production of bricks. The shape of the data does not change significantly, but like previous portions of the question, the y-scale is smaller.

```
#' Quarterly production of bricks (in millions of units) at Portland, Australia (March 1956-September 1960)  
lambda.bricksq <- BoxCox.lambda(bricksq) # 0.2548929  
plot(bricksq, main="Quarterly production of bricks", sub="Pre-transformation")  
plot(BoxCox(bricksq, lambda.bricksq), sub="Post-transformation")
```

Quarterly production of bricks

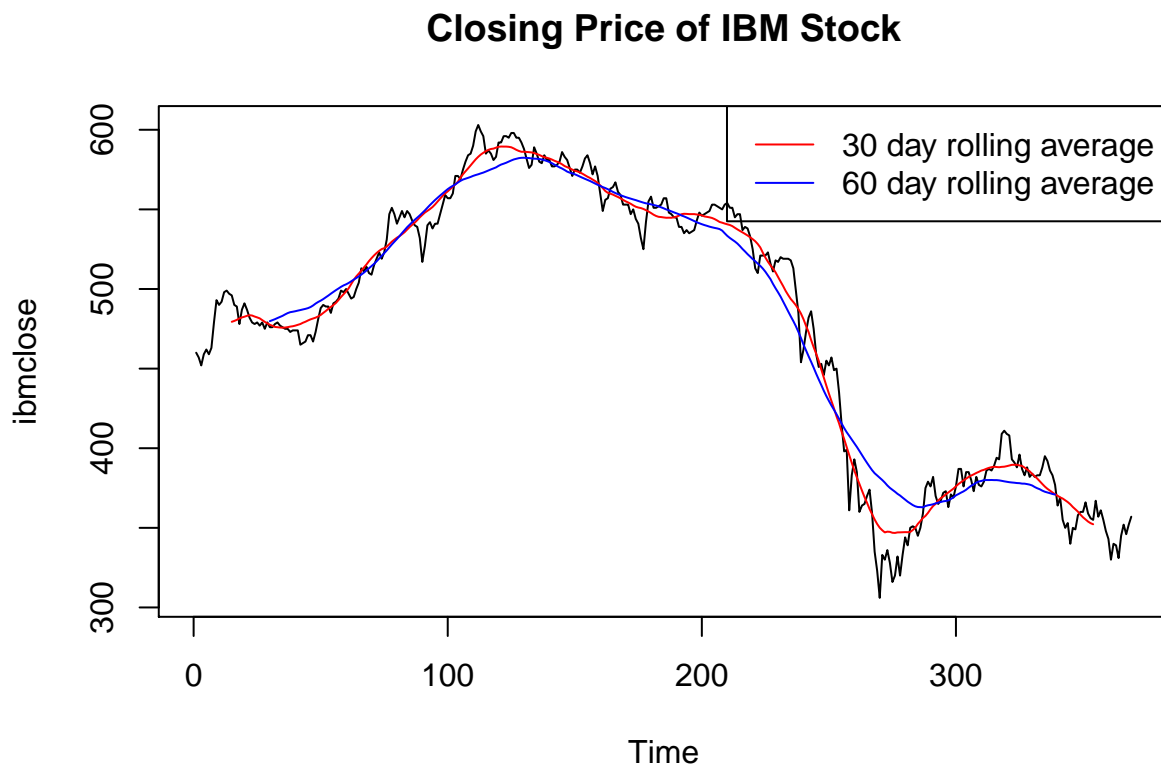


Question 2.3

Question 2.3a)

To become familiar with the `ibmclose` data set, here we plot the time series with some running averages. The big dip in price is going to give us trouble with our simplistic forecasting. The data is not in the right format for doing seasonal or month graphs (see next question).

```
#' Produce some plots of the data in order to become familiar with it.  
plot(ibmclose, main="Closing Price of IBM Stock")  
lines(rollmean(ibmclose, k=30, fill=NA), col="red")  
lines(rollmean(ibmclose, k=60, fill=NA), col="blue")  
legend("topright", lty=1, col=c("red", "blue"), legend=c("30 day rolling average", "60 day rolling average"))
```



Question 2.3b)

The `ibmclose` data is split into a 300-day training set and a 69-day test set:

```
ibmclose.training <- window(ibmclose, start=1, end=300)  
ibmclose.test <- window(ibmclose, start=301)
```

Question 2.3c)

Note that the blue prediction is for the mean prediction, and looks different due to using `plot()` versus `lines()`. Also note that both naive methods overlap their predictions. Visually inspecting the predictions, drift is clearly the most accurate; the mean is so far away as it does not account well for the bug dip in value around $t=250$.

```

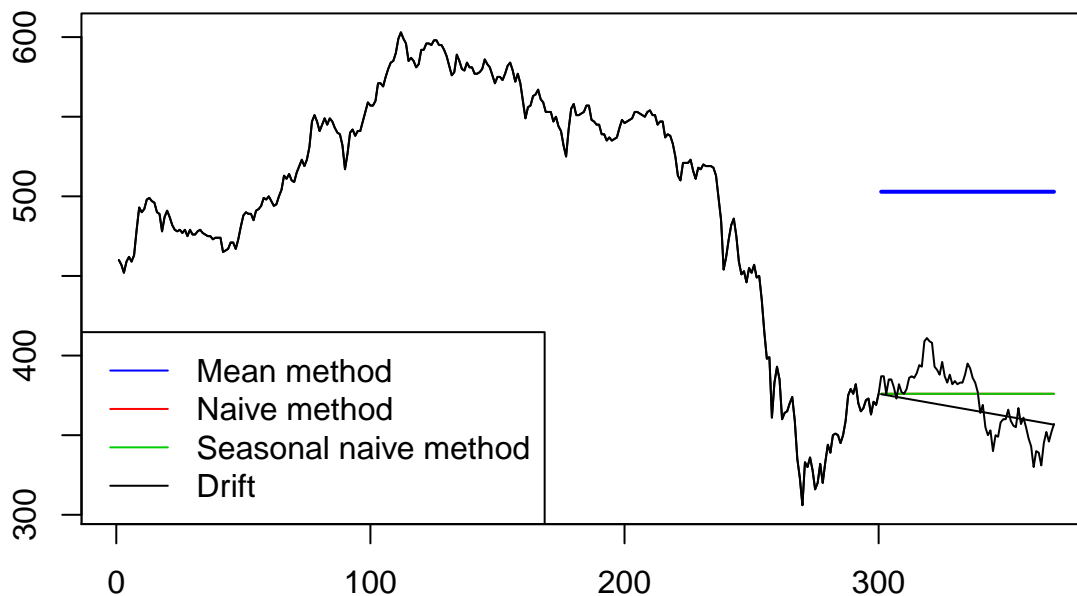
ibmclose.prediction_horizon <- length(ibmclose) - 300;

# Calc mean, naive, seasonal naive, drift predictions
ibmclose.fit.mean <- meanf(ibmclose.training, h=ibmclose.prediction_horizon)
ibmclose.fit.naive <- naive(ibmclose.training, h=ibmclose.prediction_horizon)
ibmclose.fit.seasonalNaive <- snaive(ibmclose.training, h=ibmclose.prediction_horizon)
ibmclose.fit.drift <- rwf(ibmclose.training, drift=TRUE, h=ibmclose.prediction_horizon)

# Plot data with predictions
#plot(ibmclose.training, main="Closing Price of IBM Stock")
plot(ibmclose.fit.mean, plot.conf=FALSE, main="Closing Price of IBM Stock")
lines(ibmclose.fit.naive$mean, col=2)
lines(ibmclose.fit.seasonalNaive$mean, col=3)
lines(ibmclose.fit.drift$mean, col=1)
lines(ibmclose)
legend("bottomleft", lty=1, col=c(4,2,3,1), legend=c("Mean method","Naive method","Seasonal naive method","Drift"))

```

Closing Price of IBM Stock



The various error rates are printed for mean, naive, seasonal naive and drift, respectively. It's clear that the mean error is too high compared to the rest. It seems that drift overall tends to have lower error values than the naive methods, but are generally pretty close. As said above, I think my tendency would be for the drift method.

```

# Look at error values from our predictions
accuracy(ibmclose.fit.mean, ibmclose.test)
accuracy(ibmclose.fit.naive, ibmclose.test)
accuracy(ibmclose.fit.seasonalNaive, ibmclose.test)
accuracy(ibmclose.fit.drift, ibmclose.test)

```

```

##              ME      RMSE      MAE      MPE      MAPE
## Training set  1.660438e-14  73.61532  58.72231  -2.642058  13.03019
## Test set      -1.306180e+02  132.12557  130.61797  -35.478819  35.47882

```



```

##           MASE           ACF1 Theil's U
## Training set 11.52098 0.9895779          NA
## Test set    25.62649 0.9314689 19.05515
##           ME           RMSE           MAE           MPE           MAPE           MASE
## Training set -0.2809365 7.302815 5.09699 -0.08262872 1.115844 1.000000
## Test set     -3.7246377 20.248099 17.02899 -1.29391743 4.668186 3.340989
##           ACF1 Theil's U
## Training set 0.1351052          NA
## Test set     0.9314689 2.973486
##           ME           RMSE           MAE           MPE           MAPE           MASE
## Training set -0.2809365 7.302815 5.09699 -0.08262872 1.115844 1.000000
## Test set     -3.7246377 20.248099 17.02899 -1.29391743 4.668186 3.340989
##           ACF1 Theil's U
## Training set 0.1351052          NA
## Test set     0.9314689 2.973486
##           ME           RMSE           MAE           MPE           MAPE
## Training set 2.870480e-14 7.297409 5.127996 -0.02530123 1.121650
## Test set     6.108138e+00 17.066963 13.974747 1.41920066 3.707888
##           MASE           ACF1 Theil's U
## Training set 1.006083 0.1351052          NA
## Test set     2.741765 0.9045875 2.361092

```

Question 2.4

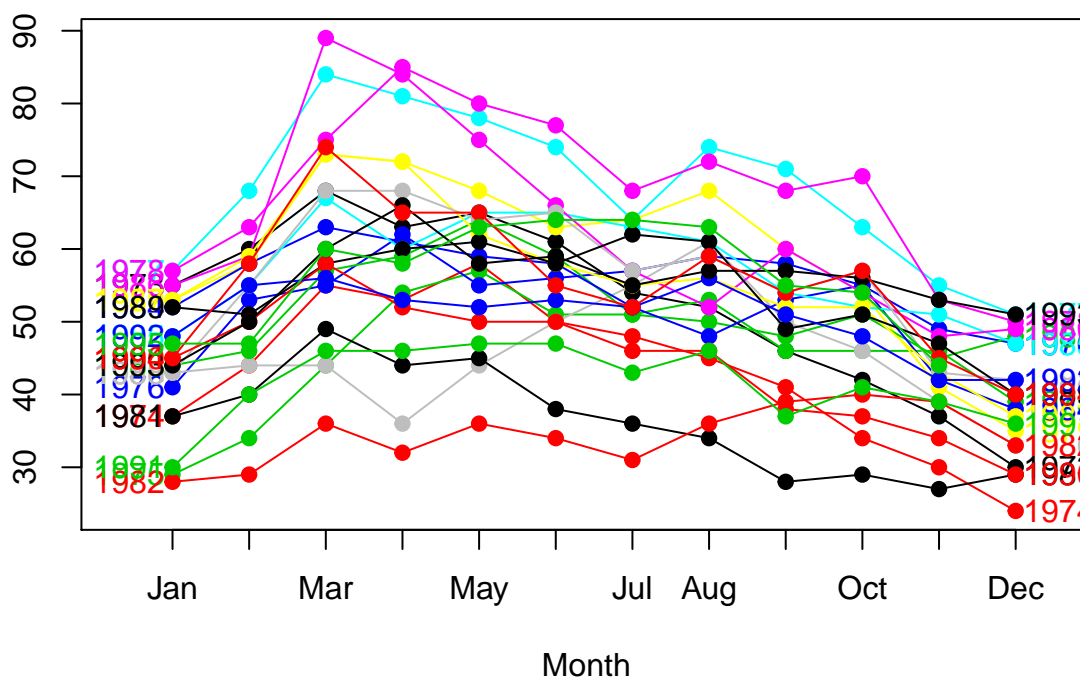
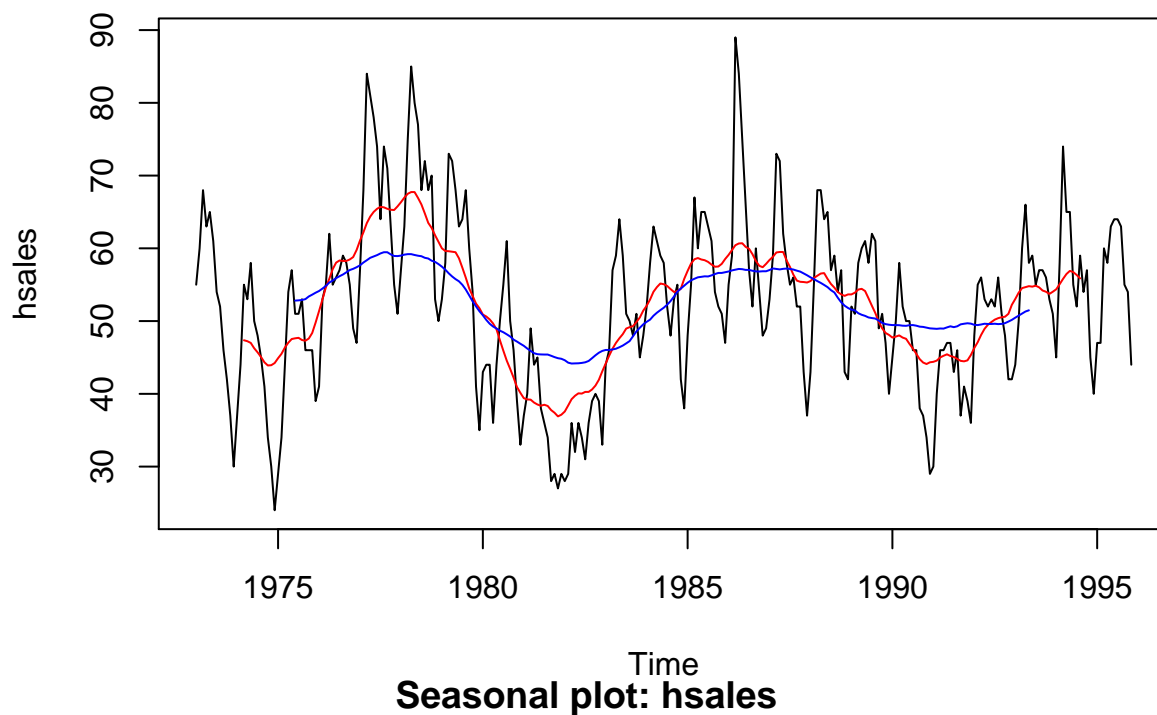
Question 2.4a)

Here a simple ts plot, with a few moving averages is plotted, as well as a month and season plot, since the data is the right format (versus previous question).

```
# Plot time series with some moving averages
plot(hsales, main="Sales of new one-family houses in the USA")
lines(rollmean(hsales, k=30, fill=NA), col="red")
lines(rollmean(hsales, k=60, fill=NA), col="blue")

# Show seasonal plot, which should a general downward trend from March onwards
seasonplot(hsales, col=1:20, pch=19, year.labels=TRUE, year.labels.left=TRUE)
```

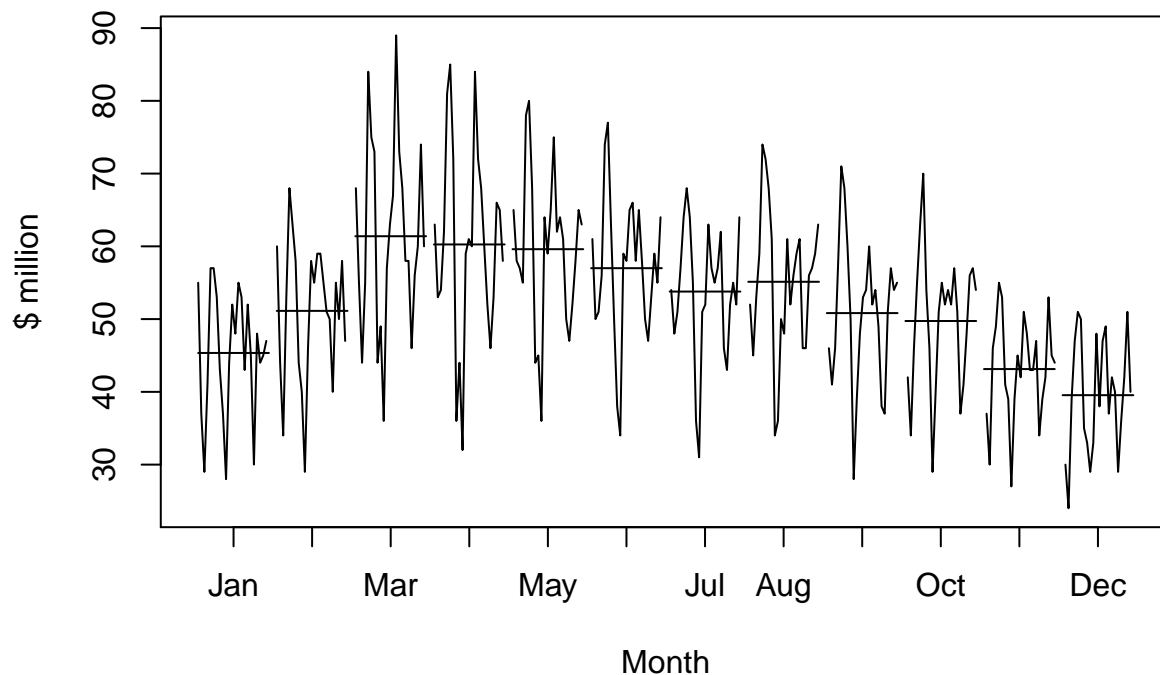
Sales of new one-family houses in the USA



Here are the sales by month, which emphasizes that there is a ramp up in sales from Jan to Mar, but declines for the rest of the year:

```
monthplot(hsales,ylab="$ million",xlab="Month",xaxt="n", main="Sales of new one-family houses in the US",
axis(1,at=1:12,labels=month.abb,cex=0.8)
```

Sales of new one-family houses in the USA



Question 2.4b)

Split the hsales data set into a training set and a test set, where the test set is the last two years of data.

```
hsales.training <- window(hsales, start=c(1973, 1), end=c(1993, 12))
hsales.test <- window(hsales, start=c(1994, 1), end=c(1995, 11))
```

Question 2.4c)

Visual inspection shows that the seasonal naive method looks pretty accurate across the 23 months of prediction, though the predicted value in 23 months looks pretty off. None of the methods seem to predict the end drop-off in value towards the end of 1995.

```
hsales.prediction_horizon <- 23;

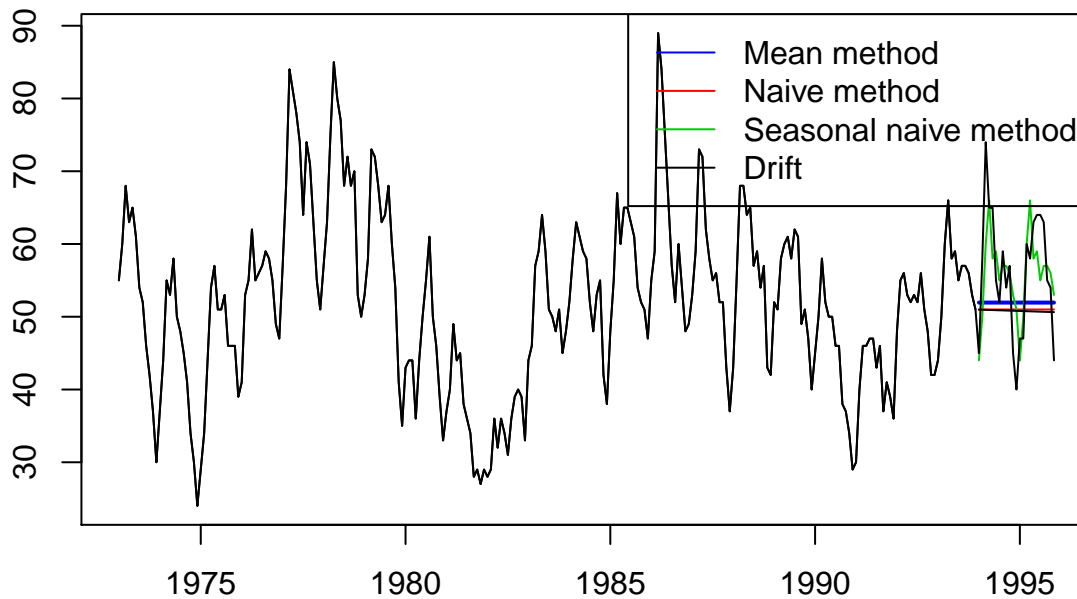
# Calculate the mean, naive, seasonal naive, drift predictions
hsales.fit.mean <- meanf(hsales.training, h=hsales.prediction_horizon)
hsales.fit.naive <- naive(hsales.training, h=hsales.prediction_horizon)
hsales.fit.seasonalNaive <- snaive(hsales.training, h=hsales.prediction_horizon)
hsales.fit.drift <- rwf(hsales.training, drift=TRUE, h=hsales.prediction_horizon)

# Plot the data with our predictions
#plot(hsales.training, main="Sales of new one-family houses in the USA w/ Forecasts")
plot(hsales.fit.mean, plot.conf=FALSE, main="Sales of new one-family houses in the USA")
lines(hsales.fit.naive$mean, col=2)
lines(hsales.fit.seasonalNaive$mean, col=3)
lines(hsales.fit.drift$mean, col=1)
```

```
lines(hsales)
legend("topright", lty=1, col=c(4,2,3,1), legend=c("Mean method","Naive method","Seasonal naive method",

```

Sales of new one-family houses in the USA



Here are the error values for the mean, naive, seasonal naive and drift methods, respectively. The seasonal naive is a clear winner here. Visually, the seasonal naive method overlaps the test data very well, and generally has the least amount of error for any error metric.

```
# Look at error values from our predictions
accuracy(hsales.fit.mean, hsales.test)
accuracy(hsales.fit.naive, hsales.test)
accuracy(hsales.fit.seasonalNaive, hsales.test)
accuracy(hsales.fit.drift, hsales.test)
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 2.480763e-15 12.138802 9.498898 -6.120182 20.30851 1.119163
## Test set    4.051587e+00 9.216133 7.850759 5.074990 13.75973 0.924979
##               ACF1 Theil's U
## Training set 0.8661515      NA
## Test set    0.5095178 1.13105
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -0.01593625 6.289813 4.988048 -0.7800232 9.880157 0.5876934
## Test set     5.00000000 9.670664 8.304348 6.8080182 14.381673 0.9784210
##               ACF1 Theil's U
## Training set 0.1829708      NA
## Test set    0.5095178 1.179633
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 0.1375000 10.576113 8.4875 -2.1016380 17.63375 1.0000000
## Test set     0.3043478 6.160886 5.0000 -0.7312374 9.12828 0.5891016
##               ACF1 Theil's U
## Training set 0.838108      NA
## Test set     0.224307 0.8031005
```

	ME	RMSE	MAE	MPE	MAPE	MASE
## Training set	2.377739e-15	6.289793	4.987730	-0.7474544	9.87819	0.5876560
## Test set	5.191235e+00	9.761548	8.393037	7.1599507	14.50303	0.9888703

	ACF1	Theil's U
## Training set	0.1829708	NA
## Test set	0.5083059	1.188562

Question 3.2a

With regards to being degenerate, I will refer to Section 3.5 about Removing Predictors, since I assume that degeneracy means ‘detecting that this predictor should probably be removed’. The rule of thumb is predictors are ‘degenerate’ if the frequency of the first and second most common values for that predictor is > 20 , while the fraction of unique values as compared to the number of samples is low.

First, the fraction of unique values over the sample size is low for all columns, so we just need to check the ratio of first and second most frequent value per predictor. However, there is some ambiguity here as to whether or not we should be including NAs as a value. I have done it both ways, since the results are a little different.

With NAs counted, the columns that are close to the 20 threshold are mycelium, int.discolor, and sclerotia. However, with NAs not counted, the columns are mycelium, sclerotia and leaf.mild. My intuition says that we should not count the NAs (since the histogram here is very clear that some columns are over the threshold) Therefore, mycelium, sclerotia and leaf.mild should be considered degenerate.

```
nrows <- nrow(Soybean);

# Lets check our rule of thumb by calculating ratio of 1st and 2nd highest freq
i <- 1;
ratiosWithoutNAs <- c()
ratiosWithNAs <- c()
for (column in colnames(Soybean)[2:length(colnames(Soybean))]) {
  # Do the calculation using count(), which includes NAs
  t <- count(Soybean, column)
  sorted_t <- t[order(-t['freq']), 'freq']
  ratiosWithNAs[i] <- (sorted_t[1] / sorted_t[2]);

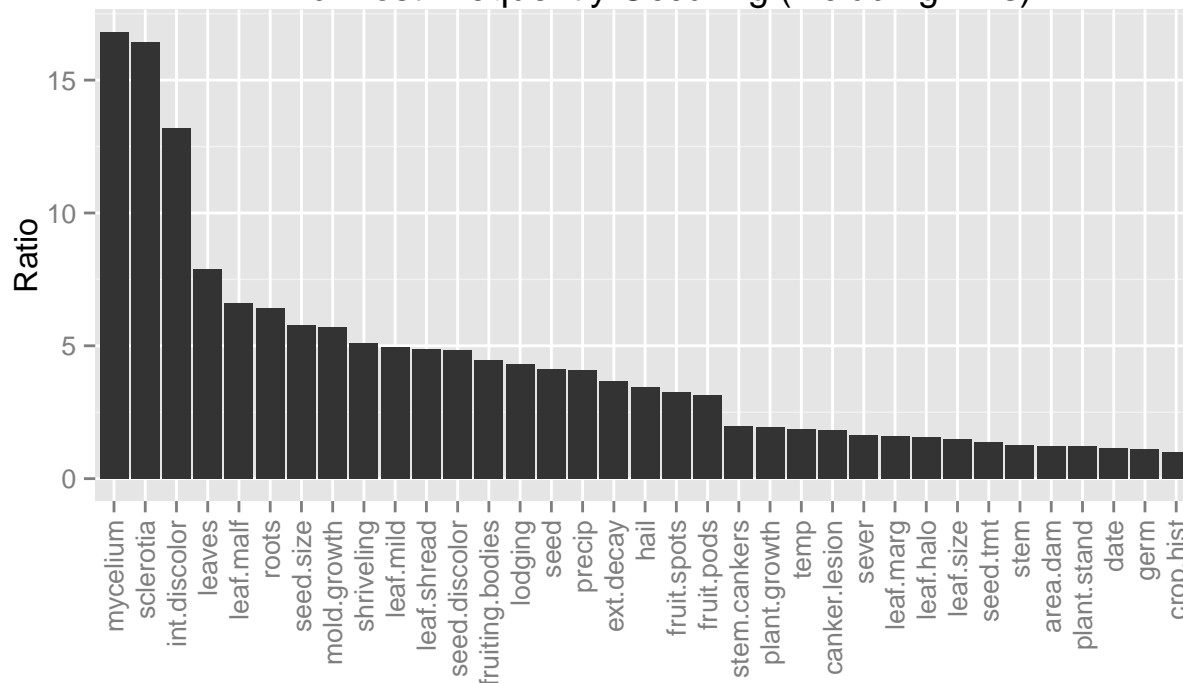
  # Do the calculation using table() which does not include NAs
  t <- table(Soybean[, column])
  sorted_t <- sort(as.numeric(t), decreasing=TRUE)
  ratiosWithoutNAs[i] <- (sorted_t[1] / sorted_t[2]);

  # Increment the idx for ratio lists
  i <- i + 1;
}

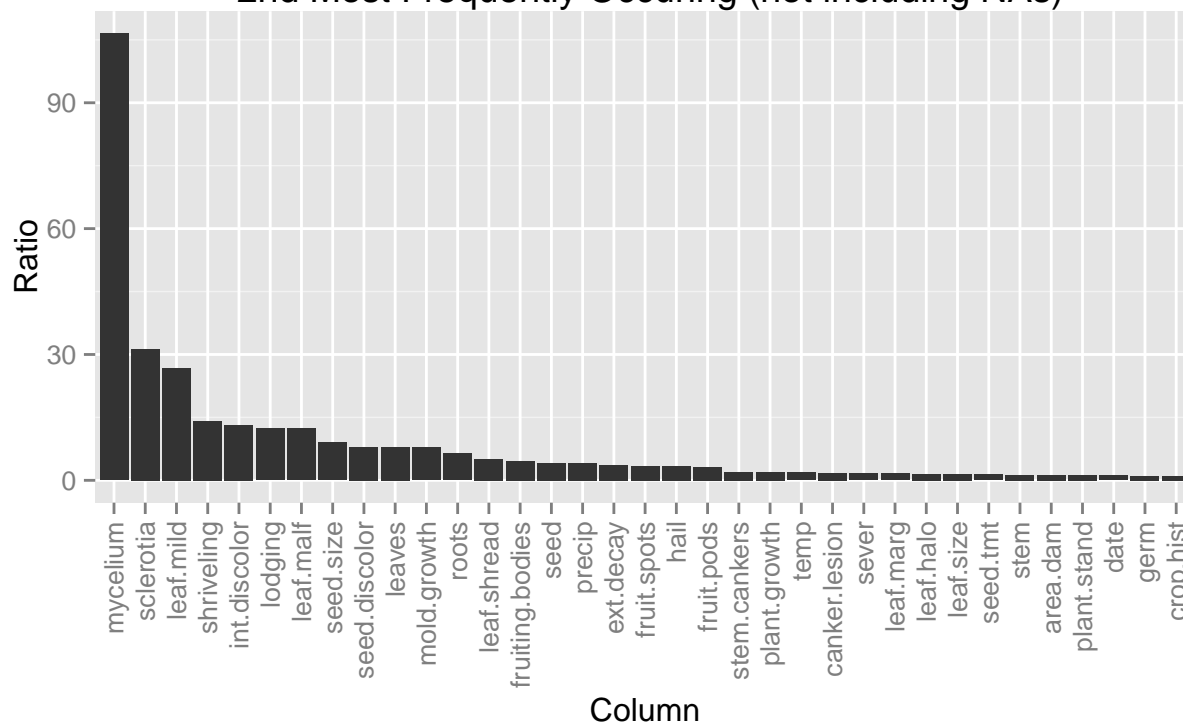
# Display the histogram when accounting for NAs
ratiosWithNAs.dist <- data.frame(y=ratiosWithNAs, x=colnames(Soybean)[2:length(colnames(Soybean))])
ratiosWithNAs.dist$x <- reorder(ratiosWithNAs.dist$x, -ratiosWithNAs.dist$y)
ggplot(ratiosWithNAs.dist) +
  geom_bar(aes(x=x, y=y), stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  ggtitle("Distribution of the Ratios of Most Frequently Occuring to \n 2nd Most Frequently Occuring")
  ylab("Ratio") + xlab("Column")

# Display the histogram when not accounting for NAs
ratiosWithoutNAs.dist <- data.frame(y=ratiosWithoutNAs, x=colnames(Soybean)[2:length(colnames(Soybean))])
ratiosWithoutNAs.dist$x <- reorder(ratiosWithoutNAs.dist$x, -ratiosWithoutNAs.dist$y)
ggplot(ratiosWithoutNAs.dist) +
  geom_bar(aes(x=x, y=y), stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  ggtitle("Distribution of the Ratios of Most Frequently Occuring to \n 2nd Most Frequently Occuring")
  ylab("Ratio") + xlab("Column")
```

Distribution of the Ratios of Most Frequently Occuring to
2nd Most Frequently Occuring (including NAs)



Distribution of the Ratios of Most Frequently Occuring to
2nd Most Frequently Occuring (not including NAs)

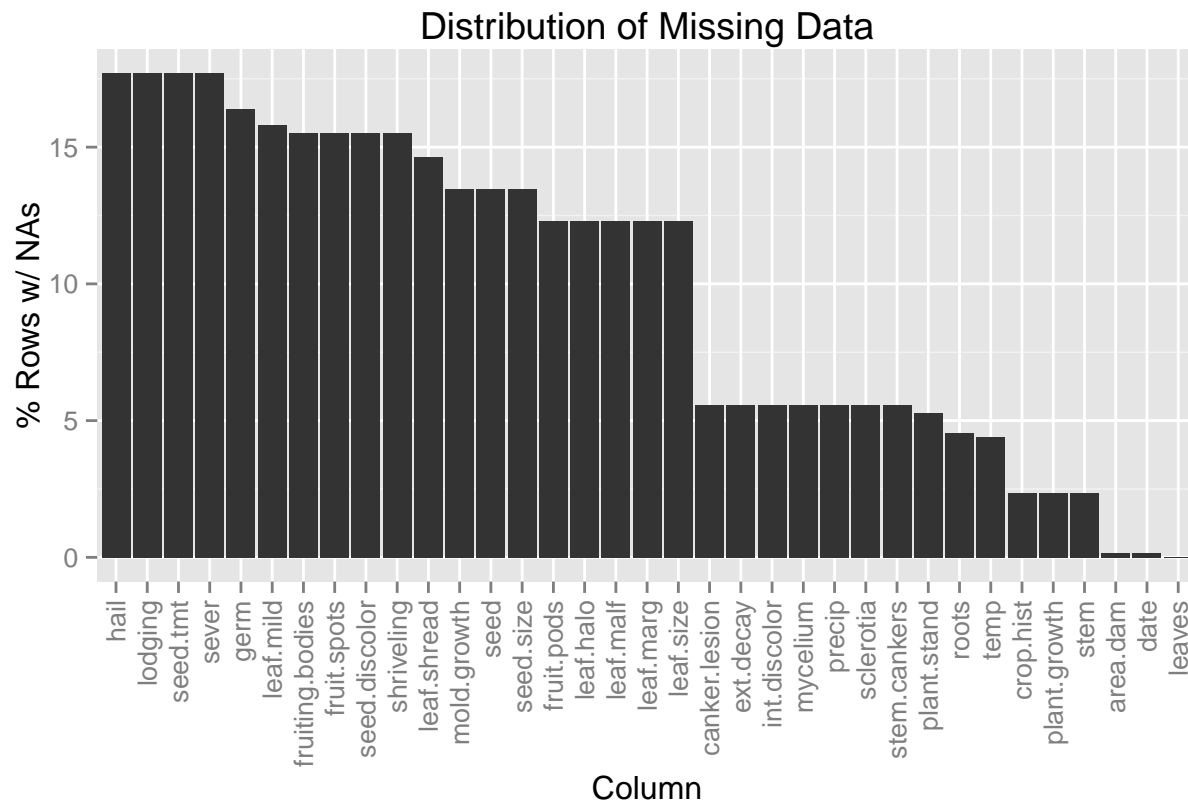


Question 3.2b

Looking at the histogram, it seems that quite a few predictors have significant number of NAs. There is a dropoff in NA level from around 12% to 5% (highest being almost 20% missing data). Also, the columns herbicide-injury, cyst-nematode, 2-4-d-injury, diaporthe-pod-&-stem-blight have no instances / rows that do not have a NA in them. The class phytophthora-rot has 68 NAs, but all other classes have complete cases:

```
i <- 1;
nas <- c()
for (column in colnames(Soybean)[2:length(colnames(Soybean))]) {
  t <- count(Soybean, column)
  nas[i] <- ifelse(length(t[is.na(t[column])]), 'freq'), t[is.na(t[column])], 'freq' * 100 / nrow, 0)
  i <- i + 1;
}

# Display histogram of distrubtion of NAs
nas.dist <- data.frame(y=nas, x=colnames(Soybean)[2:length(colnames(Soybean))])
nas.dist$x <-reorder(nas.dist$x,-nas.dist$y)
ggplot(nas.dist) +
  geom_bar(aes(x=x, y=y), stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  ggtitle("Distribution of Missing Data") +
  ylab("% Rows w/ NAs") + xlab("Column")
```



```
# The columns herbicide-injury, cyst-nematode, 2-4-d-injury, diaporthe-pod-&-stem-blight
# have no instances / rows that do not have a NA in them
# The Class phytophthora-rot has 68 NAs, but all other classes have none
withnas <- count(Soybean$Class)
```

```

withoutnas <- count(Soybean[complete.cases(Soybean), 'Class'])
for (i in 1:nrow(withoutnas)) {
  classval <- withoutnas[i, 'x']
  diff <- withnas[withnas$x == classval, 'freq'] - withoutnas[withoutnas$x == classval, 'freq']
  if(diff > 0) {
    print(paste("Class", classval, "has", diff, " NAs"))
  }
}

```

```
## [1] "Class phytophthora-rot has 68 NAs"
```

Question 3.2c)

We should start by investigating the nature of the missing data for predictors with a large number of missing values as well as why some classes have a high occurrence of NAs. Our predictive model is going to be biased if we do not have good observations for 4 or 5 of the 19 outcome classes.

For each predictor that has a high number of NAs (starting from the left in the question_3.2b histogram), we can determine if we can remove the predictor by checking if it has a high collinearity with any other predictor. If so, we can remove that predictor and simply use the collinear predictor (presuming it has less instances of NAs). Imputing values would be difficult, since these are all categorical variables, and averaging those values would make no sense. We could impute a value using K-nearest neighbors, which would find the K rows that are closest to it by some definition of 'close', and then take a majority vote on what the categorical value should be.

Question 4.4a)

The code below generates random samples of size 60 from oilType. The output shows how inconsistent the samples are with respect to the class frequencies, and vary far from the distribution from the full data.

```
## (a) Use the sample function in base R to create a completely random sample  
## of 60 oils. How closely do the frequencies of the random sample match  
## the original samples? Repeat this procedure several times of understand  
## the variation in the sampling process.  
print("Frequency from Data")  
print(freq(oilType, plot = FALSE))  
  
cat("\n")  
print("Frequency from Sampling 60 items")  
for (i in 1:5) {  
  print(freq(sample(oilType, 60), plot = FALSE))  
}
```

```
## [1] "Frequency from Data"  
## oilType  
##      Frequency Percent  
## A           37  38.542  
## B           26  27.083  
## C            3   3.125  
## D            7   7.292  
## E           11  11.458  
## F           10  10.417  
## G            2   2.083  
## Total        96 100.000  
##  
## [1] "Frequency from Sampling 60 items"  
## sample(oilType, 60)  
##      Frequency Percent  
## A           21  35.000  
## B           19  31.667  
## C            1   1.667  
## D            6  10.000  
## E            5   8.333  
## F            7  11.667  
## G            1   1.667  
## Total        60 100.000  
## sample(oilType, 60)  
##      Frequency Percent  
## A           21  35.00  
## B           16  26.67  
## C            3   5.00  
## D            6  10.00  
## E            7  11.67  
## F            7  11.67  
## G            0   0.00  
## Total        60 100.00  
## sample(oilType, 60)  
##      Frequency Percent  
## A           21  35.000
```

```
## B          17  28.333
## C           2   3.333
## D           5   8.333
## E           9  15.000
## F           5   8.333
## G           1   1.667
## Total      60 100.000
## sample(oilType, 60)
##      Frequency Percent
## A          25  41.667
## B          14  23.333
## C           3   5.000
## D           4   6.667
## E           7  11.667
## F           6  10.000
## G           1   1.667
## Total      60 100.000
## sample(oilType, 60)
##      Frequency Percent
## A          25  41.667
## B          18  30.000
## C           0   0.000
## D           4   6.667
## E           6  10.000
## F           6  10.000
## G           1   1.667
## Total      60 100.000
```

Question 4.4b)

The createDataPartition is indeed giving back different numbers every time but table() output shows how it is very consistent in how it distributes samples across class values, and is very close to the ideal from the data itself. Much better than the previous approach.

```
##' (b) Use the caret package function createDataPartition to create a stratified
##' random sample. How does this compare to the completely random samples?
##' the variation in the sampling process.
print("Frequency from Data")
print(freq(oilType, plot = FALSE))

cat("\n")
print("Frequency from Sampling 60 items")
for (i in 1:5) {
  print(freq(oilType[createDataPartition(oilType, p=60 / length(oilType), list=FALSE)], plot = FALSE))
}
```

```
## [1] "Frequency from Data"
## oilType
##      Frequency Percent
## A          37  38.542
## B          26  27.083
## C           3   3.125
## D           7   7.292
```

```

## E          11  11.458
## F          10  10.417
## G           2   2.083
## Total      96 100.000
##
## [1] "Frequency from Sampling 60 items"
## oilType[createDataPartition(oilType, p = 60/length(oilType),      list = FALSE)]
##      Frequency Percent
## A          24  37.500
## B          17  26.562
## C           2   3.125
## D           5   7.812
## E           7  10.938
## F           7  10.938
## G           2   3.125
## Total      64 100.000
## oilType[createDataPartition(oilType, p = 60/length(oilType),      list = FALSE)]
##      Frequency Percent
## A          24  37.500
## B          17  26.562
## C           2   3.125
## D           5   7.812
## E           7  10.938
## F           7  10.938
## G           2   3.125
## Total      64 100.000
## oilType[createDataPartition(oilType, p = 60/length(oilType),      list = FALSE)]
##      Frequency Percent
## A          24  37.500
## B          17  26.562
## C           2   3.125
## D           5   7.812
## E           7  10.938
## F           7  10.938
## G           2   3.125
## Total      64 100.000
## oilType[createDataPartition(oilType, p = 60/length(oilType),      list = FALSE)]
##      Frequency Percent
## A          24  37.500
## B          17  26.562
## C           2   3.125
## D           5   7.812
## E           7  10.938
## F           7  10.938
## G           2   3.125
## Total      64 100.000
## oilType[createDataPartition(oilType, p = 60/length(oilType),      list = FALSE)]
##      Frequency Percent
## A          24  37.500
## B          17  26.562
## C           2   3.125
## D           5   7.812
## E           7  10.938
## F           7  10.938

```

```
## G          2    3.125
## Total      64 100.000
```

Question 4.4c)

When the number of samples is not large, a single test set should be avoided because we may need every sample during model building. We should use somekind of resampling technique, like k-fold cross-validation. The book recommends “If the samples size is small, we recommend repeated 10-fold cross-validation for several reasons: the bias and variance properties are good and, given the sample size, the computational costs are not large.

Question 4.4d)

The code following this shows the commands I used to come to the following conclusions.

By keeping the model performance steady, while changing the test sample size leads to a relationship where higher sample size means less uncertainty. This makes sense: we would expect us to be more certain if we have more samples to evaluate the model performance with.

If you keep the sample size steady, and vary the model performance leads to a relationship where: - the uncertainty maxes when performance is 50% (which makes sense since its binomial probability). - the uncertainty is less when model performance is near either extreme (partially due to clipping of the CI by the boundry)

```
spread <- function(obj) { conf <- obj$conf.int; return(conf[2] - conf[1]);}
spread(binom.test(16, 20)) # 0.37928
spread(binom.test(32, 40)) # 0.2659556
spread(binom.test(30, 40)) # 0.2850472
spread(binom.test(10, 40)) # 0.2850472
spread(binom.test(15, 40)) # 0.3147225
spread(binom.test(20, 40)) # 0.3239644
spread(binom.test(25, 40)) # 0.3147225
spread(binom.test(40, 40)) # 0.0880973
spread(binom.test(20, 40)) # 0.3239644
spread(binom.test(21, 40)) # 0.3236002
spread(binom.test(19, 40)) # 0.3236002
spread(binom.test(19, 20)) # 0.2474677
spread(binom.test(20, 20)) # 0.1684335
spread(binom.test(10, 20)) # 0.4560843
spread(binom.test(1, 20))  # 0.2474677
```