Assignment6

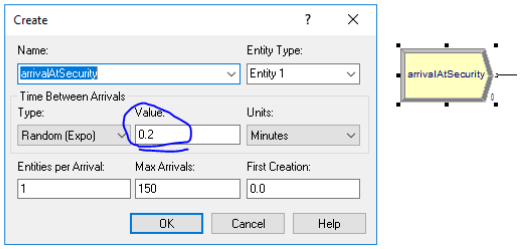
Prashant Kubsad

6/22/2020

Question 13.2

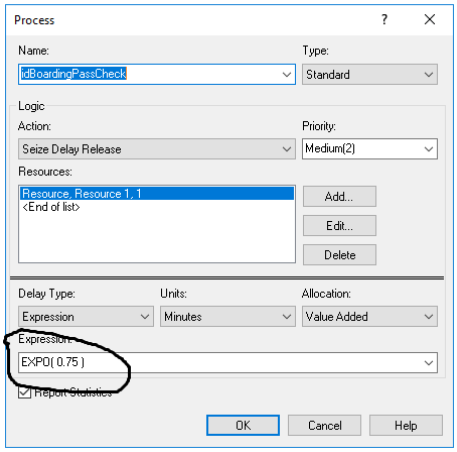
I have used Arena software to simulate the use case given in the homework. As a first step I created a simple flow with one boarding pass checkpoint and one security point. Lets take each point mentoined in the hw:

Passengers arrive according to a Poisson distribution with λ_1 = 5 per minute. As mentioned in the last week lecture, If the passenger arrival is poisson distribution, then the inter arrival time will be exponential distribution of $1/\lambda$ = 0.2. So i have added create component in arena with time between arrivals as exponential distribution of 0.2.



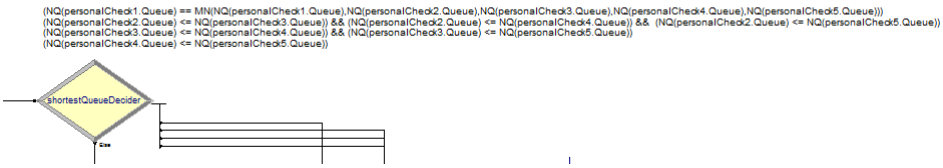
Security Arrival

ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate $\mu 2$ = 0.75 minutes. I have added a process component with exponential service time of 0.75.



Boarding Pass Station

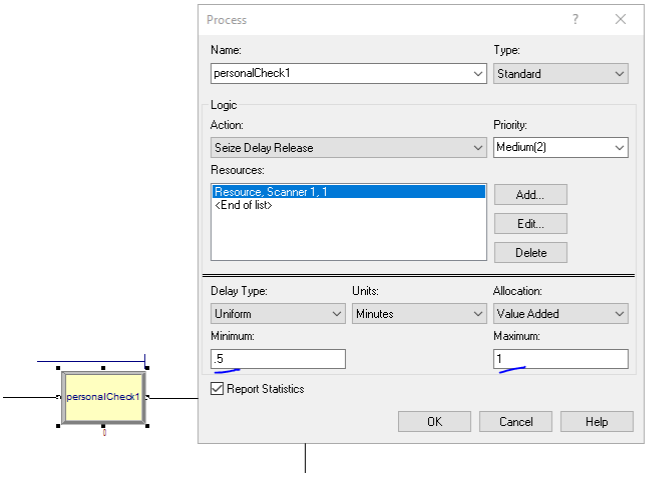
After that, the passengers are assigned to the shortest of the several personal-check queues: I have added a decision component to the block with expression that will evaluate which queue has lesser number of people and adds the next person to that queue. The logic expression is as below:



Queue Decider

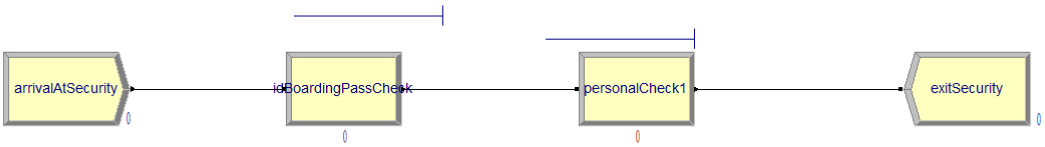
---

personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute). I have added process component representing each queue with the uniform distribution as given above.



Personal/Security Check Queue

I created a simple simulation with 1 boarding pass station with a capacity of 1 and 1 security/personal check. The flow for that looks as in the picture below.



Arena Simulation Basic Flow

When we run simulation for this configuration, we see the avg wait time is around 49 minutes.

7:53:57PM **Category Overview** June 24, 2020

Unnamed Project

Replications: 1 Time Units: Minutes

Entity

Time

VA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	1.5556	(Insufficient)	0.5808	4.7083

NVA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Wait Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	47.4694	(Insufficient)	0.00	94.5353

Transfer Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Other Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Total Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	49.0251	(Insufficient)	1.1074	95.2702

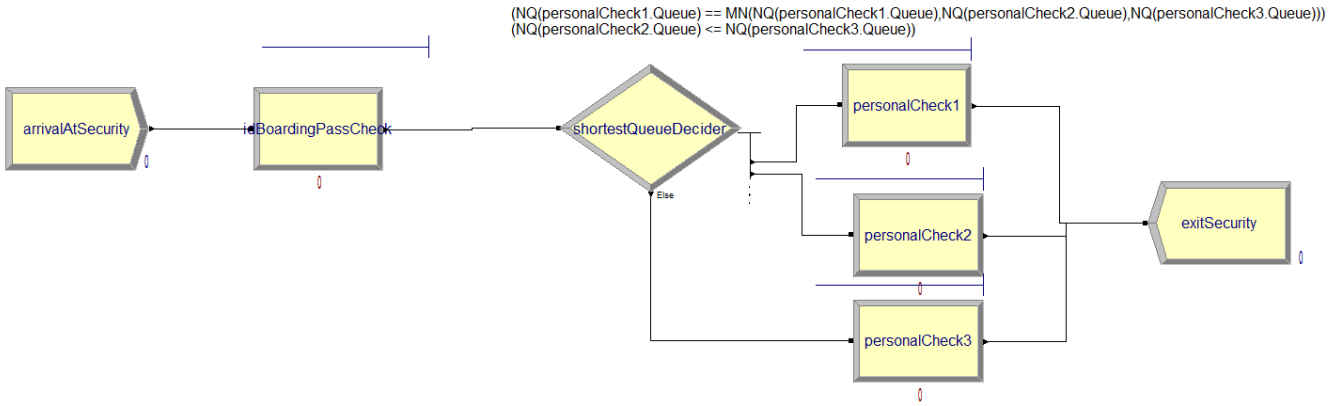
Other

Number In	Value
Entity 1	150.00

Number Out	Value
Entity 1	150.00

Report of times for simple flow

Next, I added 2 more security scanners to the mix and we see that the avg wait times came down to 47 mins and the over work in progress time came to 63.78 minutes.



Arena Simulation With Multiple Scanners

Report:

Unnamed Project

Replications: 1 Time Units: Minutes

Entity

Time

VA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	1.4935	(Insufficient)	0.5754	4.4106

NVA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Wait Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	47.0029	(Insufficient)	0.00	92.2041

Transfer Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Other Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	(Insufficient)	0.00	0.00

Total Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	48.4964	(Insufficient)	1.1074	93.4256

Other

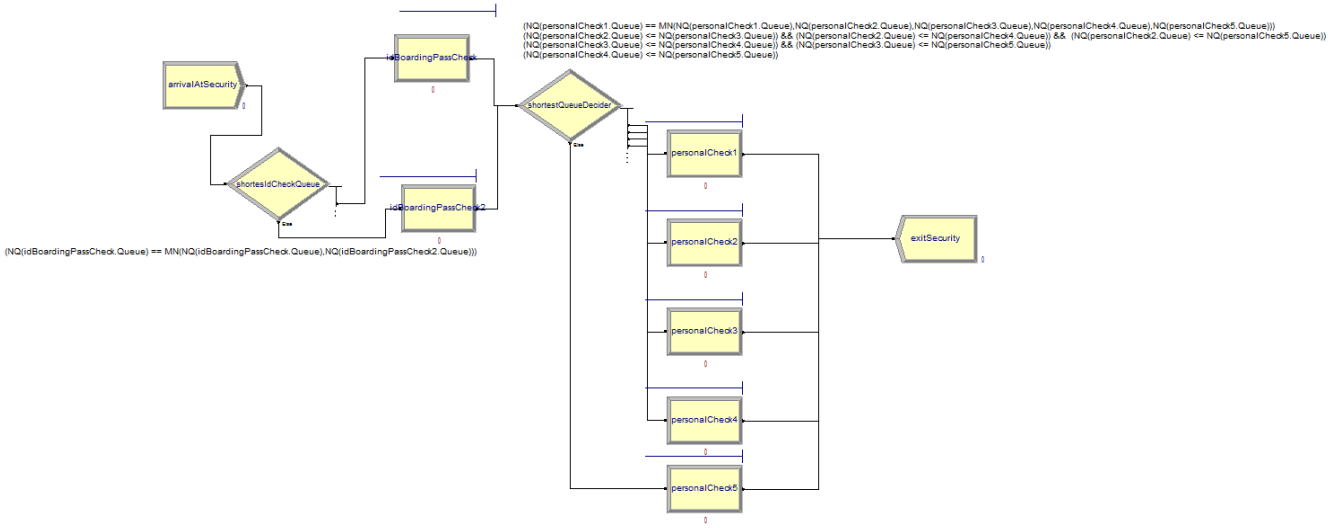
Number In	Value
Entity 1	175.00

Number Out	Value
Entity 1	175.00

WIP	Average	Half Width	Minimum Value	Maximum Value
Entity 1	63.7848	(Correlated)	0.00	125.00

Report of times for flow with 3 queues

To reduce the avg WIP time, I kept trying combinations of number of security queues and boarding pass stations. I found the below combination to reduce the WIP time to under 15 mins. This model has 2 boarding pass stations with a capacity of processing 3 people at a time. Totally 5 security lines which is processing one person at a time.



Arena Simulation With 2 Boarding Pass stations and 5 Scanners

Report times:

12:02:58AM **Category Overview** June 24, 2020

Unnamed Project

Replications: 1 Time Units: Minutes

Entity

Time

VA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	1.5118	0.018386063	0.5099	7.3166

NVA Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	0.000000000	0.00	0.00

Wait Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.8997	0.117340270	0.00	8.4860

Transfer Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	0.000000000	0.00	0.00

Other Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	0.00	0.000000000	0.00	0.00

Total Time	Average	Half Width	Minimum Value	Maximum Value
Entity 1	2.4115	0.131433229	0.5444	9.8813

Other

Number In	Value
Entity 1	5933.00

Number Out	Value
Entity 1	5922.00

WIP	Average	Half Width	Minimum Value	Maximum Value
Entity 1	11.9106	0.719208316	0.00	31.0000

Report of times for final model

Resoureces used to achieve above times:

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Scanner 1	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑
2	Resource 1	Fixed Capacity	3	0.0	0.0	0.0		0 rows	☑
3	Scanner 2	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑
4	Scanner 3	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑
5	Scanner 4	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑
6	ibpc2	Fixed Capacity	2	0.0	0.0	0.0		0 rows	☑
7	personalCheck 5	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑

I have attached the full final report to the assignment.

I have uploaded the animation of the actual simulation to youtube:

https://www.youtube.com/watch?v=tRTVwx7XdB0&feature=youtu.be (https://www.youtube.com/watch?v=tRTVwx7XdB0&feature=youtu.be)

Question 14.1

14.1.1. Use the mean/mode imputation method to impute values for the missing data.

I have used the mice package to illustrate the missing data and how it is distributed across the data. MICE package in R gives very good representation of missing data. In the diagram we can see there are 683 rows without any missing data (blue boxes), there are 16 rows with missing values and all the 16 are in cloumn V7(red box). This tells us that all the missing data is in column V7. Also, vis_miss function gives good representation of missing data whcih will help us to identify clusters of data missing. In our example we can see where those 16 rows of missing data is. If there were multiple columns with missing data, this graph would give good cluster representation of missing data.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

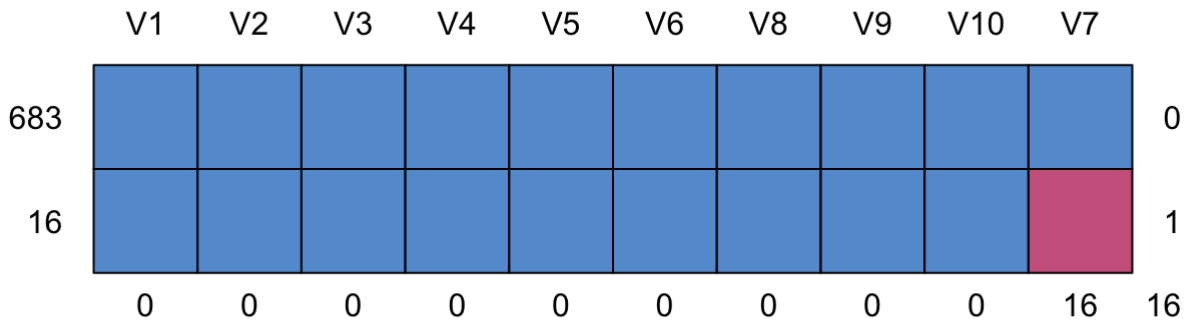
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(visdat)
#setting seed for consistent results
set.seed(777)

#Read the text file into data, any cell with ? is treated as missing(na) data.
data<-read.delim("breast-cancer-wisconsin.data.txt", sep=',', na=c('?'),header = FALSE)
head(data)
```

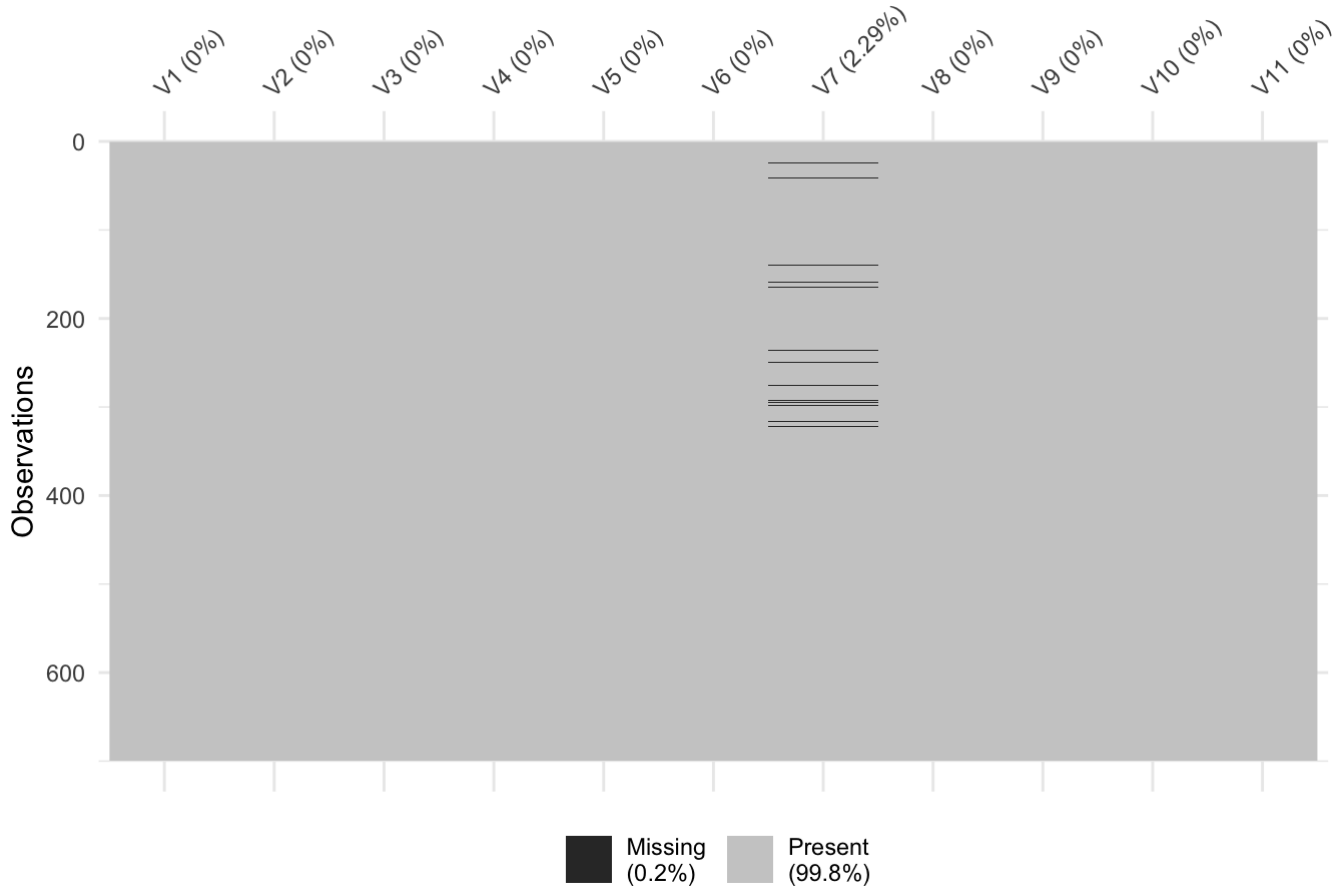
```
##        V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
## 4 1016277  6  8  8  1  3  4  3  7   1   2
## 5 1017023  4  1  1  3  2  1  3  1   1   2
## 6 1017122  8 10 10  8  7 10  9  7   1   4
```

```
#printing details of missing data with chart
md.pattern(data[,-11])
```




```
##     V1 V2 V3 V4 V5 V6 V8 V9 V10 V7
## 683  1  1  1  1  1  1  1  1   1  1  0
## 16   1  1  1  1  1  1  1  1   1  0  1
##      0  0  0  0  0  0  0  0   0 16 16
```

```
#representation of missing data per each column and rows with missing data
vis_miss(data)
```




```
#finding out missing rows
missingRows<-which(is.na(data$V7))

dataAfterRemovingMissingRows<-data[-missingRows,]
dataOfMissingRows<-data[missingRows,]

#split of full data to see how many are 2 and 4
table(data$V11)
```

```
##
##   2   4
## 458 241
```

```
#split of data after removing the missing data rows
table(dataAfterRemovingMissingRows$V11)
```

```
##
##   2   4
## 444 239
```

```
#split of data with only missing data
table(dataOfMissingRows$V11)
```

```
##
##  2  4
## 14  2
```

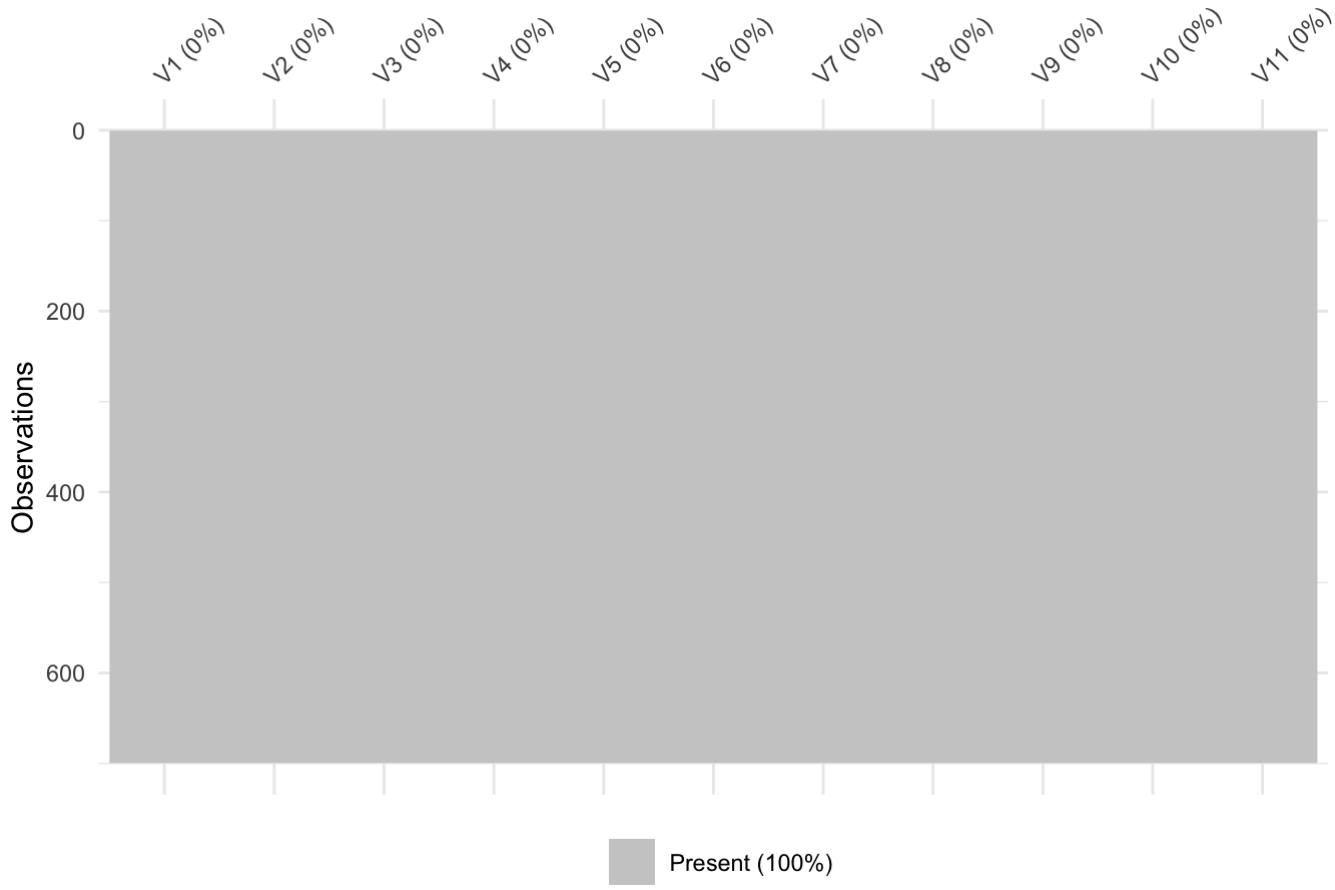
From the above graph we can see that only 0.2% of the data is missing and imputing these records will not significantly impact the model. As mentioned in the lecture, it is well within the 5% rule of thumb to check if we can fill the missing data without impacting the model significantly. As mentioned in the office hours, the split of response column (V11) for full data is 458 2's and 241 4's. If we remove the rows with missing data, we see the similar split of 444 - 239. But if we just take a look at rows with missing data, it is 14 -2 which means the missing data is heavy on response value of 2's.

Since column V7 (bare nuclie) is a categorical column from 1 to 10, I am using the mode function to fill the missing data. Mode function gives us the value that occurs most amongst the rest of the non-missing data.

```
# R does not have a mode function, so created a mode function
# source: https://www.tutorialspoint.com/r/r_mean_median_mode.htm
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

#calling above function to find the mode value
modeValue <- as.numeric(getmode(data[-missingRows,"V7"]))
#mode value
modeValue
```

```
## [1] 1
```

```
#creating a copy of data set to add the modeValue
modeImputedData<-data
modeImputedData[missingRows,"V7"]=modeValue
#data representation after adding all the data
# you can see the graph is no longer reporting any missing data
vis_miss(modeImputedData)
```



Just for the sake of experimenting, I have run mean method as well to fill the missing data with mean of the values.

```
#calculating mean value of column 7
meanValue<-round(mean(as.numeric(data[-missingRows,"V7"])))
meanValue
```

```
## [1] 4
```

```
#creating a copy of data set to add the modeValue
meanImputedData<-data
meanImputedData[missingRows,"V7"]=meanValue
```

14.1.2. Use regression to impute values for the missing data.

I have cleaned up the data by removing 1st column and 11th column as they are not significant to determine the values of V7. I have used cpairs chart and cor function to plot the co-relation between the factors. We can visualise the co realtion among all the factors using cpairs chart. Darker color pink indicates strong co-relation between the factors while lighter colors yellow and green show weak or no co relation between the factors. Ex. Cells intersecting V7 and V4 show co relation between V7 and V4. Similarly this chart shows there is a co relation between V7 and V3, V7 and V8. So, to decide the significant factors, lets run stepwise regression and backward elimination methods which we learnt in the last assignment.

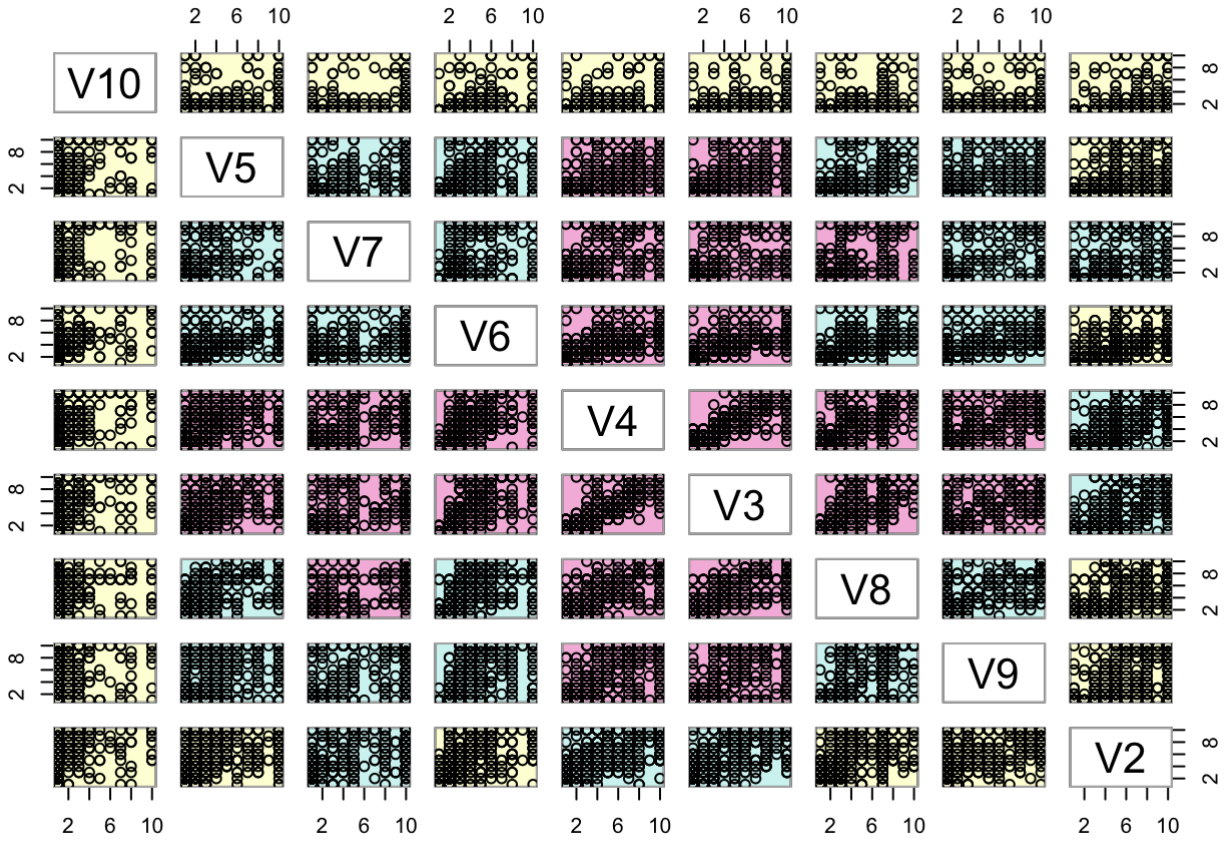
```
#setting seed for consistent results
set.seed(777)
#loading required data
library(gclus)
```

```
## Loading required package: cluster
```

```
library(DAAG)
```

```
## Loading required package: lattice
```

```
#removing 1st column and last column as they are not significant to determining missing
 V7 values
cleanedDataForNonMissingRows<-dataAfterRemovingMissingRows[,2:10]
cleanedDataForMissingRows<-dataOfMissingRows[,2:10]

#Identifying if there is any co relation between V7 and any other factors
my.abs<-abs(cor(cleanedDataForNonMissingRows))
my.colors<-dmat.color(my.abs)
my.ord<-order.single(cor(cleanedDataForNonMissingRows))
cpairs(cleanedDataForNonMissingRows,my.ord,panel.colors = my.colors)
```



```
#Performing step wise regression to identify the significant factors.
stepwise<-step(lm(V7~1,data=cleanedDataForNonMissingRows),scope=V7~V2+V3+V4+V5+V6+V8+V9+
V10,direction = "both",trace = FALSE)
stepwise
```

```
##
## Call:
## lm(formula = V7 ~ V4 + V5 + V8 + V2, data = cleanedDataForNonMissingRows)
##
## Coefficients:
## (Intercept)           V4           V5           V8           V2
##     -0.5360       0.3173       0.3323       0.3238       0.2262
```

```
#Performing backward elimination method to identify the significant factors.
backward<-step(lm(V7~.,data=cleanedDataForNonMissingRows),direction = "backward",trace =
FALSE)
backward
```

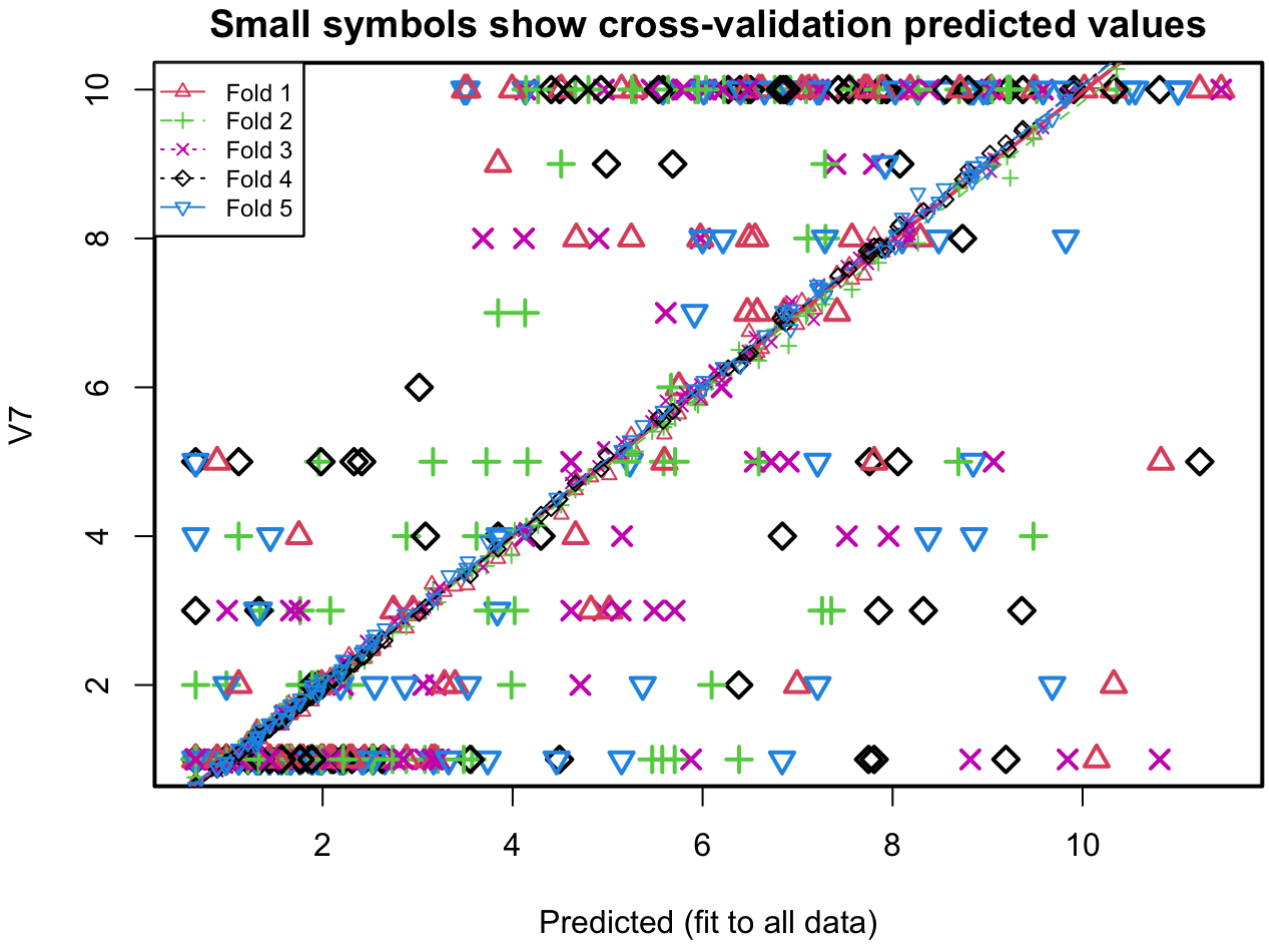
```
##
## Call:
## lm(formula = V7 ~ V2 + V4 + V5 + V8, data = cleanedDataForNonMissingRows)
##
## Coefficients:
## (Intercept)           V2           V4           V5           V8
##     -0.5360       0.2262       0.3173       0.3323       0.3238
```

```
#since both methods gave same set of columns as significant factors, using these and run
ning a model
modelStepwise<-lm( V7 ~ V4 + V5 + V8 + V2,data=cleanedDataForNonMissingRows)
summary(modelStepwise)
```

```
##
## Call:
## lm(formula = V7 ~ V4 + V5 + V8 + V2, data = cleanedDataForNonMissingRows)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8115 -0.9531 -0.3111  0.6678  8.6889
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.53601    0.17514  -3.060   0.0023 **
## V4           0.31729    0.05086   6.239 7.76e-10 ***
## V5           0.33227    0.04431   7.499 2.03e-13 ***
## V8           0.32378    0.05606   5.775 1.17e-08 ***
## V2           0.22617    0.04121   5.488 5.75e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.274 on 678 degrees of freedom
## Multiple R-squared:  0.6129, Adjusted R-squared:  0.6107
## F-statistic: 268.4 on 4 and 678 DF,  p-value: < 2.2e-16
```

```
#performing cross validation to check the performance of the model
modelCvStepwise<-cv.lm(cleanedDataForNonMissingRows,modelStepwise,m=5,printit = FALSE)
```

```
## Warning in cv.lm(cleanedDataForNonMissingRows, modelStepwise, m = 5, printit = FALS
E):
##
##  As there is >1 explanatory variable, cross-validation
##  predicted values for a fold are not a linear function
##  of corresponding overall predicted values.  Lines that
##  are shown for the different folds are approximate
```




```
SST <- sum((as.numeric(cleanedDataForNonMissingRows$V7) - mean(as.numeric(cleanedDataFor
NonMissingRows$V7)))^2)
totalSSE<-sum((modelCvStepwise$Predicted-cleanedDataForNonMissingRows$V7)^2)
R2<-1-totalSSE/SST
R2
```

```
## [1] 0.6129398
```

Stepwise and backward elimination methods both gave same set of significant parameters **V7 ~ V4 + V5 + V8 + V2**. Using these significant variables I have constructed the linear regression model and run a 5 fold cross-validation. This results in a fairly good R-squared of **61.3%**. Using this regression model, I have predicted the values of missing V7 rows. I have used round function as the value is a categorical value and cannot have fractions.

```
set.seed(777)
#Predicting the missing v7 values using the model above.
predictedV7<-round(predict(modelStepwise,newdata=cleanedDataForMissingRows))
predictedV7
```

```
##  24  41 140 146 159 165 236 250 276 293 295 298 316 322 412 618
##   5   8   1   2   1   2   3   2   2   6   1   3   5   2   1   1
```

```
regressionImputedData<-data
regressionImputedData[missingRows,]$V7<-predictedV7
```

14.1.3 Use regression with perturbation to impute values for the missing data

As mentioned in the lecture, I have used the normal distribution of missing values determined above to include in as a factor to address the variations in the data set. Using the predicted values in the above step as mean, I have created normal distribution of the predicted data. Since these values are fractions, am rounding them off. We observe few of the values are less than 0, so I have set the floor limit and cieling limit using pmax and pmin functions. I have also plotted chart showing the comparision between predicted values of V7 by a regression model v/s adjusted value with perturbations of the predicted value.

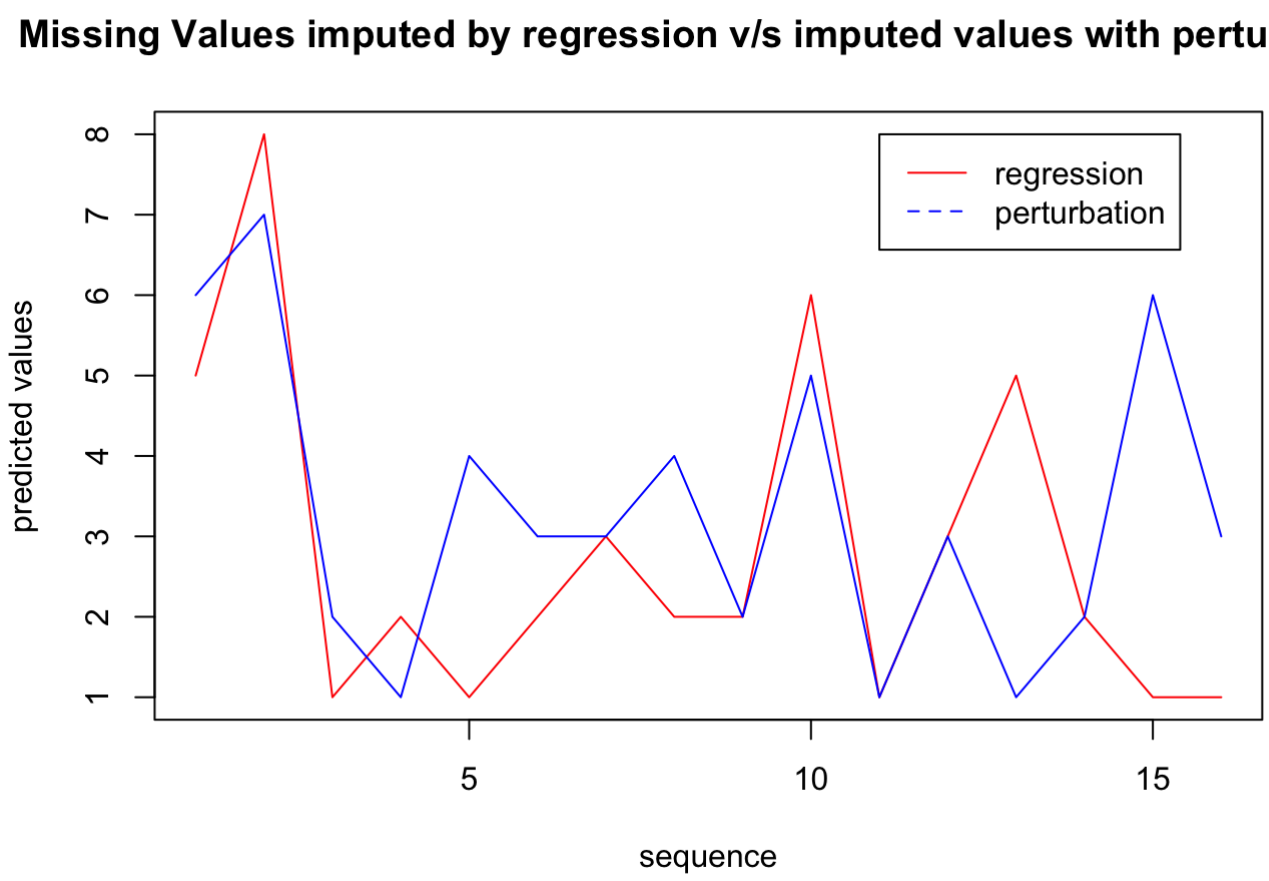
```
set.seed(777)
perturbations<-rnorm(nrow(cleanedDataForMissingRows), predictedV7, sd(predictedV7))
perturbations
```

```
##  [1] 6.0307838 7.1612463 2.0750849 1.1606767 4.4487107 3.3075074 3.4266031
##  [8] 4.3338245 1.5659879 5.2024459 0.3596635 3.1139878 1.0414714 1.9289575
## [15] 5.8646761 3.0463468
```

```
roundedPerturbations<-round(perturbations)
flooredPerturbations<-pmax(roundedPerturbations,1)
finalPertubations<-pmin(flooredPerturbations,10)
finalPertubations
```

```
##  [1] 6 7 2 1 4 3 3 4 2 5 1 3 1 2 6 3
```

```
regPerturbationImputedData<-data
regPerturbationImputedData[missingRows,]$V7<-finalPertubations

plot(seq(1,16),predictedV7,type="l",col="red",
     main="Missing Values imputed by regression v/s imputed values with perturbation",
     ylab="predicted values",
     xlab="sequence")
lines(seq(1,16),finalPertubations,type="l",col="blue")
legend(11, 8, legend=c("regression", "perturbation"),
       col=c("red", "blue"), lty=1:2)
```



14.1.4 Compare the results and quality of classification models (e.g., SVM, KNN) build using the data sets from questions 1,2,3;

Lets run KNN classification on the data set we got in the above steps and see the accuracy of each.

KNN Model on Mode Imputed data:

As we learnt in hw1, before running classification we have to split the data into training and validation set. I have randomized the data selection for training and validationDataSet. After splitting the data, I have run a quick check to see if the split of data is not biased towards one of the 2 categorical values in the response column. We see that percentage of occurence of 2 is same in validation set and training set. We can assume the split is not biased towards one particular output variable. I figured we will be running KKNN function on 4 datasets - modeImputedDataSet, meanImputedDataSet , regressionImputedDataSet, regression & perturbation data set. So I created a function which will be called for each data set. For KNN classification on modeImputed data set, we get accuracy of **93.1% with highest K at K=14.**. Note: K=1 is also giving same accuracy as K=14, but i am selecting K=14 because K=1 usually leads to overfitting data, so selecting the next best K value.

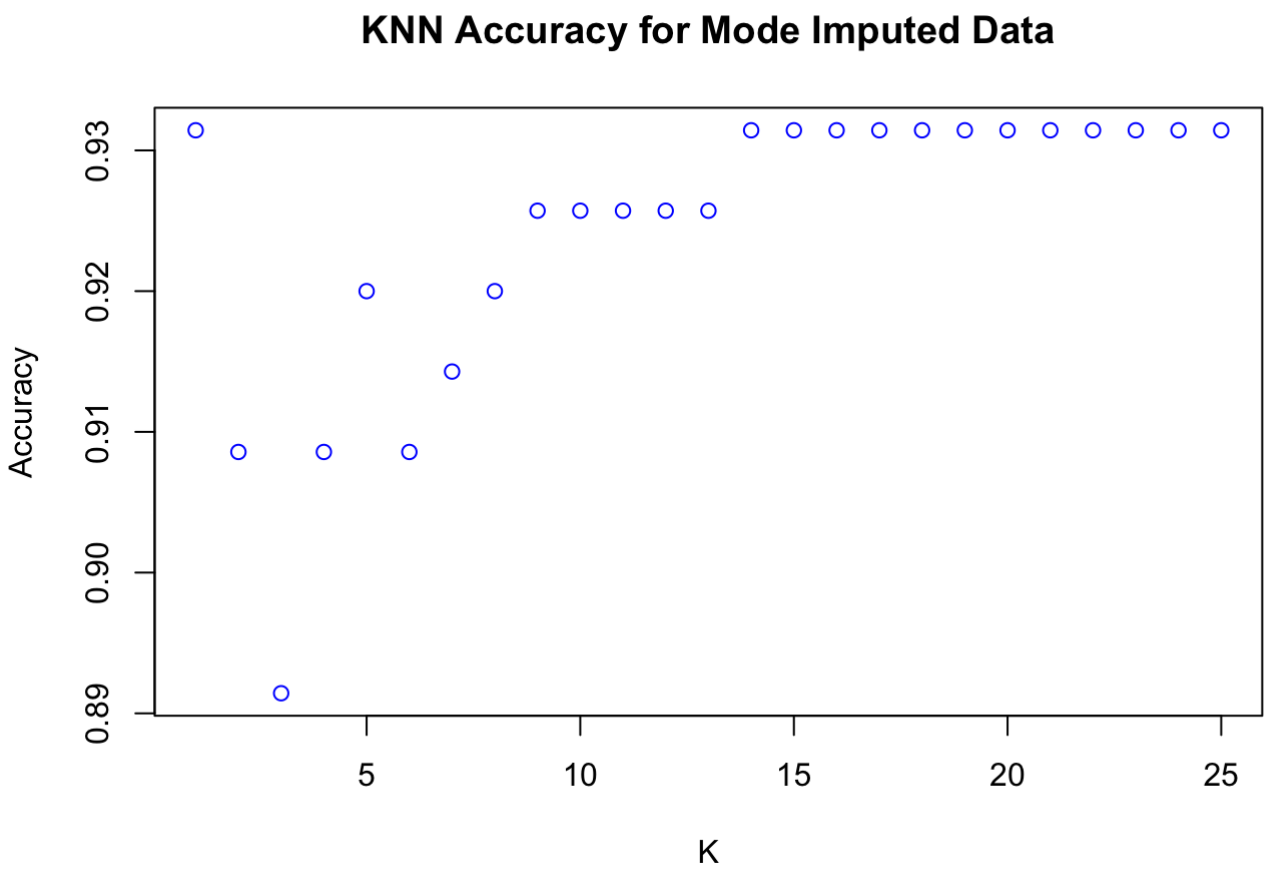
```
set.seed(777)
library(kknn)
#splitting the data into 2 sets of 75%,30% randomising the selection of the data.
ss <- sample(1:nrow(modeImputedData),round(0.75*nrow(modeImputedData)),replace=FALSE)
trainingModeDataSet <- modeImputedData[ss,]
validationModeDataSet <- modeImputedData[-ss,]

#validating if the split of data has similar mix of data
t<-table(trainingModeDataSet$V11)
v<-table(validationModeDataSet$V11)

#checking the percentage of occuerences of 2 in training and validation set.
#split is good with similar percentage of occurences of 2 and 4.
percentageOf2InTrainingSet<-(t[names(t)==2])/nrow(trainingModeDataSet)
percentageOf2InTrainingSet
```

```
##         2
## 0.6545802
```

```
percentageOf2InValidationSet<-(v[names(v)==2])/nrow(validationModeDataSet)
percentageOf2InValidationSet
```

```
##         2
## 0.6571429
```

```
#function to perform KKNN classification and returns accuracy list for each value of K
performKKNN<-function(trainingDataSet, validationDataSet){
  #creating list of K values from 1 to 25.
  kList = seq(1,25)
  #initializing frame to store accuracies for each value of k
  accuracyList <- data.frame(k=integer(),accuracy=double())
  #looping for each K
  for(k in kList){
    # looping for each value of K
    modelTesting<-kknn (formula=V11~V2+V3+V4+V5+V6+V7+V8+V9+V10,
                        train=trainingDataSet,
                        test = validationDataSet,
                        scale=TRUE,
                        kernel = "gaussian",
                        k=k,
                        distance=2)
    prediction=round(modelTesting$fitted.values)
    accuracy<-sum(prediction==validationDataSet[,11])/nrow(validationDataSet)
    accuracyList[nrow(accuracyList) + 1,] = list(k,accuracy)
  }
  return(accuracyList);
}

#call the KKNN function and store the accuracy list in the variable
accuracyListForModeImputedData<-performKKNN(trainingDataSet=trainingModeDataSet, validat
ionDataSet = validationModeDataSet)

#find the best K
bestK<-accuracyListForModeImputedData[which.max(accuracyListForModeImputedData$accurac
y),]$k

#find the best accuracy
bestAccuracy<-accuracyListForModeImputedData[which.max(accuracyListForModeImputedData$ac
curacy),]$accuracy
#plot accuracy vs k
plot(accuracyListForModeImputedData$k,accuracyListForModeImputedData$accuracy,col="blue"
,xlab="K", ylab="Accuracy",
       main="KNN Accuracy for Mode Imputed Data")
```



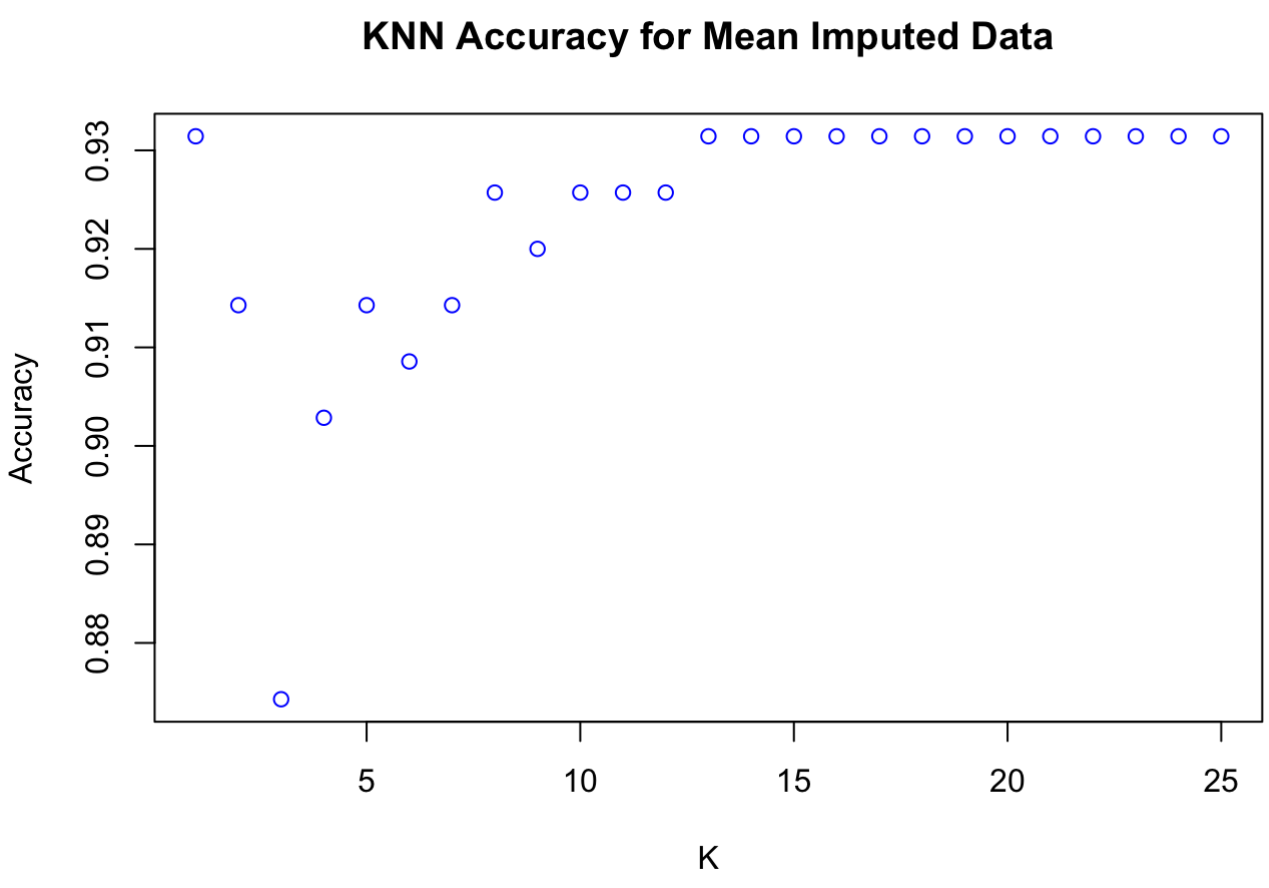

```
  cat("\nbest k= 14  with accuracy=",bestAccuracy)
```

```
##
## best k= 14  with accuracy= 0.9314286
```

KNN Model on Mean Imputed data:

Mean imputed data set also gives same result as Mode Imputed data set.

```
set.seed(777)
#splitting the data into 2 sets of 75%,25% randomising the selection of the data.
ss <- sample(1:nrow(meanImputedData),round(0.75*nrow(meanImputedData)),replace=FALSE)
trainingMeanDataSet <- meanImputedData[ss,]
validationMeanDataSet <- meanImputedData[-ss,]
#call the KKNN function and store the accuracy list in the variable
accuracyListForMeanImputedData<-performKKNN(trainingDataSet=trainingMeanDataSet, validat
ionDataSet = validationMeanDataSet)
#find the best K
bestK<-accuracyListForMeanImputedData[which.max(accuracyListForMeanImputedData$accurac
y),]$k

#find the best accuracy
bestAccuracy<-accuracyListForMeanImputedData[which.max(accuracyListForMeanImputedData$ac
curacy),]$accuracy

#plot accuracy vs K
plot(accuracyListForMeanImputedData$k,accuracyListForMeanImputedData$accuracy,col="blue"
,xlab="K", ylab="Accuracy",
     main="KNN Accuracy for Mean Imputed Data")
```

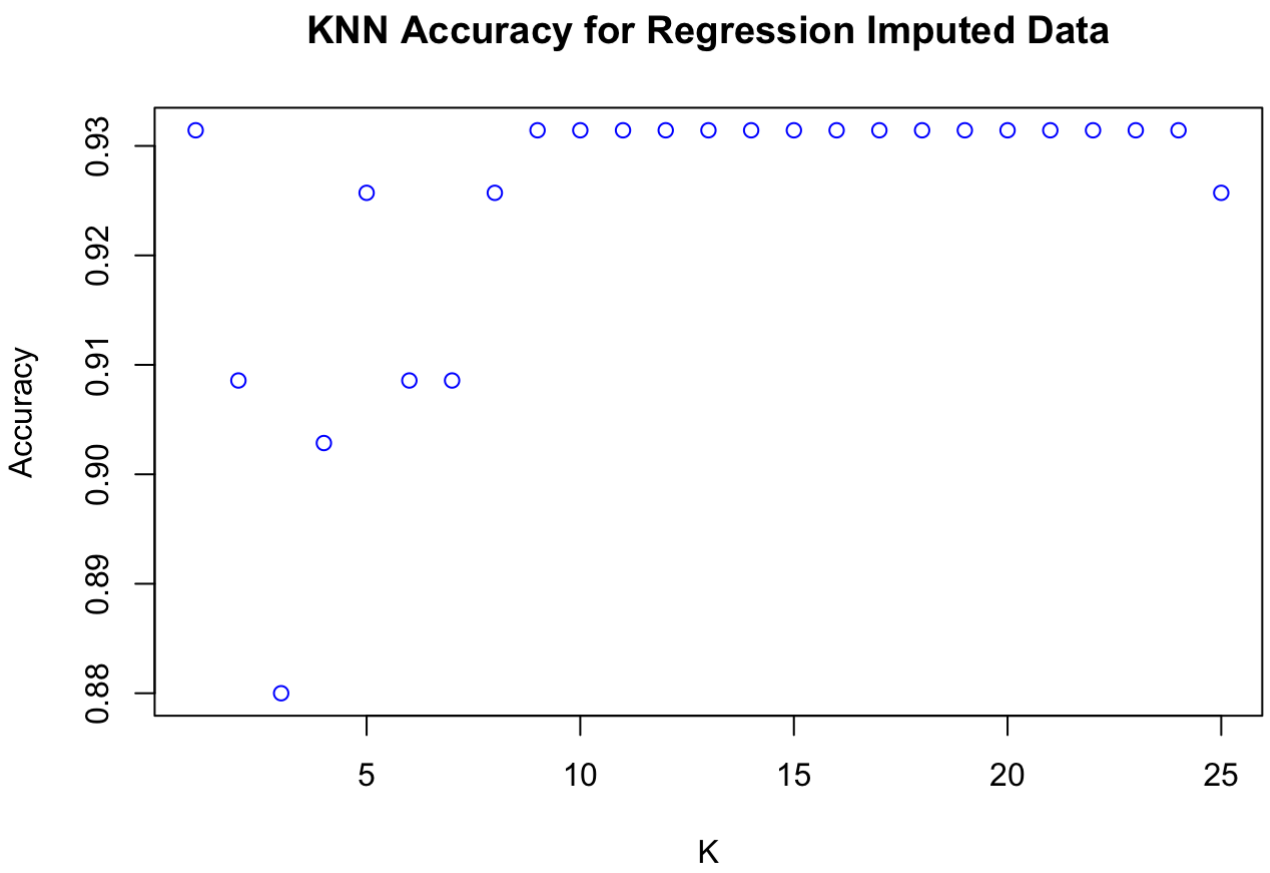



```
cat("\nbest k= 13  with accuracy=",bestAccuracy)
```

```
##
## best k= 13  with accuracy= 0.9314286
```

KNN Model on Regression Imputed data:

```
set.seed(777)
#splitting the data into 2 sets of 75%,25% randomising the selection of the data.
ss <- sample(1:nrow(regressionImputedData),round(0.75*nrow(regressionImputedData)),repla
ce=FALSE)
trainingRegDataSet <- regressionImputedData[ss,]
validationRegDataSet <- regressionImputedData[-ss,]
#call the KKNN function and store the accuracy list in the variable
accuracyListForRegressionImputedData<-performKKNN(trainingDataSet=trainingRegDataSet, va
lidationDataSet = validationRegDataSet)
#find the best K
bestK<-accuracyListForRegressionImputedData[which.max(accuracyListForRegressionImputedDa
ta$accuracy),]$k

#find the best accuracy
bestAccuracy<-accuracyListForRegressionImputedData[which.max(accuracyListForRegressionIm
putedData$accuracy),]$accuracy

#plot accuracy vs K
plot(accuracyListForRegressionImputedData$k,accuracyListForRegressionImputedData$accurac
y,col="blue",xlab="K", ylab="Accuracy",
     main="KNN Accuracy for Regression Imputed Data")
```




```
cat("\nbest k= 9  with accuracy=",bestAccuracy)
```

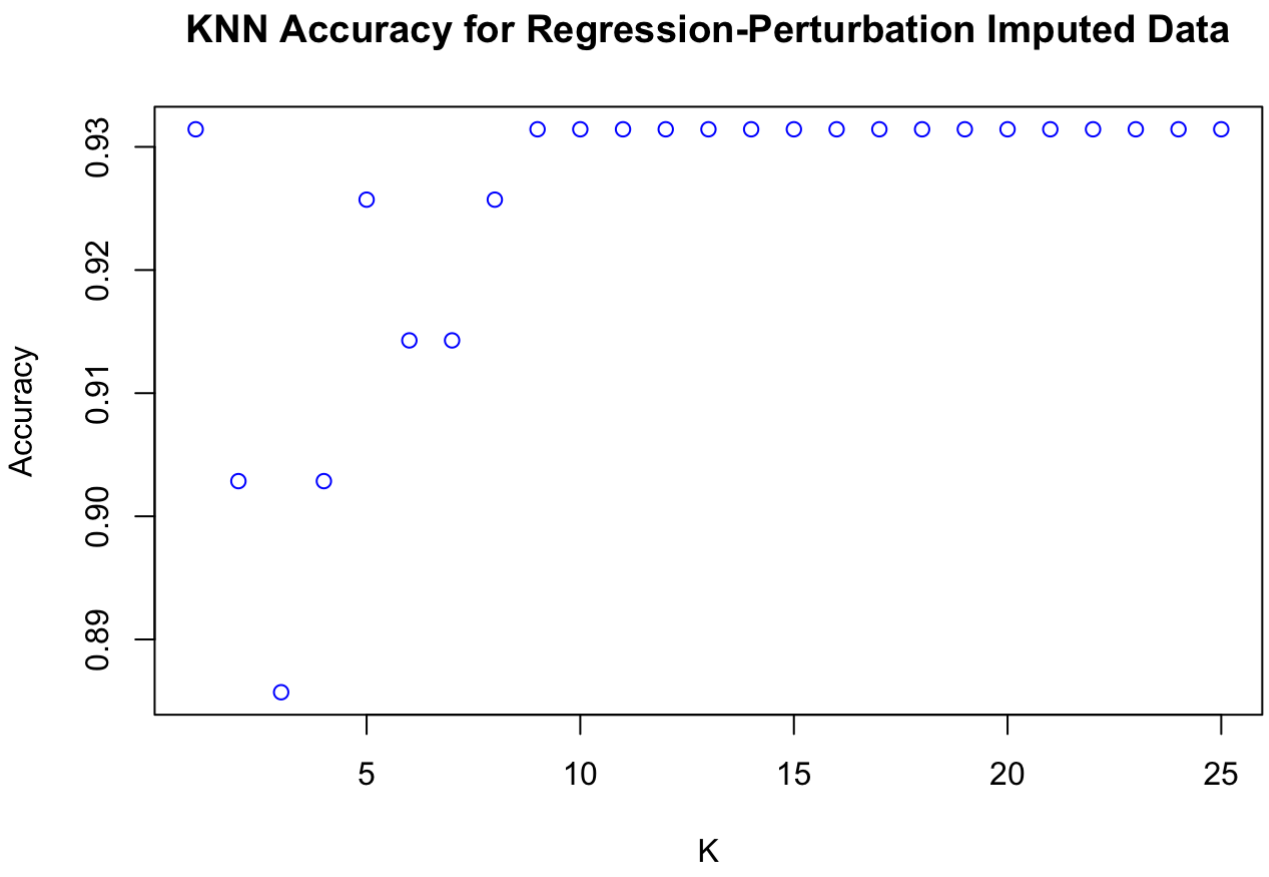
```
##
## best k= 9  with accuracy= 0.9314286
```

KNN Model on Regression -Perturbation Imputed data:

Similar approach as above, using regression-perturbation data set. We see that all the methods give almost similar accuracies at different values of K.

- Mode Imputed data, best K=14 , best-accuracy= 93.1%
- Mean Imputed data, best K=13 , best-accuracy= 93.1%
- Regression Imputed data, best K=9 , best-accuracy= 93.1%
- Regression Perturbation Imputed data, best K=9 , best-accuracy= 93.1%

```
set.seed(777)
#splitting the data into 2 sets of 75%,25% randomising the selection of the data.
ss <- sample(1:nrow(regPerturbationImputedData),round(0.75*nrow(regPerturbationImputedDa
ta)),replace=FALSE)
trainingRegPertDataSet <- regPerturbationImputedData[ss,]
validationRegPertDataSet <- regPerturbationImputedData[-ss,]
#call the KKNN function and store the accuracy list in the variable
accuracyListForRegressionPertImputedData<-performKKNN(trainingDataSet=trainingRegPertDat
aSet, validationDataSet = validationRegPertDataSet)
#find the best K
bestK<-accuracyListForRegressionPertImputedData[which.max(accuracyListForRegressionPertI
mputedData$accuracy),]$k

#find the best accuracy
bestAccuracy<-accuracyListForRegressionPertImputedData[which.max(accuracyListForRegressi
onPertImputedData$accuracy),]$accuracy

#plot accuracy vs K
plot(accuracyListForRegressionPertImputedData$k,accuracyListForRegressionPertImputedData
$accuracy,col="blue",xlab="K", ylab="Accuracy",
     main="KNN Accuracy for Regression-Perturbation Imputed Data")
```



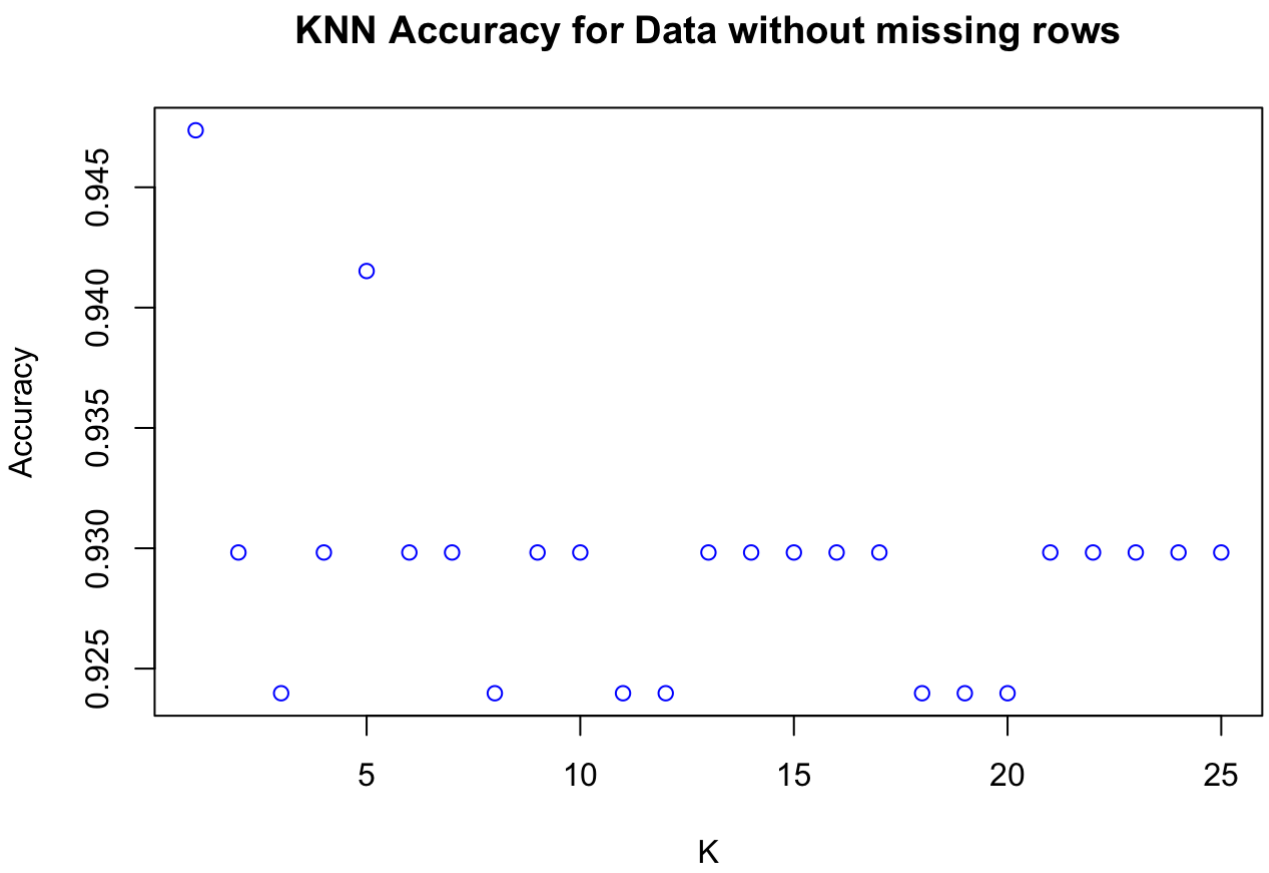

```
cat("\nbest k= 9  with accuracy=",bestAccuracy)
```

```
##
## best k= 9  with accuracy= 0.9314286
```

14.1.4.2 KNN Model on the data that remains after data points with missing values are removed; :

Running the same KKNN function with data created after removing missing rows earlier. We see the highest accuracy is for K=1, but with only 1 neighbour to classify mostly leads to overfitting, I am going with the next best accuracy of **k=5 and accuracy=94.7%**. If we compare all the output we have before with this, we see that imputing data did not perform any better in classification, since the missing data was only 2.9%, we could have excluded the data and done classification.

```
set.seed(777)
#splitting the data into 2 sets of 75%,25% randomising the selection of the data.
ss <- sample(1:nrow(dataAfterRemovingMissingRows),round(0.75*nrow(dataAfterRemovingMissi
ngRows)),replace=FALSE)
trainingNonMissingDataSet <- dataAfterRemovingMissingRows[ss,]
validationNonMissingDataSet <- dataAfterRemovingMissingRows[-ss,]
#call the KKNN function and store the accuracy list in the variable
accuracyListForNonMissingData<-performKKNN(trainingDataSet=trainingNonMissingDataSet, va
lidationDataSet = validationNonMissingDataSet)
#find the best K
bestK<-accuracyListForNonMissingData[which.max(accuracyListForNonMissingData$accuracy),]
$k

#find the best accuracy
bestAccuracy<-accuracyListForNonMissingData[which.max(accuracyListForNonMissingData$accu
racy),]$accuracy

#plot accuracy vs K
plot(accuracyListForNonMissingData$k,accuracyListForNonMissingData$accuracy,col="blue",x
lab="K", ylab="Accuracy",
     main="KNN Accuracy for Data without missing rows")
```



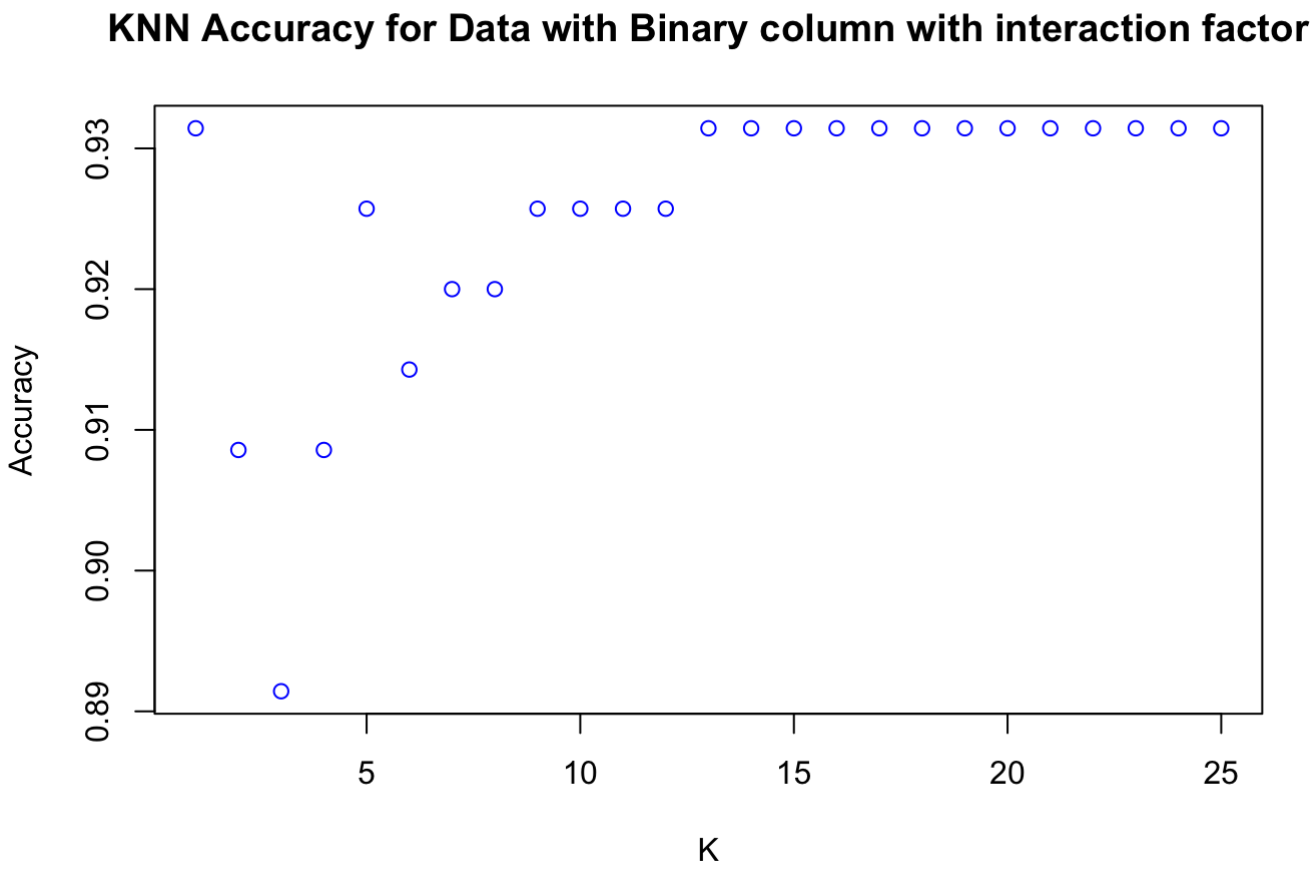

```
cat("\nbest k= 5 or   with accuracy=0.942")
```

```
##
## best k= 5 or   with accuracy=0.942
```

14.1.4.3 KNN Model on the data set when a binary variable is introduced to indicate missing values.

As mentioned in the lecture, I have added a new column V12, which has value of 0 or 1 based on if the value of V7 is missing or present. Added 13th column which will have interaction factor by multiplying the 12th column with the 7th column. Next run the same KKNN including column 13 instead of column 7. We find the **highest accuracy as 93.1 at K = 13**, again am not considering K=1 as that also similar accuracy.

```
set.seed(777)
binaryData<-data

#adding binary column, 0 for all missing rows, 1 for all non missing row
binaryData[missingRows,12]<-0
binaryData[-missingRows,12]<-1

# creating interaction factor by multiplying above column with binary value
binaryData[missingRows,13]<-0;
binaryData[-missingRows,13]<-binaryData[-missingRows,12]*binaryData[-missingRows,7]

#splitting the data into 2 sets of 75%,25% randomising the selection of the data.
ss <- sample(1:nrow(binaryData),round(0.75*nrow(binaryData)),replace=FALSE)
trainingBinaryDataSet <- binaryData[ss,]
validationBinaryDataSet <- binaryData[-ss,]
#perform KNN
#creating list of K values from 1 to 25.
  kList = seq(1,25)
  #initializing frame to store accuracies for each value of k
  accuracyBinaryList <- data.frame(k=integer(),accuracy=double())
  #looping for each K
  for(k in kList){
    # looping for each value of K
    modelTesting<-kknn (formula=V11~V2+V3+V4+V5+V6+V8+V9+V10+V13,
                        train=trainingBinaryDataSet,
                        test = validationBinaryDataSet,
                        scale=TRUE,
                        kernel = "gaussian",
                        k=k,
                        distance=2)
    prediction=round(modelTesting$fitted.values)
    accuracy<-sum(prediction==validationBinaryDataSet[,11])/nrow(validationBinaryDataSe
t)
    accuracyBinaryList[nrow(accuracyBinaryList) + 1,] = list(k,accuracy)
  }

#find the best K
bestK<-accuracyBinaryList[which.max(accuracyBinaryList$accuracy),]$k

#find the best accuracy
bestAccuracy<-accuracyBinaryList[which.max(accuracyBinaryList$accuracy),]$accuracy

#plot accuracy vs K
plot(accuracyBinaryList$k,accuracyBinaryList$accuracy,col="blue",xlab="K", ylab="Accurac
y",
     main="KNN Accuracy for Data with Binary column with interaction factor")
```




```
cat("\nbest k= 13 with accuracy=0.931")
```

```
##
## best k= 13 with accuracy=0.931
```

14.1.4.3 KSVM Model on all the data sets

I have performed the KSVM calculations as well similar to KNN above.

```
set.seed(777)
library(kernlab)
#function to perform KKNN classification and returns accuracy list for each value of K
performKSVM<-function(trainingDataSet, validationDataSet){
  #creating list of K values from 1 to 25.
  kList = seq(1,25)
  #initializing frame to store accuracies for each value of k
  accuracyList <- data.frame(c=integer(),accuracy=double())
  #looping for each K
  seqC<-c(0.1,1,10,20,100,1000)
  for(c in seqC){
    # looping for each value of K
    model<-ksvm(as.matrix(trainingDataSet[,2:10]),
                      as.factor(trainingDataSet[,11]),
                      type="C-svc",
                      kernel="vanilladot",
                      C=c,
                      scaled=TRUE)
   # see what the model predicts
    pred <- predict(model,validationDataSet[,2:10])
    accuracy<-sum(pred == validationDataSet[,11]) / nrow(validationDataSet)
    accuracyList[nrow(accuracyList) + 1,] = list(c,accuracy)
  }
  return(accuracyList);
}

ksvmAccuracyListForModeImputedData<-performKSVM(trainingDataSet=trainingModeDataSet,
                                                validationDataSet = validationModeDataSe
t)
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
ksvmAccuracyListForMeanImputedData<-performKSVM(trainingDataSet=trainingMeanDataSet,
                                                validationDataSet = validationMeanDataSe
t)
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
ksvmAccuracyListForRegImputedData<-performKSVM(trainingDataSet=trainingRegDataSet,
                                               validationDataSet = validationRegDataSet)
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
ksvmAccuracyListForRegPertImputedData<-performKSVM(trainingDataSet=trainingRegPertDataSe
t,
                                                   validationDataSet = validationRegPert
DataSet)
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
ksvmAccuracyListForNonMissingData<-performKSVM(trainingDataSet=trainingNonMissingDataSe
t,
                                                 validationDataSet = validationNonMissingD
ataSet)
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
ksvmAccuracies<-data.frame(ksvmAccuracyListForMeanImputedData$c
                           , ksvmAccuracyListForMeanImputedData$accuracy
                           , ksvmAccuracyListForModeImputedData$accuracy
                           , ksvmAccuracyListForRegImputedData$accuracy
                           , ksvmAccuracyListForRegPertImputedData$accuracy
                           , ksvmAccuracyListForNonMissingData$accuracy)

colnames(ksvmAccuracies)<-c("c","ModeImputed","MeanImputed",
                            "RegressionImputed", "Reg-Pert Imputed","NonMissingData")
ksvmAccuracies
```

```
##       c ModeImputed MeanImputed RegressionImputed Reg-Pert Imputed
## 1 1e-01   0.9428571   0.9428571         0.9428571        0.9428571
## 2 1e+00   0.9485714   0.9485714         0.9485714        0.9485714
## 3 1e+01   0.9485714   0.9485714         0.9485714        0.9485714
## 4 2e+01   0.9485714   0.9485714         0.9485714        0.9485714
## 5 1e+02   0.9485714   0.9485714         0.9485714        0.9485714
## 6 1e+03   0.9485714   0.9485714         0.9485714        0.9485714
##   NonMissingData
## 1      0.9590643
## 2      0.9590643
## 3      0.9649123
## 4      0.9649123
## 5      0.9649123
## 6      0.9649123
```

Question 15.1 Describe a situation or problem from your job, everyday life, current events, etc., for which optimization would be appropriate. What data would you need?

Answer: I am currently working as application developer in one of the leading insurance companies. Me and my team are responsible for all the client facing applications like company website, mobile apps, alexa , google home and numerous chat bots. We can use optimization for prescriptive analysis to decide how many server instances we have to run to handle the varying network traffic that comes into our systems. With cloud based architecture data and computation costs have come down a lot but still there is a scope to further fine tune the usage. We can optimize the number of content delivey servers we have to maintain so that users do not see delay when webpages load on their devices. We can fine tune the search engine optimization parameters which will place our webpages ahead of our competitiors. The amount of traffic that comes in into our network depends on various factors like:

- Time of the day: 8 am EST - 10:00 pm EST usually have higher traffic which slows down before and after.
- Any new announcements in the next few days: Any promotional events will have unusually high network activity around the announcement dates.
- Network maintainence schedules: Maintenance like server patching, fixing vulnerabilities reduces network availability to our customers.
- Federal Interest Rates changes - Whenever interest rates drop, we see more customers trying to get insurance quotes.
- Employment rates: High employment rates relates to more number of people buying houses and applying for mortgage insurance.