Team A - Deliverable 5: Model Selection

Jason Leonard
Susmitha Abbaiahvari
Mamata Khadka

IS 777 - Data Analytics

Dr. Koru

Index

Deliverable 1

Considered Datasets

Yearly Data Air Quality Index from the EPA (1980 - 2021)
(Source / Location)

This dataset consisted of ~34k rows/observations **(*n* ≈ 34,000)** and 21 columns, of which 5 (shown in *Figure 1.0*) were selected as potential predictor variables **(*p* = 4)**, which would have been used to predict the number of ***Good Days*** (type: numerical) in the context of air quality.

Figure 1.0

Variable Name	**Type**
State	Categorical
County	Categorical
Year	Numerical
Days with AQI (Air Quality Index)	Numerical

We believed this dataset to be very applicable to the topic of this class -- Healthcare. Climate change has a well documented affect on healthcare and the general well being of citizens worldwide (source). Studying the trend of air quality would allow healthcare experts to better prepare for worsening conditions in regards to respiratory issues and ensure they would not be ill-prepared with low resources. Ultimately, this dataset and project was not selected because the resulting model would be too complicated and out of the scope of this class.

In Hospital Mortality Prediction
(Source / Location)

This dataset consisted of 1,177 rows/observations **(*n* = 1,177)** and 51 columns, of which 8 (shown in *Figure 2.0*) were selected as potential predictor variables **(*p* = 8)**, which would have been used to predict ***Outcome*** (type: categorical) which represented if a patient died or remained alive while their stay in the Hospital.

Figure 2.0

Variable Name	**Type**
Age	Numerical
Gender	Categorical
BMI	Numerical
Hypertensive	Categorical
Diabetes	Categorical
Respiratory Rate	Numercial
Ethnicity	Categorical
Mean Blood Pressure	Numercial

Management of resources is very difficult in a hospital setting as it is difficult to know who may need the most care at any given time. Utilizing this dataset to predict specific patients who displayed the highest potential of dying could help healthcare providers better allocate their existing resources to those with the highest risk, before their condition endangers their life. The primary reason this dataset was not used is because the BMI variable was missing for 28% of observations. Considering this was substantially above the 5% margin allowed for case-wise deletion the dataset was not selected.

Heart Failure Prediction (Cardiovascular Disease)
(Source / Location)

This dataset consisted of 299 rows/observations **(*n* = 299)** and 13 columns, of which 8 (*shown in Figure 3.0*) were selected to be potential predictor variables **(*p* = 8)**; These could potentially be used to predict ***Death Event*** (type: categorical) which represented whether a patient died from Heart Failure or not

Figure 3.0

Variable Name	**Type**
Age	Numerical
Sex	Categorical
Diabetes	Categorical
High Blood Pressure	Categorical
Smoking	Categorical
Anaemia	Categorical
Ejection Fraction	Numerical
Platelets	Numerical

This data was optimal for our team as it was relatively well formatted and had no missing data. If a model was created using this database, it would better identify patients who were most at risk from suffering heart failure given their biological features, previous diagnosis, and lifestyle habits. However, considering this dataset was extremely small (only 299 observations) we felt it was not adequate in size. Additionally, after searching the internet, this dataset has been used many times by various groups and individuals to perform statistical analysis therefore it outright does not suit the criteria of the project.

2020 BRFSS Survey Data and Documentation
(Source / Location) The survey data and its corresponding codebook are available from the "Survey Data and Documentation" tab.

This dataset contains 401,958 rows/observations **(*n* = 401,958)** and 279 columns, of which 8 (*shown in Figure 4.0*) were selected as potential predictor variables **(p = 8)**. We wanted to predict a variable that could potentially correlate with both socioeconomic and biological factors, so we decided to choose Body Mass Index (BMI), designated as ***BMI_5*** in the data set (type: numerical).

Figure 4.0

Variable	**Variable Description**	**Type**
SEXVAR	Sex of respondent	Categorical
GENHLTH	General Health	Categorical
PHYS HLTH	Number of days physical health is not good in the past 30 days	Numerical
MENT HLTH	Number of days mental health is not good in the past 30 days	Numerical
EXERANY2	Exercise in the past 30 days	Numerical
SLEPTIM1	Average hours of sleep in a 24-hour period	Numerical
EDUCA	Education Level	Categorical
EMPLOY1	Employment Status	Categorical

The BRFSS dataset is a nationally representative sample of the United States conducted by the U.S. Centers for Disease Control and Prevention (CDC). The data represents all 50 states and is considered as one of the most comprehensive behavior data sets in the U.S. BRFSS collects the data using the standardized protocol of data collection that minimizes the potential observational bias that may occur during the telephone surveys. According to the CDC, the US obesity prevalence was 42% in 2017-2018. Over the past few decades the proportion of obese US adults have significantly increased. Since obesity is directly linked to various chronic health conditions including heart disease, stroke, type 2 diabetes, and cancer, it is important to reduce the burden of obesity and health conditions. Therefore, we selected BMI as our outcome for this project and the most appropriate dataset available was BRFSS. Although BRFSS was the best suited dataset, we did not select this dataset because of the relatively higher proportion of missing values for the predictors.

COVID-19 Estimated ICU Beds Occupied by State (time series)
(Source / Location)

There are 1613 rows/observations for this dataset **(*n* = 1613)** and 12 rows, of which 5 (Shown in Figure 5.0) were selected as predictor variables **(*p* = 5)**. The outcome variable of interest was the **number of available beds** (type: numerical) both on a national and state level.

Figure 5.0

Variable	**Type**
State	Categorical
Collection Date	Numerical
Staffed adult ICU beds occupied estimated	Numerical
Total Staffed Adult ICU Beds	Numerical
Percentage of staffed adult ICU beds occupied estimated	Numerical

COVID-19 and its related hospitalization have been a major concern all over the world including U.S. During the early phase of the pandemic health systems and hospitals were occupied with infected individuals and the admittance to the hospital beds were granted based on the severity of the disease. Therefore, it is highly relevant to predict the number of ICU beds in each state. However, the available dataset was not selected because of the time series nature of the data and the required method to adequately perform statistical analysis on this data set is not in the scope . Furthermore, there were very limited predictors to be used in the regression analysis.

<u>**Selected Dataset**</u>

Breast Cancer Dataset
(<u>Source / Location</u>)

This dataset consists of 683 observations **(*n* = 683)** and 9 columns of which all 9 (see *Figure 6.0*) are considered as potential predictor variables **(*p* = 9)**; Which would be used to predict ***Class*** (type: categorical) which identifies a tumor as benign or malignant.

Figure 6.0

Variable Name	**Type**
Clump_Thickness	Numerical
Uniformity of Cell Size	Numerical
Uniformity of Cell Shape	Numerical
Marginal Adhesion	Numerical
Single Epithelial Cell Size	Numerical
Bare Nuclei	Numerical
Bland Chromatin	Numerical
Normal Nucleoli	Numerical
Mitoses	Numerical

This dataset met all the criterias listed in the deliverable, and its application in the realm of breast cancer detection made it extremely relative to Healthcare. A prediction model using this dataset could help doctors quickly identify patients who have the most likelihood to have a malignant tumor and prioritize those patients. Considering no one in the group has had significant experience with statistical analysis, this dataset would prove a good opportunity to learn more about statistical application of methods we have learned. This particular dataset does have a relatively small sample size compared to the other dataset, so it would prove challenging to deliver adequate results, however it should also provide the group with a good benchmark of an adequate sample required for various statistical models.

Deliverable 2

Data Cleaning
The data within our dataset was fully intact and did not contain any missing values. We dropped the “Sample Code Number” column/variable as it was not directly related to predicting the outcome and converted the response variable (Class) from numerical values into strings for easier interpretation in code. In the original dataset 2 = Benign and 4 = Malignant, we simply replaced the numbers with their string representations.

Number of Observations
n = 683

Number of Parameters
p = 9

Response Variable

Class

Predictor Variables

Clump Thickness
Uniformity of Cell Size
Uniformity of Cell Shape
Marginal Adhesion
Single Epithelial Cell Size
Bare Nuclei
Bland Chromatin
Normal Nucleoli
Mitoses

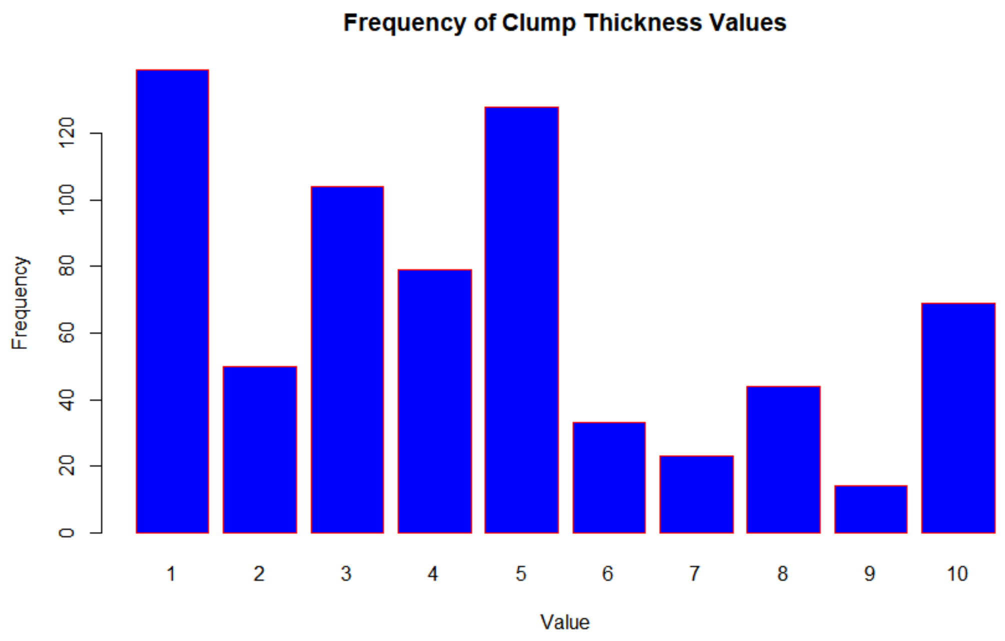
Descriptive Analysis

Summary Statistics

Parameter Name	Min.	1st Quartile	Median	Mean	3rd Quartile	Max
Clump Thickness	1.000	2.000	4.000	4.442	6.000	10.000
Uniformity of Cell Size	1.000	1.000	1.000	3.151	5.000	10.000
Uniformity of Cell Shape	1.000	1.000	1.000	3.215	5.000	10.000
Marginal Adhesion	1.000	1.000	1.000	2.83	4.00	10.000
Single Epithelial Cell Size	1.000	2.000	2.000	3.234	4.000	10.000
Bare Nuclei	1.000	1.000	1.000	3.445	6.000	10.000
Bland Chromatin	1.000	2.000	3.000	3.445	5.000	10.000
Normal Nucleoli	1.000	1.00	1.00	2.87	4.00	10.000
Mitoses	1.000	1.000	1.000	1.603	1.000	10.000
Class (Output Variable)	2.0	2.0	2.0	2.7	4.0	4.0

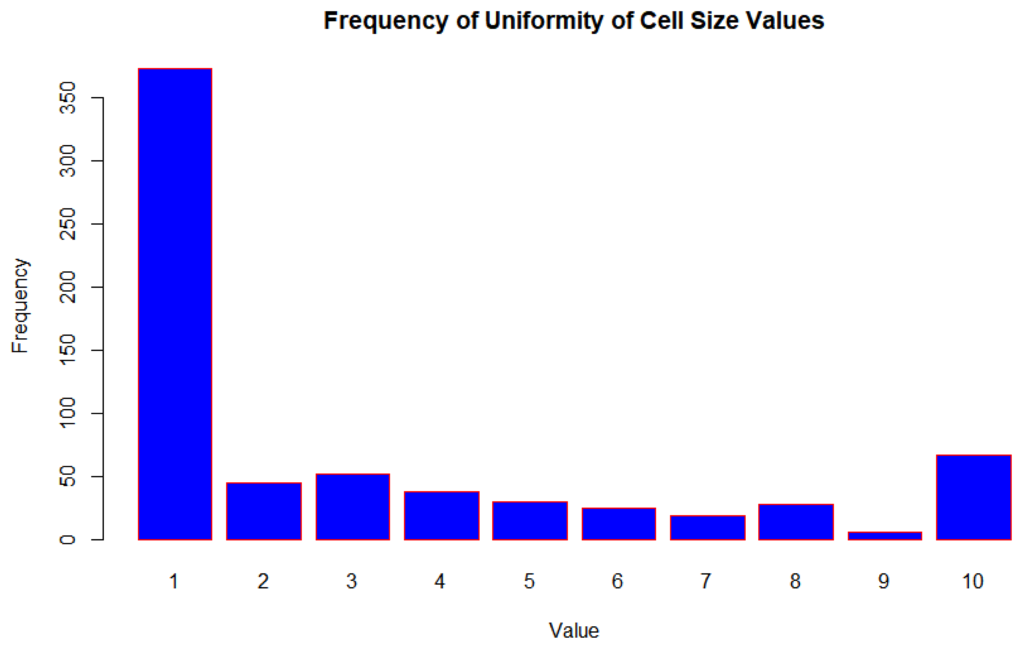
Parameter Bar Plots & Analysis

Clump Thickness



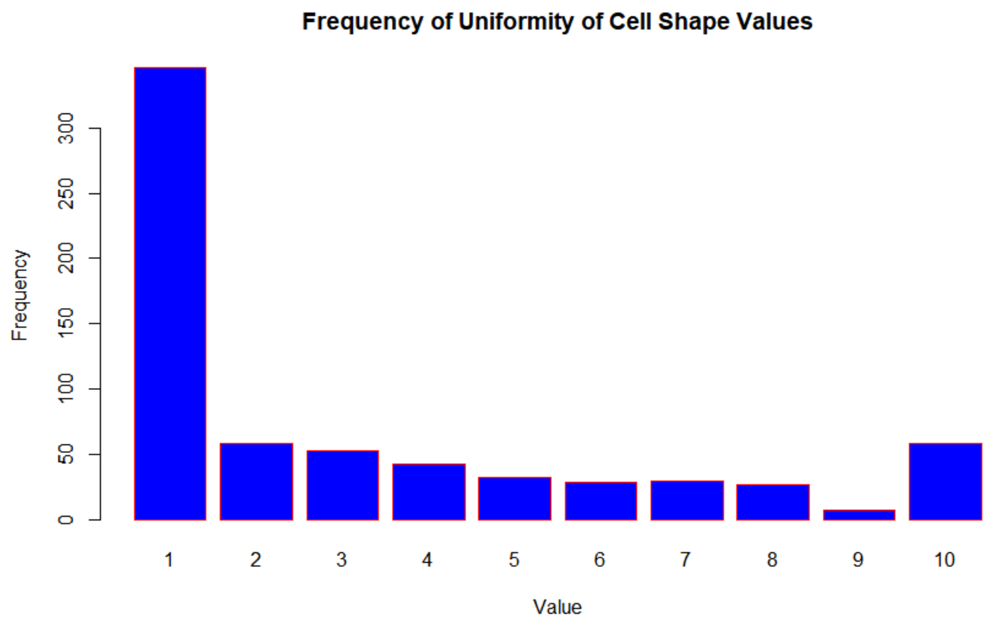
Values	Frequency (n)	Percentage (%)
1	139	20.35
2	50	7.32
3	104	15.23
4	79	11.57
5	128	18.74
6	33	4.83
7	23	3.37
8	44	6.44
9	14	2.05
10	69	10.10
Total	683	100

Uniformity of Cell Size



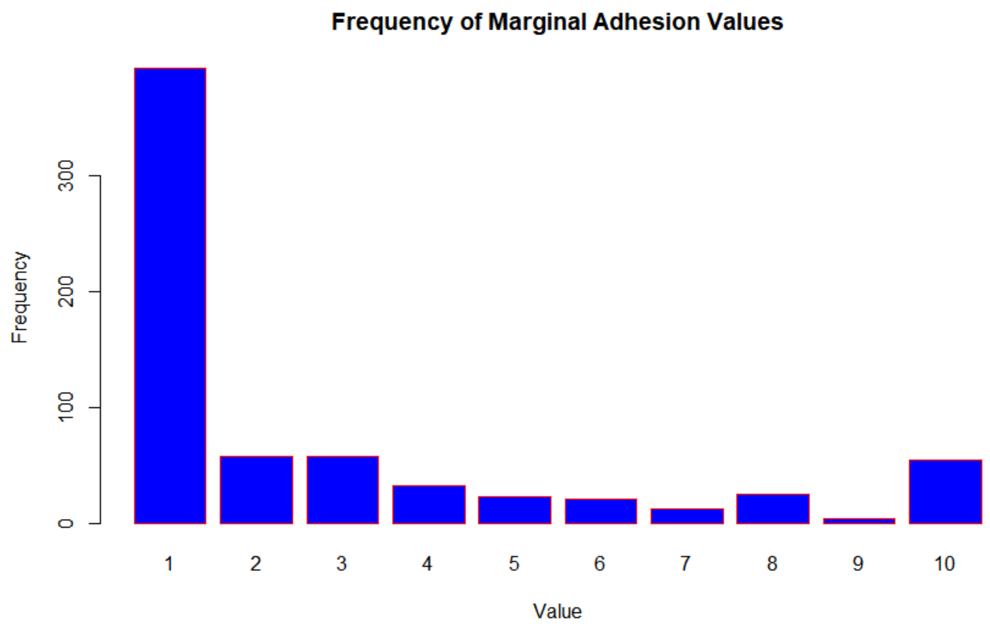
Values	Frequency (n)	Percentage (%)
1	373	54.61
2	45	6.59
3	52	7.61
4	38	5.56
5	30	4.39
6	25	3.66
7	19	2.78
8	28	4.10
9	6	0.88
10	67	9.81
Total	683	100

Uniformity of Cell Shape



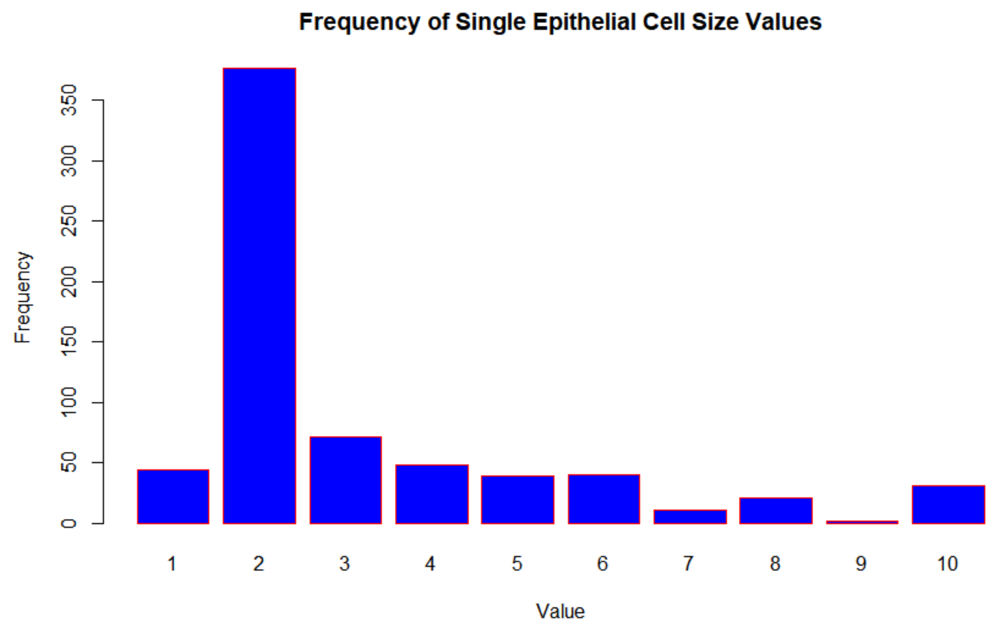
Values	Frequency (n)	Percentage (%)
1	346	50.66
2	58	8.49
3	53	7.76
4	43	6.30
5	32	4.69
6	29	4.25
7	30	4.39
8	27	3.95
9	7	1.02
10	58	8.49
Total	683	100

Marginal Adhesion



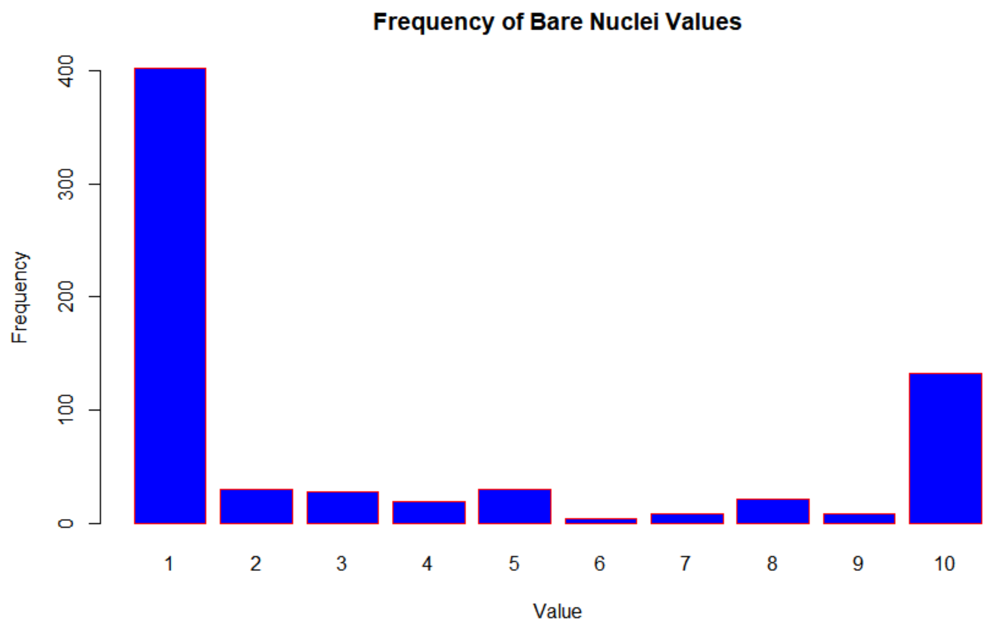
Values	Frequency (n)	Percentage (%)
1	393	57.54
2	58	8.49
3	58	8.49
4	33	4.83
5	23	3.37
6	21	3.07
7	13	1.90
8	25	3.66
9	4	0.59
10	55	8.05
Total	683	100

Single Epithelial Cell Size



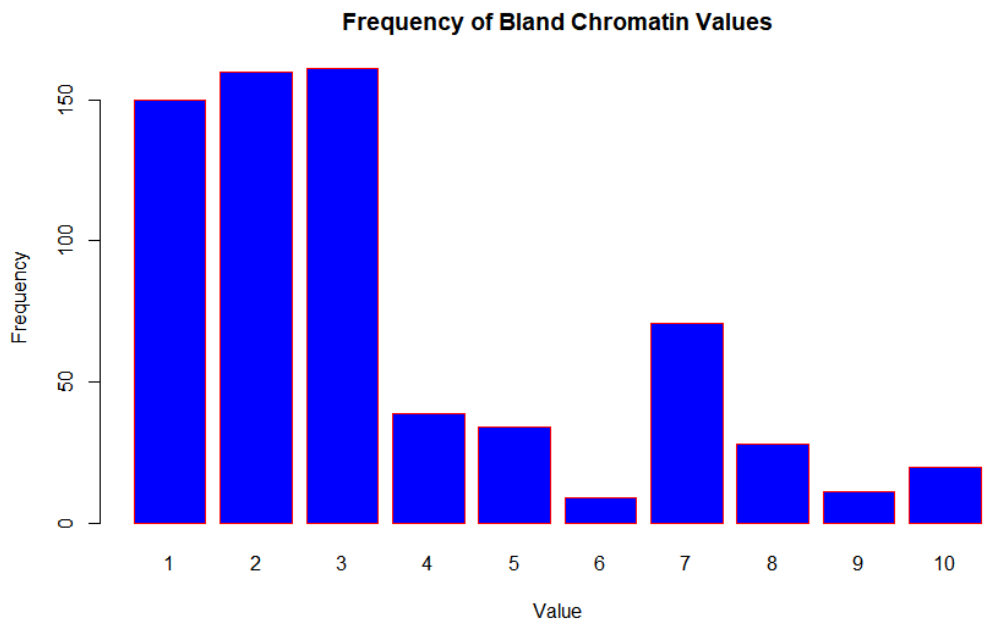
Values	Frequency (n)	Percentage (%)
1	44	6.44
2	376	55.05
3	71	10.40
4	48	7.03
5	39	5.71
6	40	5.86
7	11	1.61
8	21	3.07
9	2	0.29
10	31	4.54
Total	683	100

Bare Nuclei



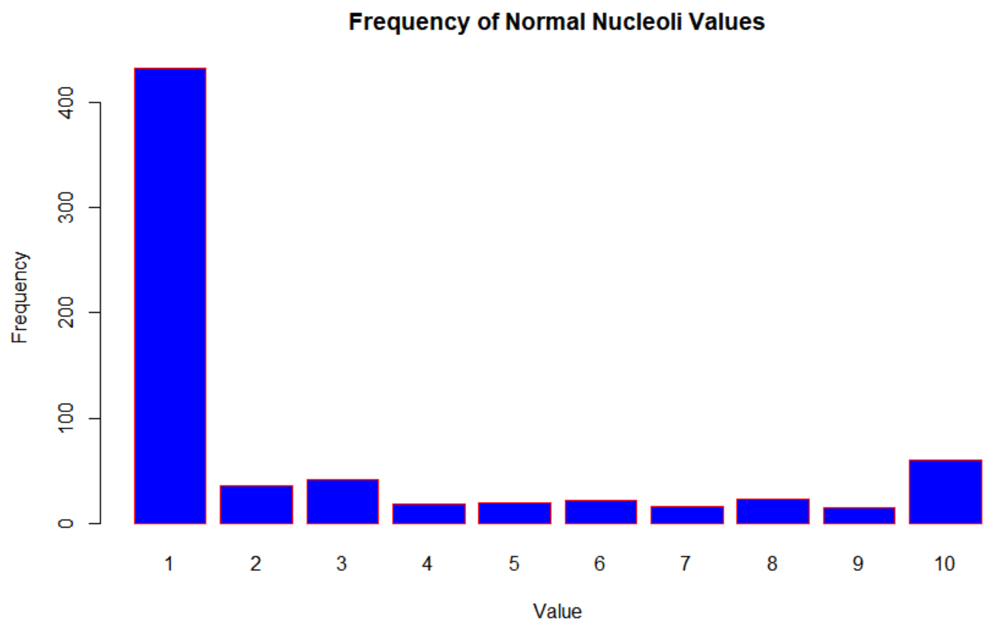
Values	Frequency (n)	Percentage (%)
1	402	58.86
2	30	4.39
3	28	4.10
4	19	2.78
5	30	4.39
6	4	0.59
7	8	1.17
8	21	3.07
9	9	1.32
10	132	19.33
Total	683	100

Bland Chromatin



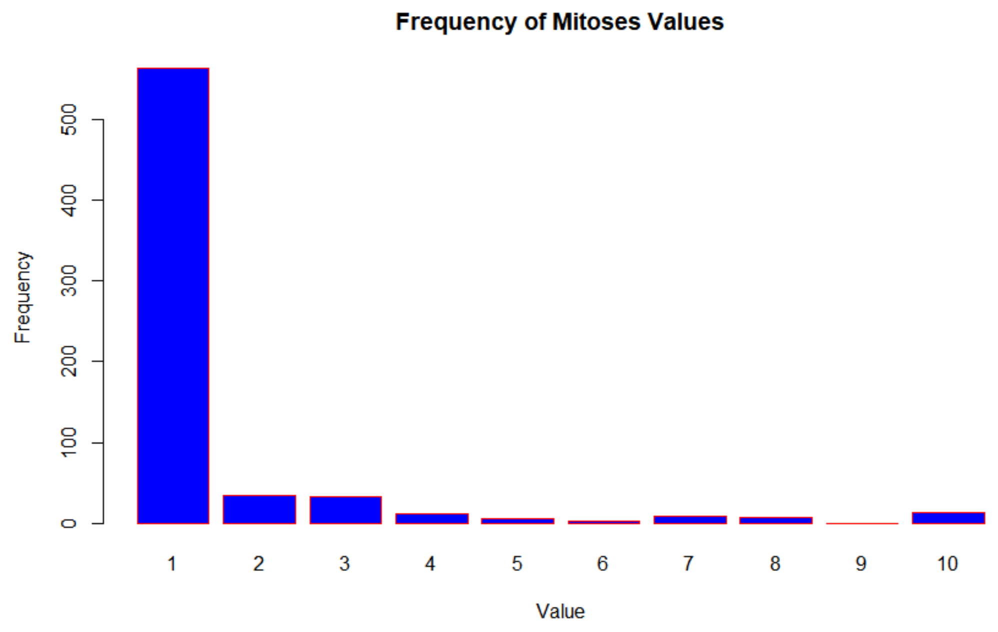
Values	Frequency (n)	Percentage (%)
1	150	21.96
2	160	23.43
3	161	23.57
4	39	5.71
5	34	4.98
6	9	1.32
7	71	10.4
8	28	4.1
9	11	1.61
10	20	2.93
Total	683	100

Normal Nucleoli



Values	Frequency (n)	Percentage (%)
1	432	63.25
2	36	5.27
3	42	6.15
4	18	2.64
5	19	2.78
6	22	3.22
7	16	2.34
8	23	3.37
9	15	2.2
10	60	8.78
Total	683	100

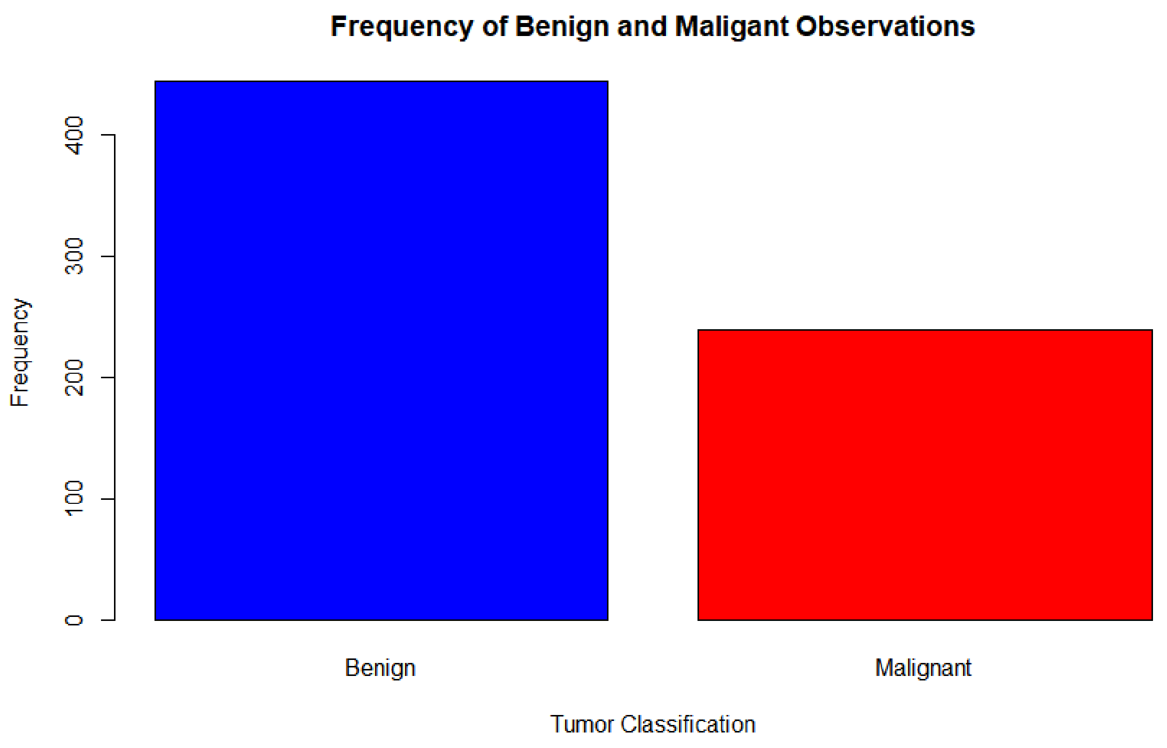
Mitoses



Values	Frequency (n)	Percentage (%)
1	563	82.43
2	35	5.12
3	33	4.83
4	12	1.76
5	6	0.88
6	3	0.44
7	9	1.32
8	8	1.17
10	14	2.05
Total	683	100

Output Variable Bar Plot & Analysis

Class



Levels	Frequency (n)	Percentage (%)
Benign	444	65.01
Malignant	239	34.99
Total	683	100

Analysis Plan

Considering our outcome variable ("*Class*") is binary, **Logistic regression** seems like a good model to implement as it would be relatively easy to interpret our data because it's probability is naturally bound between 0 and 1. **Linear Discriminant Analysis** (LDA) may also be a beneficiary model if our dataset has a good amount of separation, it would be worthwhile to visually inspect the data using scatter plots. However since the dataset contains 9 parameters, there may be some separation that exists in the data that is not depicted on a 2D or even 3D chart. The most interesting aspect of LDA is the ability to modify the threshold to suit different requirements of the model. In our case, we would need a model that accurately predicts if a tumor is Malignant, as a false positive is preferable to a false negative -- a false negative meaning the patient has a malignant tumor, but was classified as benign and a false positive meaning the opposite. Given this preference, LDA's Threshold modification could lead to a smaller error rate for misclassified malignant tumors. However, since both Logistic Regression and LDA rely on linear decision boundaries, if our data does not have easily separable clusters via linear functions and both models perform rather poorly, then QDA or even KNN will be considered. Ofcourse, because **Bayes Classifier** is considered the "unattainable gold standard" of classifiers it will also be potentially implemented as a model, primarily to set a benchmark for the other models we implement. Our response variable for this dataset is called "*Class*" which defines if an observation is a Malignant (denoted as 4 in the dataset) or Benign (denoted as 2 in the dataset). For now, we are considering all of our columns (except *Sample Code Number*) as predictors. We plan to use a backward selection approach to determine which variables are statistically significant as we have 9 parameters and would like to reduce *p* if possible.

For pedagogical purposes we plan to implement **Linear Regression** by converting our Logistic Regression estimated probabilities into log-odds to use as a numerical outcome variable. We are most interested in testing to see if certain methods like the use of interaction terms and non-linear transformations can help lower Test MSE in the model. Ofcourse, Linear Regression will not be used to actually predict if a tumor is Benign or Malignant using this approach, as the dummy variable approach would be much more applicable, but still not as effective as simply utilizing Logistic Regression, and other classifiers build to handle predicting categorical outcomes. We are more interested in simply going through the process of implementing the various methods discussed in the course.

Deliverable 3

Predictor Relationships

Covariance Matrix
To better understand our data before analysis we created a Covariance Matrix because all of our variables are constructed on the exact same scale (scale of 1-10), and thus it could provide us with a better understanding of how all of our variables interacted with each other. Additionally, since the covariance of any variable and itself reduces to that variable's variance, this would also conveniently display the variance of all our variables. Before we start coding, we install the various packages and libraries used in this deliverable by using the **install.packages()** and **library()** functions.

```
install.packages('caTools')
library(caTools)

install.packages('Metrics')
library(Metrics)

install.packages('xtable')
library(xtable)
```

To start we first set our working directory to the same folder of our data set either by RStudio's GUI or by the command **setwd()** and then import our CSV file/data set using the **read.csv()** function with the header=T modifier since our dataset does contain labels for each of the columns. We also ensure that R understands that our response variable is categorical by using the **as.factor()** function. This makes it so there are no issues during modeling, as we found sometimes R will not convert values from CSV into variables accurately.

```
setwd("C:/Users/absus/OneDrive/Desktop/777/Project")

MyData = read.csv("Breast Cancer Prediction_Clean.csv",header=T)

MyData$Class <- as.factor(MyData$Class)
```

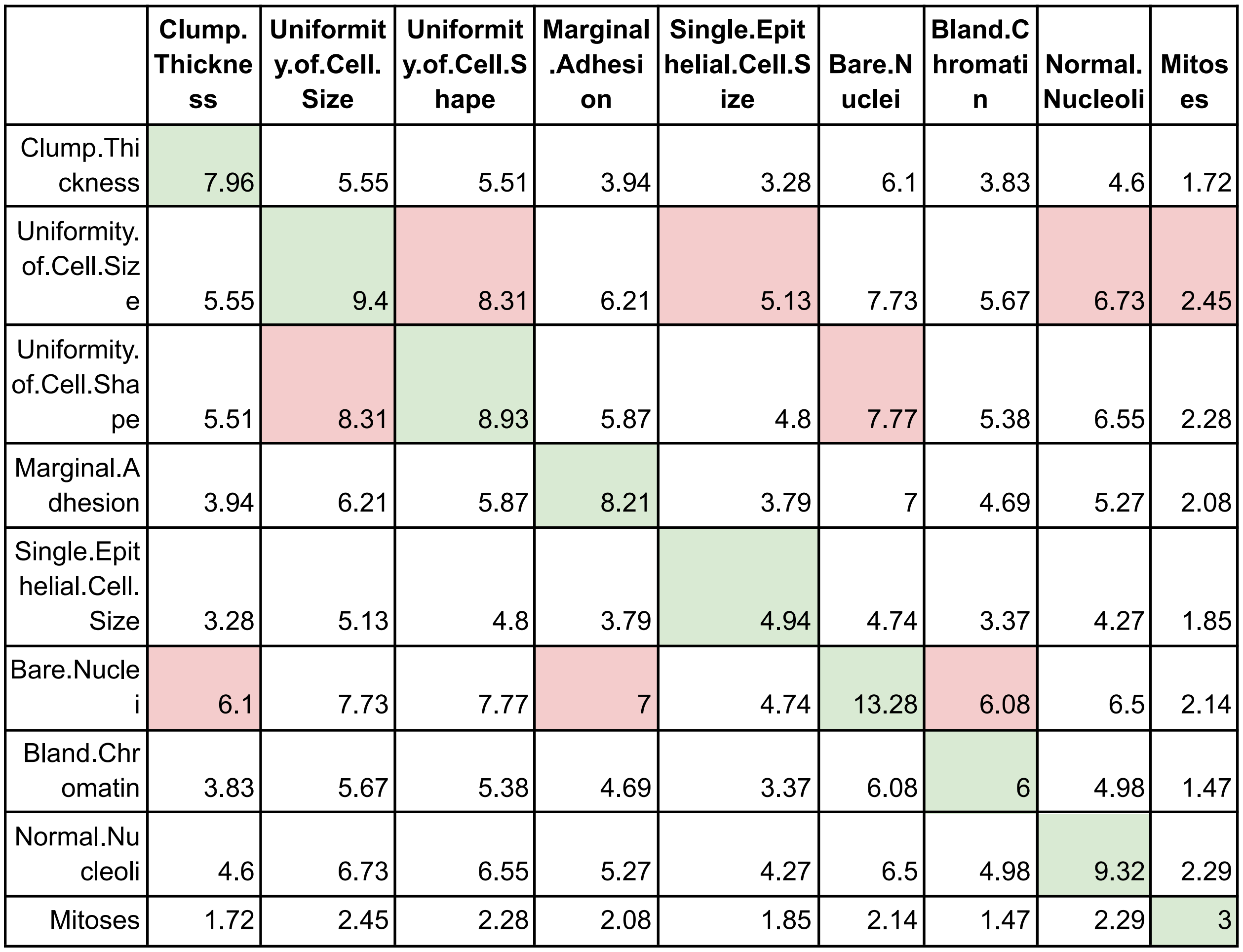
After the data is successfully imported we can create a rudimentary covariance table by implementing the built in R covariance function, **cov()** and pass it our data frame that holds the CSV data. Notably, we limit the columns placed within the Covariance matrix to rows one through nine because column 10 contains our response variable. This will automatically print out a crude text based table into the console. While this print out is suitable for quickly observing the table, it is not properly formatted for proper inference.

```
covMatrix <- cov(MyData[,1:9])
covMatrix
```

For better inference we utilized the xtable package installed earlier, allowing R to export any data frame or matrix by placing it within an HTML table. For this process we create a html head, an html table, and then place the html table within the html body - finally, we write the html file into the working directory.

```
html.head <- paste("<head>" ,
                '<link rel="stylesheet" type="text/css"
href="mystyle.css"/>',"</head>",sep='\n')

html.table <- paste(print(xtable(covMatrix),type='html','res.html'),
                collapse = "\n")

html.body <- paste("<body>", html.table,"</body>")

write(paste(html.head,html.body,sep='\n'),"CovMatrix.html")
```

Finally, we took the html table and placed it within a Google sheet file where we could easily highlight and manipulate the data (shown in the image below). The variance of all terms were highlighted in green, and the highest covariance between two different variables was highlighted with red within each column. From our inference, *Uniformity of Cell Size*, *Uniformity of Cell Shape, and Bare Nucleoli* tended to be the variables with the highest covariance in all the columns. Both of these variables should be considered important as they seem to have a strong relationship with all predictors.



	Clump.Thickness	**Uniformity.of.Cell.Size**	**Uniformity.of.Cell.Shape**	**Marginal.Adhesion**	**Single.Epithelial.Cell.Size**	**Bare.Nuclei**	**Bland.Chromatin**	**Normal.Nucleoli**	**Mitoses**
Clump.Thickness	7.96	5.55	5.51	3.94	3.28	6.1	3.83	4.6	1.72
Uniformity.of.Cell.Size	5.55	9.4	8.31	6.21	5.13	7.73	5.67	6.73	2.45
Uniformity.of.Cell.Shape	5.51	8.31	8.93	5.87	4.8	7.77	5.38	6.55	2.28
Marginal.Adhesion	3.94	6.21	5.87	8.21	3.79	7	4.69	5.27	2.08
Single.Epithelial.Cell.Size	3.28	5.13	4.8	3.79	4.94	4.74	3.37	4.27	1.85
Bare.Nuclei	6.1	7.73	7.77	7	4.74	13.28	6.08	6.5	2.14
Bland.Chromatin	3.83	5.67	5.38	4.69	3.37	6.08	6	4.98	1.47
Normal.Nucleoli	4.6	6.73	6.55	5.27	4.27	6.5	4.98	9.32	2.29
Mitoses	1.72	2.45	2.28	2.08	1.85	2.14	1.47	2.29	3

For easy viewability, click here to view this spreadsheet within the Google sheets interface.

Correlation Matrix
While it was important to understand the covariance between each variable, it would also be easier to interpret the data if it was presented in a manner that was bound between -1 and +1. The approach to constructing a Correlation Matrix is very similar to constructing a Covariance Matrix. Instead of using the cov() function, we employ the **cor()** function, along with the same html table exporting as before.

```
corMatrix <- cor(MyData[,1:9])
corMatrix

html.table <- paste(print(xtable(corMatrix),type='html','res.html'),
                collapse = "\n")

html.body <- paste("<body>", html.table,"</body>")

write(paste(html.head,html.body,sep='\n'),"CorMatrix.html")
```

At first glance we notice that all of our variables are positively correlated and as expected the predictors *Uniformity of Cell Shape* and *Uniformity of Cell Size* serve to have the highest correlation among all the predictor terms. It would be interesting to include these variables as interaction terms within the models and observe its effect on model accuracy.

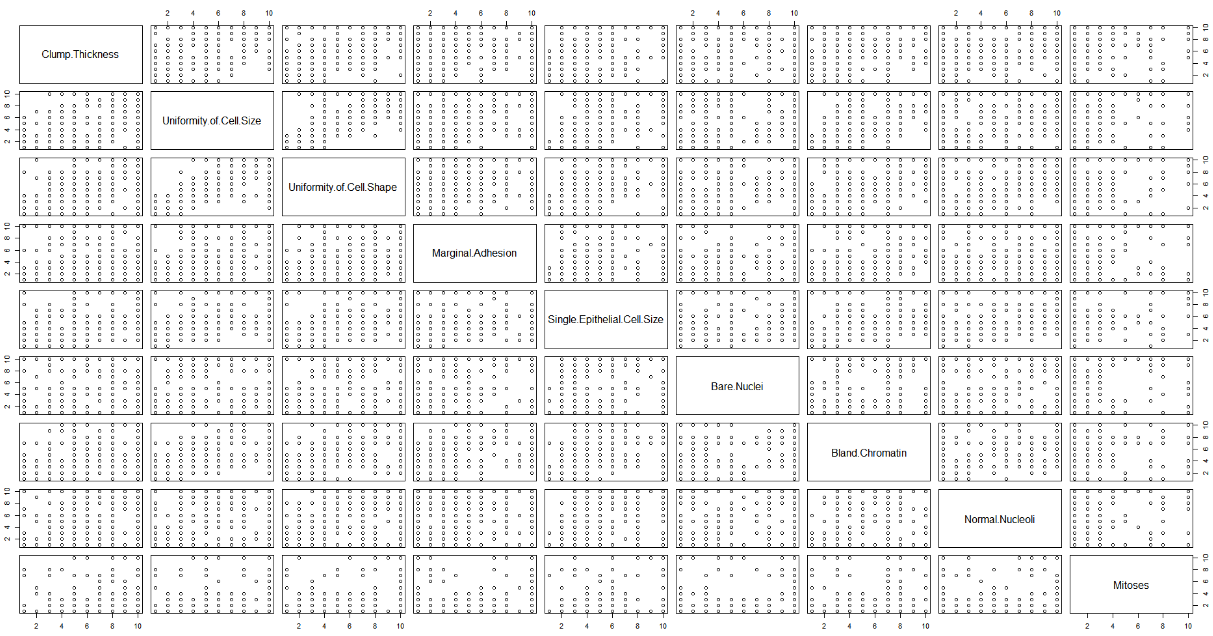
	Clump.Thickness	Uniformity.of.Cell.Size	Uniformity.of.Cell.Shape	Marginal.Adhesion	Single.Epithelial.Cell.Size	Bare.Nuclei	Bland.Chromatin	Normal.Nucleoli	Mitoses
Clump.Thickness	1	0.64	0.65	0.49	0.52	0.59	0.55	0.53	0.35
Uniformity.of.Cell.Size	0.64	1	0.91	0.71	0.75	0.69	0.76	0.72	0.46
Uniformity.of.Cell.Shape	0.65	0.91	1	0.69	0.72	0.71	0.74	0.72	0.44
Marginal.Adhesion	0.49	0.71	0.69	1	0.59	0.67	0.67	0.6	0.42
Single.Epithelial.Cell.Size	0.52	0.75	0.72	0.59	1	0.59	0.62	0.63	0.48
Bare.Nuclei	0.59	0.69	0.71	0.67	0.59	1	0.68	0.58	0.34
Bland.Chromatin	0.55	0.76	0.74	0.67	0.62	0.68	1	0.67	0.35
Normal.Nucleoli	0.53	0.72	0.72	0.6	0.63	0.58	0.67	1	0.43
Mitoses	0.35	0.46	0.44	0.42	0.48	0.34	0.35	0.43	1

For easy viewability, click here to view this spreadsheet within the Google sheets interface. *Note*: this spreadsheet is included in the same file as the Covariance spreadsheet, it is viewable as the second sheet, accessible in the lower left hand side of the Google Sheets interface.

Pairs Plot

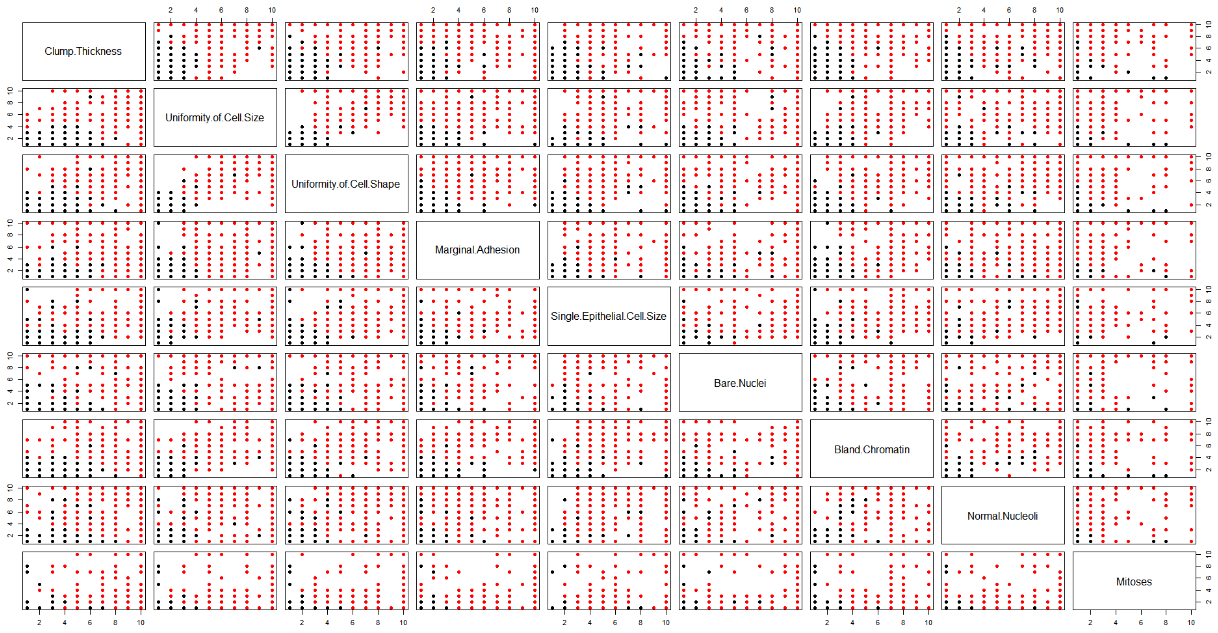
To view the interaction between two variables at a time, we implemented a simple pairs plot by using the built in R function **pairs()**, limiting our plot data to the first nine columns that contain our predictors and using the cex.labels modifier to increase the font of our labels. While this result was adequate, it was hard to decipher any information. We believe the reason our data is hard to interpret is because it only contains integers and no rational numbers, so the data is not visibility grouped making it hard to see any distinguishable grouping.

```
pairs(MyData[,1:9])
```



To slightly improve visibility, we implemented a second pairs plot that would highlight the Malignant cases in red. For this, we again implemented the built in R function pairs(), however this time we create a vector (group) that holds a 1 if the outcome is Benign, and 2 if the outcome is Malignant. These values will correspond to the index of the vector that will be applied to the color modifier (col) within the pairs function. We also employ the pch modifier to change the shape to a filled in circle for better visibility.

```
group <- NA
group[MyData$Class == "Benign"] <- 1
group[MyData$Class == "Malignant"] <- 2
pairs(MyData[,1:9], col = c("black", "red")[group], cex.labels=1.5,
pch = 16)
```



While this plot is still hard to infer from, we do note a “wedge” shape in the lower left hand side of most of the plots, indicating that as each predictor increases the odds of a tumor being malignant also increases. Most of the observations that were low in both predictors tend to be benign.

Model Construction and Analysis

Logistic Regression w/ All Predictor Variables
Considering our response variable is binary, the most suitable model seems to be Logistic Regression. Implementing this model is rather simple. First we start by setting a seed for the session by using the **set.seed()** function, this was to ensure that the team would be able to achieve reproducible results when running the script on separate machines. Next we split our dataset into two halves by using the caTools packages installed prior. To split the dataset we first define the dataset we are splitting and the ratio of the split between the training and testing sets by using the **sample.split()** function. Then we assign a half to the train subset, and a half to the test subset. Considering our data set was 683 observations, it was not divisible cleanly in half so 342 observations were given to the training set, and 341 were given to the test set. These two sets were always used across all the logistic regression models to ensure consistency.

```
set.seed(5)
```

```
split = sample.split(MyData$Class, SplitRatio = 0.5)
train = subset(MyData, split == TRUE)
test = subset(MyData, split == FALSE)
```

We then fitted training data to the logistic regression model by using the **glm()** function with the family modifier set to “binomial” to indicate our choice of logistic regression. After the model was successfully constructed, we used the **summary()** function to take a look at which variables were considered significant by the model and the magnitude each predictor had on the outcome.

```
glm.fit=glm(train$Class~.,data=train,family=binomial)
summary(glm.fit)
```

```
Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)                -11.82596  2.13908  -5.529 3.23e-08 ***
Clump.Thickness            0.68913    0.24516   2.811  0.00494 **
Uniformity.of.Cell.Size    -0.23345   0.40180  -0.581  0.56122
Uniformity.of.Cell.Shape   0.09322    0.39732   0.235  0.81451
Marginal.Adhesion          0.61828    0.21771   2.840  0.00451 **
Single.Epithelial.Cell.Size   0.53250 0.31572   1.687  0.09168 .
Bare.Nuclei                0.53886    0.19282   2.795  0.00520 **
Bland.Chromatin            0.67464    0.28945   2.331  0.01977 *
Normal.Nucleoli            0.27954    0.22518   1.241  0.21447
```

```
Mitoses                     0.04172    0.36117   0.116  0.90804
---
```

To our surprise only four of the variables (Clump.Thickness, Marginal.Adhesion, Bare.Nuclei, and Bland Chromatin) were deemed significant by the model. But we continued to test the model and arrive at a confusion matrix. First we used the **predict()** function to gather all the predictions for the test set. After this has completed we construct a new vector that will hold our categorical predictions - we use the **rep()** function to fill a vector (glm.pred) with all indices/observations with “Benign.” Next, we alter the glm.pred vector to swap in “Malignant” whenever the probability of that observation exceeds 0.5. Finally we construct the Confusion Matrix by comparing the results of the predictions against the actual values of the test subset by using the **table()** function. We can also gauge the accuracy of the model by dividing the total correctly categorized tumors by the total sum of all observations predicted - this gave us an accuracy of 96.5% which we consider excellent considering only 34.99% of our total observations contained Malignant observations. It is considerably better than randomly guessing or simply always predicting an outcome as Benign.

```
glm.probs = predict(glm.fit,test,type="response")

glm.pred = rep("Benign",341)
glm.pred[glm.probs>0.5] = "Malignant"

confmatrix <- table(glm.pred,test$Class)
confmatrix

glm.pred        Benign     Malignant
 Benign         212        2
 Malignant      10         117

(confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
0.9648094
```

Logistic Regression w/ Significant Variables
Considering the previous model deemed that the majority of the predictors were not significant we implemented a model with only those variables to see what effect it would have on the model's accuracy. So we created a second model, but this time only used *Clump Thickness*, *Marginal Adhesion*, *Bare Nuclei*, and *Bland Chromatin* as predictors.

```
glm.fit2=glm(train$Class~Clump.Thickness+Marginal.Adhesion+Bare.Nuclei
,data=train,family=binomial)
summary(glm.fit2)
```

```
Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     -11.3629   1.8339  -6.196 5.80e-10 ***
Clump.Thickness 0.8616     0.2081   4.141 3.46e-05 ***
Marginal.Adhesion   0.7325 0.1931   3.793 0.000149 ***
Bare.Nuclei     0.6558     0.1703   3.852 0.000117 ***
Bland.Chromatin 0.7317     0.2508   2.917 0.003530 **
---
```

Interestingly, all of these predictors are considerably small in p-value. This was a great result, but we wanted to continue with the model to understand how this would affect the accuracy. We employed the same code as the last logistic regression model to determine accuracy. Our model performed with 97.1% accuracy which was marginally better than the performance of the model with all the variables (96.5% accuracy). We expected a greater jump in accuracy, but were pleased with the result.

```
glm.probs = predict(glm.fit2,test,type="response")

glm.pred = rep("Benign",341)
glm.pred[glm.probs>0.5] = "Malignant"

confmatrix <- table(glm.pred,test$Class)
confmatrix
```

```
glm.pred        Benign     Malignant
  Benign        215        3
  Malignant     7          116
```

```
(confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
0.9706745
```

Logistic Regression w/ Different Seed Values

One of our concerns was varying p-values every time the model was constructed. Our hypothesis for why our p-values are so misleading is based on the small number of observations in the dataset. Considering we only have 683 total observations, splitting the data would lead to only 342 observations for testing, meaning there is less data to determine if the observations occurred by random chance. To test this we implemented a different seed value in the set.seed() function to see its effect on which predictors the model deemed significant. We again split the data using the caTools package and implemented a new logistic regression model to compare to our first. Here we noted ***one*** of our variables was deemed statistically significant, which seems to support our assumption that spitting the data has an effect on significance.

```
set.seed(1)

split = sample.split(MyData$Class, SplitRatio = 0.5)
train = subset(MyData, split == TRUE)
test = subset(MyData, split == FALSE)

glm.fit3=glm(train$Class~.,data=train,family=binomial)
summary(glm.fit3)
```

```
Coefficients:
                            Estimate Std. Error z value Pr(>|z|)
(Intercept)                 -9.00438   1.49432  -6.026 1.68e-09 ***
Clump.Thickness             0.41290    0.23622   1.748   0.0805 .
Uniformity.of.Cell.Size     -0.08181   0.34151  -0.240   0.8107
Uniformity.of.Cell.Shape    0.63798    0.41309   1.544   0.1225
Marginal.Adhesion           0.23076    0.15397   1.499   0.1339
Single.Epithelial.Cell.Size -0.18011   0.26177  -0.688   0.4914
Bare.Nuclei                 0.36391    0.15544   2.341   0.0192 *
Bland.Chromatin             0.45488    0.27200   1.672   0.0945 .
Normal.Nucleoli             0.30175    0.18066   1.670   0.0949 .
Mitoses                     0.55649    0.41068   1.355   0.1754
```

Nonetheless, we continued with the model to see if it would also have any drastic effect on this model's accuracy. Again, by constructing a confusion matrix similar to the last few models. Once again our model performed very well (97.7% accuracy) despite our p-values being very high. For future measures of performance (beyond this deliverable) we will not try to split our dataset as it leads us to misleading p-values that are difficult to predict. Instead we should rely on other methods such as K-Cross Fold Validation so we may make use of all our observations for model training.

```
glm.probs = predict(glm.fit3,test,type="response")

glm.pred = rep("Benign",341)
glm.pred[glm.probs>0.5] = "Malignant"

confmatrix <- table(glm.pred,test$Class)
confmatrix

glm.pred        Benign     Malignant
  Benign        216        2
  Malignant     6          117

(confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
0.9765396
```

Linear Regression w/ ALL Variables
For pedagogical purposes we constructed a Linear Regression model to see how we could improve Test MSE on a non-classification model. To start we constructed a Logistic Regression model using all of our observations, this way we can attain the predictions for every single observation.

```
set.seed(5)

glm.convert = glm(MyData$Class~.,data=MyData,family=binomial)
summary(glm.convert)

glm.con.probs=predict(glm.convert,type="response")
```

After attaining the predictions, we converted the probability estimates into log-odds by implementing the **log1p()** function. We then use these as the new response variable in our dataset. First we create a new dataset called “LRData” which is a copy of the original dataset, but with the response variable *Class* dropped from the dataframe. We then add our response variable as a column to the dataframe and call it *Outcome*, to create the new dataframe which will be used for all Linear Regression.

```
glm.con.log = log1p(glm.con.probs)

LRData = subset(MyData, select = -c(Class))
LRData$Outcome <- glm.con.log
```

Now we use the same methodology as used in Logistic Regression to split the data using the caTools package and sample.split() function. Again, similarly to the Logistic Regression model we fit our model to the training dataset; However, this time we use the **lm()** function to indicate that we are developing a Linear Regression model.

```
split.LR = sample.split(LRData$Outcome, SplitRatio = .5)
train.LR = subset(LRData, split == TRUE)
test.LR = subset(LRData, split == FALSE)

lm.fit=lm(train.LR$Outcome~.,data=train.LR)
summary(lm.fit)
```

```
Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 -0.162830   0.011006 -14.794  < 2e-16 ***
Clump.Thickness             0.020687   0.002545   8.129 8.69e-15 ***
Uniformity.of.Cell.Size     0.016836   0.004322   3.896 0.000118 ***
Uniformity.of.Cell.Shape    0.011871   0.004271   2.779 0.005757 **
Marginal.Adhesion           0.004500   0.002700   1.666 0.096600 .
Single.Epithelial.Cell.Size  0.007427   0.003571   2.080 0.038314 *
Bare.Nuclei                 0.028963   0.002265  12.787  < 2e-16 ***
Bland.Chromatin             0.013067   0.003624   3.605 0.000360 ***
Normal.Nucleoli             0.014028   0.002610   5.374 1.45e-07 ***
Mitoses                     -0.005092   0.003234  -1.575 0.116293
---
```

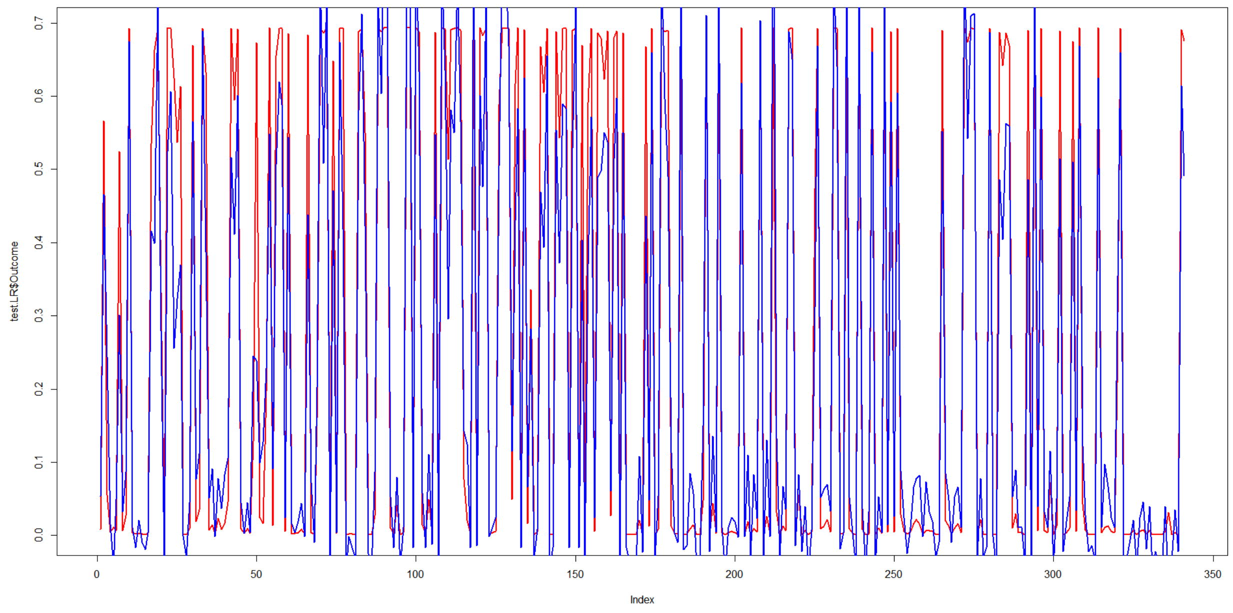
Linear Regression seems to claim more values are statistically significant, but as discussed in the previous sections, the p-values seem to vary based on the seed value. We quickly produce the Training Mean Square Error (MSE) by using the **mean()** function and taking the average of all the residuals of the model squared. This model has a Training MSE of 0.0079, which is close to zero. Ideally our Test MSE will also be close to this value.

```
mean(lm.fit$residuals^2)
0.007930733
```

Obviously, Training MSE is not the only metric we consider when comparing models. So for our next comparison we decided to create a visual graph using **plot() -** This line graph depicts the predicted values in blue, overlaid on the red line which depicted the actual values. To create the line colors we used the col modifier, and made the lines on the graph thicker using the lwd modifier. As the graph shows, the model does not accurately predict the actual values in the areas where the red line is poking through the graph.

```
lm.predict = predict(lm.fit, test.LR)
```

```
plot(test.LR$Outcome,type="l",lty=1.8,lwd=2,col="red")
lines(lm.predict,type="l",lwd=2,col="blue")
```



Finally, we compute the Test MSE by using the Metrics package. We simply use the **mse()** function included in the package. This model achieved a Test MSE of 0.0087 which was remarkably close to our Training MSE of 0.0079. Overall this model performed better than expected. The upcoming section displays our results in trying to improve this result.

```
mse(test.LR$Outcome, predict(lm.fit, test.LR))
0.008748418
```

Linear Regression w/ Non-Linear Transformations
Since we implemented reducing *p* to only significant variables with our Logistic Regression models, we wanted to explore how we could experiment with the Bias-Variance tradeoff by relaxing the linear model. We again implemented a Linear Regression model, but this time we included the predictors deemed significant by the last model as squared predictors, as well as standard predictors. All of the squared terms were considered significant based on p-value, except *Normal Nucleoli*^2. Interestingly, including squared predictors made *Marginal Adhesion* a significant predictor, as it was not in the previous model.

```
lm.fit2=lm(train.LR$Outcome~.+I(Clump.Thickness^2)+I(Uniformity.of.Cel
l.Size^2)+I(Uniformity.of.Cell.Shape^2)+I(Single.Epithelial.Cell.Size^
2)+I(Bare.Nuclei^2)+I(Bland.Chromatin^2)+
I(Normal.Nucleoli^2),data=train.LR)

summary(lm.fit2)
```

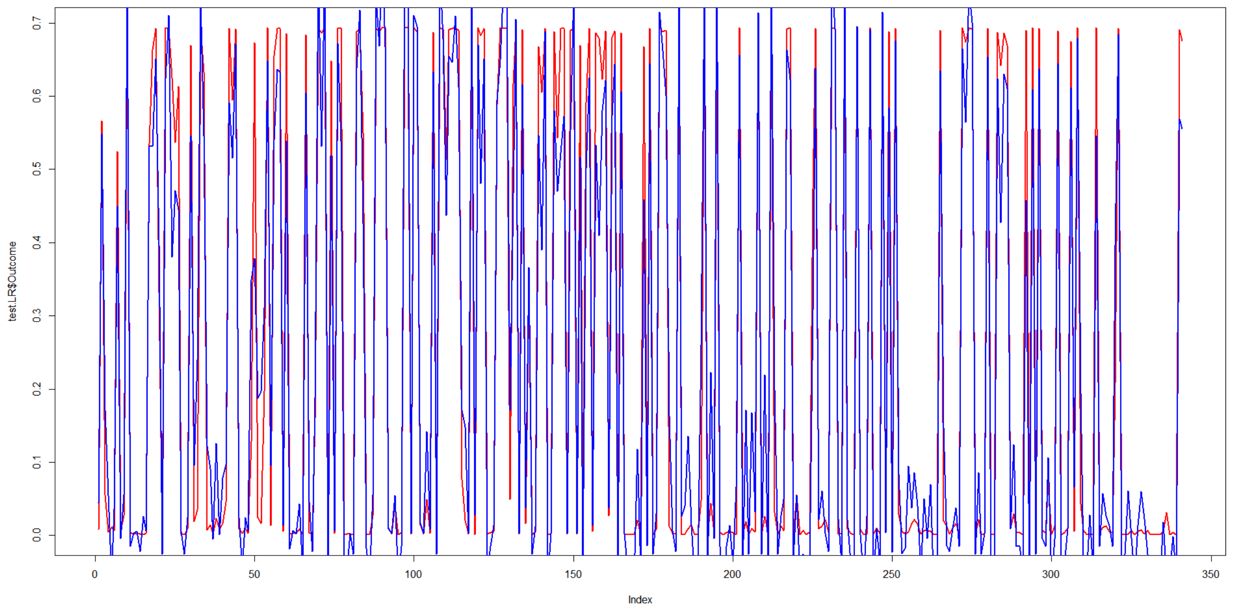
```
Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                      -3.080e-01  2.192e-02 -14.053  < 2e-16 ***
Clump.Thickness                  -1.553e-03  5.444e-03  -0.285 0.775592
Uniformity.of.Cell.Size           4.802e-02  1.084e-02   4.429 1.30e-05 ***
Uniformity.of.Cell.Shape          6.125e-02  9.782e-03   6.262 1.20e-09 ***
Marginal.Adhesion                 5.641e-03  2.205e-03   2.558 0.010969 *
Single.Epithelial.Cell.Size       2.522e-02  9.842e-03   2.563 0.010833 *
Bare.Nuclei                       3.621e-02  7.147e-03   5.067 6.81e-07 ***
Bland.Chromatin                   4.439e-02  7.459e-03   5.951 6.88e-09 ***
Normal.Nucleoli                   2.101e-02  7.948e-03   2.643 0.008620 **
Mitoses                          -8.226e-06  2.639e-03  -0.003 0.997515
I(Clump.Thickness^2)              1.709e-03  5.027e-04   3.399 0.000761 ***
I(Uniformity.of.Cell.Size^2)     -2.515e-03  8.679e-04  -2.897 0.004018 **
I(Uniformity.of.Cell.Shape^2)    -4.516e-03  8.238e-04  -5.481 8.48e-08 ***
I(Single.Epithelial.Cell.Size^2) -1.666e-03  8.293e-04  -2.009 0.045315 *
I(Bare.Nuclei^2)                 -1.597e-03  6.228e-04  -2.565 0.010770 *
I(Bland.Chromatin^2)             -3.160e-03  7.128e-04  -4.433 1.27e-05 ***
I(Normal.Nucleoli^2)             -1.170e-03  7.090e-04  -1.650 0.099884 .
---
```

We then calculated the Training MSE once again using the mean() function to find that it was now lower than the previous model. Ofcourse, this is expected as the function is now more variable and fits this specific sample better. However, when comparing the visual overlaid line graphs, there seems to be less red poking through, possibly indicating that this model also fairs better in predicting. This visual examination is confirmed when we run the Test MSE, which is also lower than it was in the previous model.

```
mean(lm.fit2$residuals^2)
0.004699561
```

```
lm.predict2 = predict(lm.fit, test.LR)
```

```
plot(test.LR$Outcome,type="l",lty=1.8,lwd=2,col="red")
lines(lm.predict2,type="l",lwd=2,col="blue")
```



```
mse(test.LR$Outcome, predict(lm.fit2, test.LR))
0.006040118
```

Finally, we use the **anova()** method to compare the original linear regression model to our new model with Non-Linear Transformations. We find that the F-statistic is 31.922, beyond the critical point of 1, and the p-value associated with it is definitely significant - meaning that our new model with Non-Linear Transformations is superior to the original model.

```
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1    332 2.7123
2    325 1.6073  7    1.1051 31.922 < 2.2e-16 ***
---
```

Potential Problems

Of all the problems discussed in this deliverable, the biggest one seems to be the size of our dataset. Splitting the dataset into halves seems to provide accurate models, but because the p-values change depending on which random observations are selected, tweaking the model seems to be difficult. Two solutions for this problem would be to simply change the ratio of training data to testing data, or to perform K-Cross Validation so we could utilize all of our data for training. It seems appropriate to test both methods and see which one provides better results. The second problem seems to be how this data might perform on other classifiers. Originally we were worried Logistic Regression might not perform well on our data, but the results of our prediction eased us of that issue. However, it is true that Logistic Regression data does not perform well on data that is well separated, indicating to us that our data is not well separated making it difficult to utilize for Linear Discriminant Analysis (LDA) and for K-Nearest Neighbors. Overall, we are very pleased with how Logistic Regression performed and will look to maximize our prediction with other validation techniques.

Deliverable 4

Model Assessment
We created various models increasing in polynomial order to test the accuracy of our model as it became more flexible. We tested these models using various validation methods - Using all Data for Training and Testing, Validation Set, Leave-One-Out Cross Validation, and 5-fold Cross Validation.

All Data for Training and Testing
We first start our code by importing all of our required prerequisites using the **install.packages()** and **library()** functions. After which we import our data from the CSV file using **read.csv()**, and ensure R reads our outcome variable as a Categorical one using **as.factor()**. Finally we set the seed for the session using **set.seed()** to get reproducible results from the script.

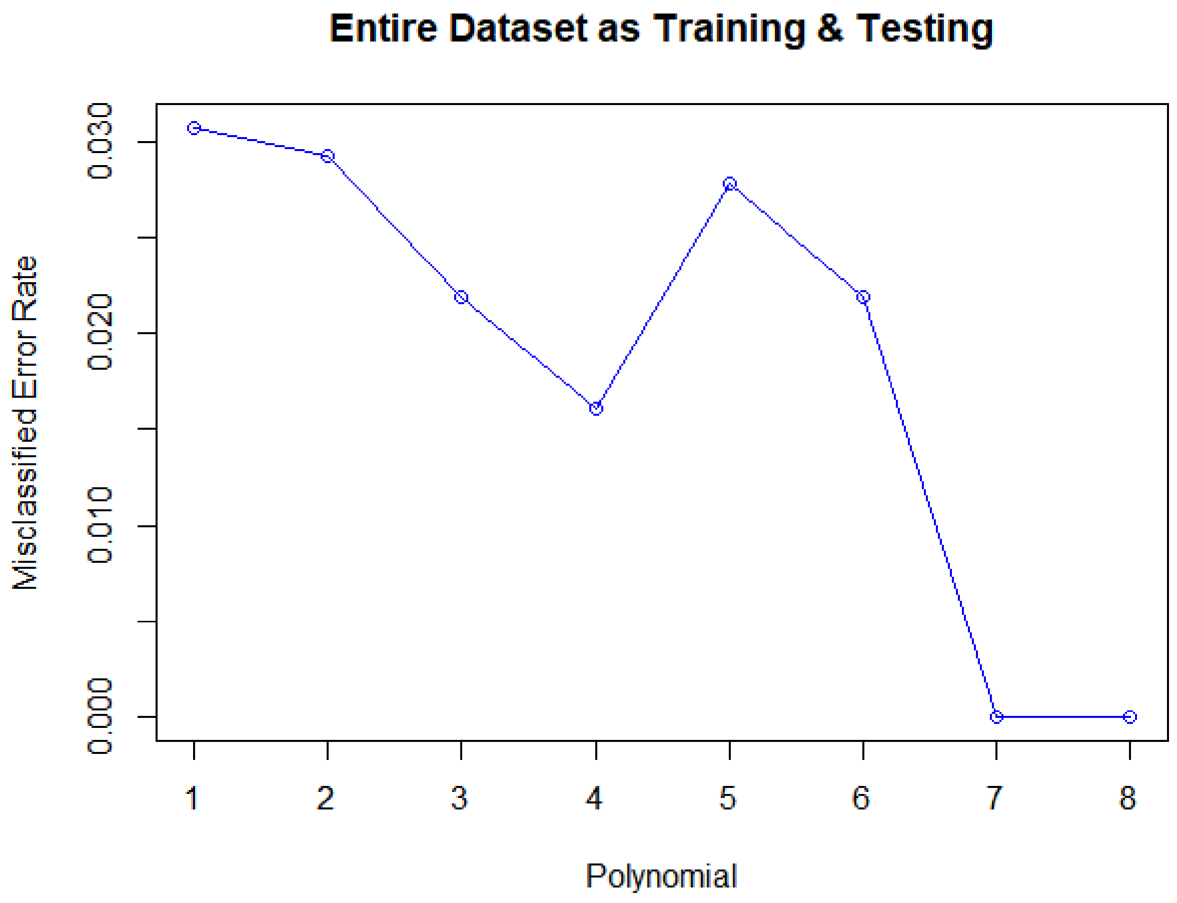
```
install.packages('caTools')
library(caTools)

install.packages('Metrics')
library(Metrics)

MyData <- read.csv("Breast Cancer Prediction_Clean.csv", header=T)
MyData$Class <- as.factor(MyData$Class)

set.seed(5)
```

We create an empty vector called **EntireData_Errors** to hold the error rates of our models using the **c()** function - this creates a null variable for us to fill later. We create a for-loop using the **for()** function and ask it to loop 8 times, each time incrementing the polynomial. Initially, the team tried to loop through 10 times as the assignment stated, however R gave us an error - `Error in ploy(Mitosis, i): 'degree' must be less than number of unique points.` Meaning that the Mitosis column did not have more than 9 unique values, and thus the flexibility of the model could not be pushed past a polynomial degree of 8, so long as Mitosis was included in the model. Considering our models are retaining all variables for this deliverable, our polynomial range was effectively limited to 1-8. This is the reason our for-loop only loops 8 times. Inside the for-loop we create a Logistic Regression model using the **glm()** function with the family modifier set to “binomial.” Since we want to increment the polynomial with every iteration, we use the **poly()** function when inserting our predictors. We pass the for-loop counter variable into poly() to represent the current polynomial. After each model is constructed we grab the probabilities for each observation using **predict()** and convert any probability over 0.5 to “Malignant.” Then, construct a confusion matrix and calculate our accuracy. Our Error Rate is then calculated by performing 1 - accuracy. This Error Rate is appended to the empty vector created earlier. This process is repeated until the for-loop steps through 8 models with increasing flexibility.

```
EntireData_Errors = c()
for(i in 1:8){

glm.fit=glm(Class~poly(Clump.Thickness,i)+poly(Uniformity.of.Cell.Size
,i)+poly(Uniformity.of.Cell.Shape,i)+poly(Marginal.Adhesion,i)+poly(Si
ngle.Epithelial.Cell.Size,i)+poly(Bare.Nuclei,i)+poly(Bland.Chromatin,
i)+poly(Normal.Nucleoli,i)+poly(Mitoses,i),data=MyData,family=binomial
)

  glm.probs = predict(glm.fit,MyData,type="response")

  glm.pred = rep("Benign",683) #There are 683 observations in the
MyData Set
  glm.pred[glm.probs>0.5] = "Malignant"

  confmatrix <- table(glm.pred,MyData$Class)

  accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

  ErrorRate = 1 - accuracy
  EntireData_Errors = append(EntireData_Errors, ErrorRate)
}
```

Finally, we can plot the Error rates fairly simply by using the **plot()** function. We pass the **EntireData_Errors** vector that holds the error rates for all of our models, set the type modified to "o" to indicate we want a line graph with points, and provide some labels. The output is a plot displaying the error rates of our models as flexibility increases using the same dataset for both training and testing.

```
plot(EntireData_Errors, type = "o", col = "blue", xlab ="Polynomial",
ylab ="Misclassified Error Rate", main = "Entire Dataset as Training &
Testing")
```



As expected, the model generally performs well as the polynomial increases. This is because we trained and tested with the same data, making the model very variable but only usable on this specific sample. The higher polynomial models would most likely perform differently on outside data because this validation method does not use any unseen data, meaning it underestimates the True Error Rate. Interestingly, we note that the Error Rate increased around polynomial 5 and 6 before dropping significantly, which is different from Training Error Rate patterns observed in the textbook.

All Data for Training and Testing Error Rates

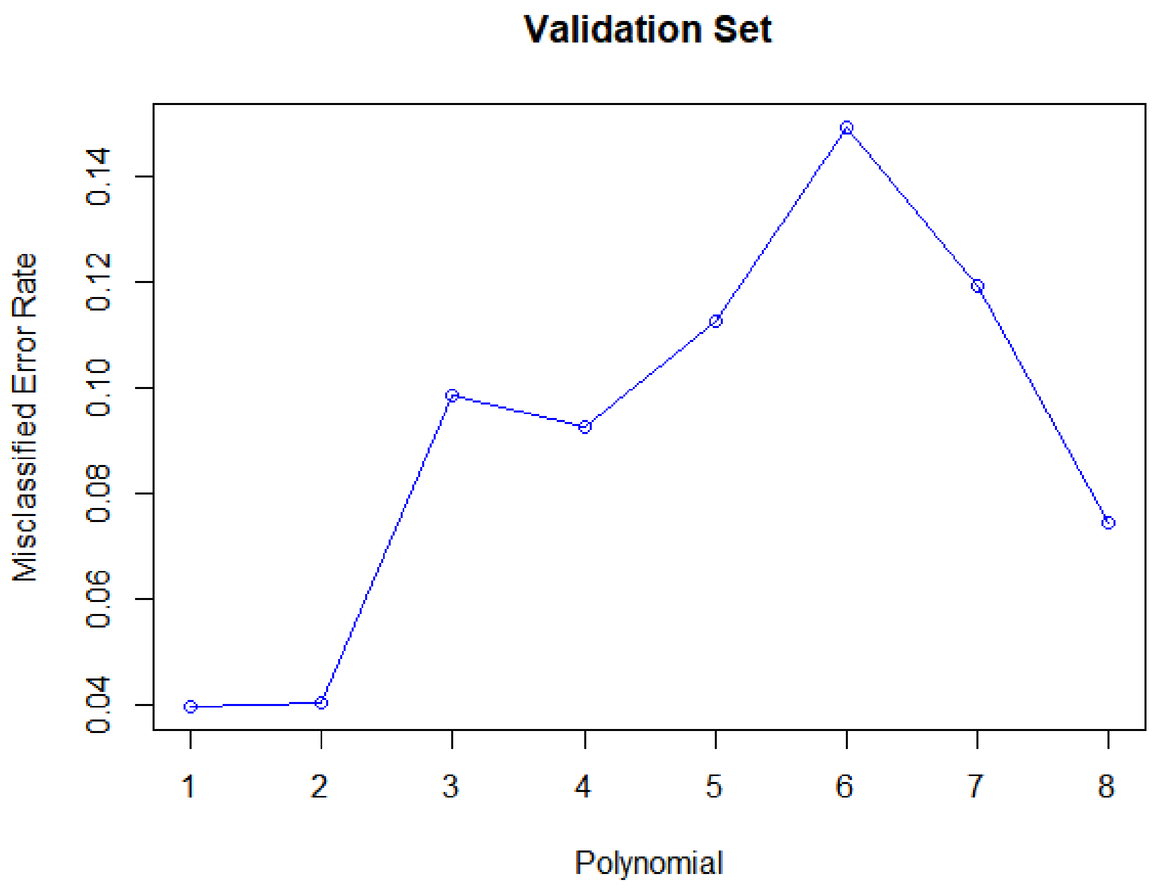
Polynomial	1	2	3	4	5	6	7	8
Error Rate	0.03074671	0.02928258	0.02196193	0.01610542	0.02781845	0.02196193	0.00000000	0.00000000

Validation Set

This methodology should provide more realistic results than the previous validation method as it will use data not seen by the model during the testing phase. Again, we start by creating an empty variable called **ValidSet_Errors** to hold all of our error rates. However, this time we split our dataset into two parts. We decided to use a ratio of 70% Training (478 observations) to 30% Testing (205 observations) for this test. We split the data by using the **sample.split()** function provided by the caTools library & package. We create two subsets of our data called **train** and **test**. Similar to the previous method we again create a for-loop using the same model structure, however this time we set the data equal to train. When we use the predict() function, this time we are interested in the estimated probabilities of the test subset. The probabilities are converted into labels and compared against the actual values. These comparisons are loaded into a confusion matrix, from which we calculate the Error Rate(s).

```
ValidSet_Errors = c()

split = sample.split(MyData$Class, SplitRatio = 0.8)
train = subset(MyData, split == TRUE)
test = subset(MyData, split == FALSE)

for(i in 1:8){

glm.fit=glm(train$Class~poly(Clump.Thickness,i)+poly(Uniformity.of.Cell.Size,i)+poly(Uniformity.of.Cell.Shape,i)+poly(Marginal.Adhesion,i)+poly(Single.Epithelial.Cell.Size,i)+poly(Bare.Nuclei,i)+poly(Bland.Chromatin,i)+poly(Normal.Nucleoli,i)+poly(Mitoses,i),data=train,family=binomial)

  glm.probs = predict(glm.fit,test,type="response")

  glm.pred = rep("Benign",137) #There are 137 observations in the Test Subset
  glm.pred[glm.probs>0.5] = "Malignant"

  confmatrix <- table(glm.pred,test$Class)
  accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

  ErrorRate = 1 - accuracy
  ValidSet_Errors = append(ValidSet_Errors, ErrorRate)
}
```

After this has been calculated we construct a plot using the same **plot()** function, this time using the **ValidSet_Errors** vector that stores the Error Rates from this validation method. The output is a plot displaying the error rates of our models as flexibility increases using the Validation Set method.



The Validation Set method has resulted in dramatically different results, here increasing the Polynomial degree makes the model perform worse, notably peaking at degree 6. However, the most important information to infer, is that making the model more flexible resulted in a higher error rate than before. Meaning the linear model is the best when all variables are included. In our previous deliverable we increased the accuracy by only squaring the *significant* terms, but it seems performance will drop if we square *all* terms.

Validation Set Error Rates

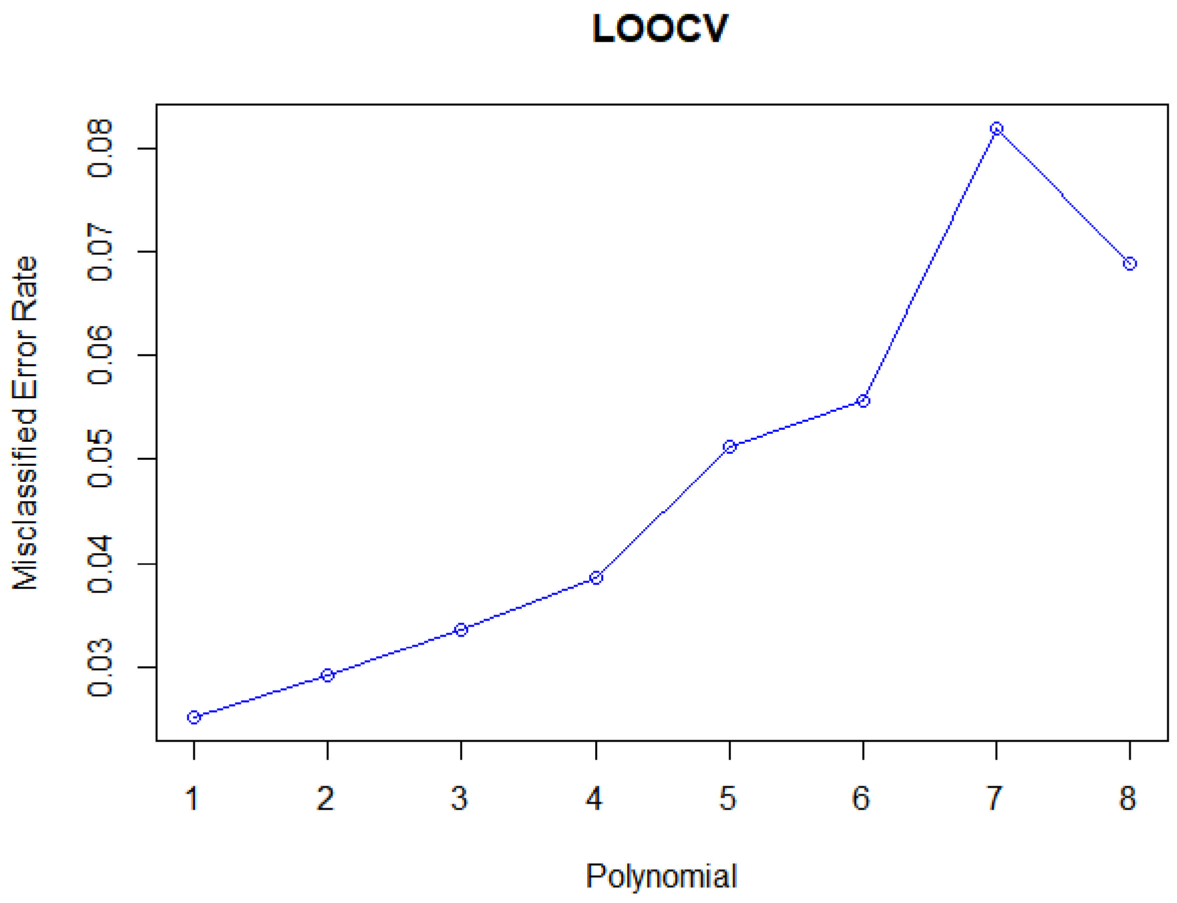
Polynomial	1	2	3	4	5	6	7	8
Error Rate	0.03973510	0.04026846	0.09868421	0.09271523	0.11258278	0.14935065	0.11920530	0.07432432

Leave One Out Cross Validation

We expect to see more realistic Error Rates with Leave One Out Cross Validation (LOOCV) as we can use the entire dataset to train the model, and validate using one observation at a time. To utilize LOOCV, we use **library()** function to import the "boot" library, which gives us access to the **cv.glm()** function needed to run this validation method. We again create an empty vector called **Loocv_Errors** to hold the error rates of our models and create a for-loop. This time we are allowed to use the entire dataset for training the model, so we set data equal to **MyData** (dataframe that holds all data). Following model creation in each iteration, we use the **cv.glm()** function, passing it our data and model, to acquire our Error Rate. Each iteration, the error rate is added to the **Loocv_Errors** vector.

```
library(boot)
Loocv_errors=c()

for(i in 1:8){

glm.fit=glm(Class~poly(Clump.Thickness,i)+poly(Uniformity.of.Cell.Size,i)+poly(Uniformity.of.Cell.Shape,i)+poly(Marginal.Adhesion,i)+poly(Single.Epithelial.Cell.Size,i)+poly(Bare.Nuclei,i)+poly(Bland.Chromatin,i)+poly(Normal.Nucleoli,i)+poly(Mitoses,i),data=MyData,family=binomial)

  Loocv_errors[i]=cv.glm(MyData,glm.fit)$delta[1]
}
```

After the for-loop is completed and our vector has been filled, we call on the **plot()** function to graph our Error Rates.



Here we noted that the Error Rates were significantly lower than before, during the Validation Set Test method, meaning that perhaps it was overestimated before and the actual Error Rate is lower. We assume this difference in Error Rate is because Validation Test tends to overestimate the Error Rate compared to LOOCV. Nonetheless, since our dataset is relatively small (*n* < 1000) we would prefer validation models that did not split our dataset into two parts, allowing us to maximize the number of observations used in training. We also observed again that any flexibility of the model seems to result in a higher Error Rate, indicating that our ideal model would not benefit from increasing flexibility for *all* variables.

LOOCV Error Rates

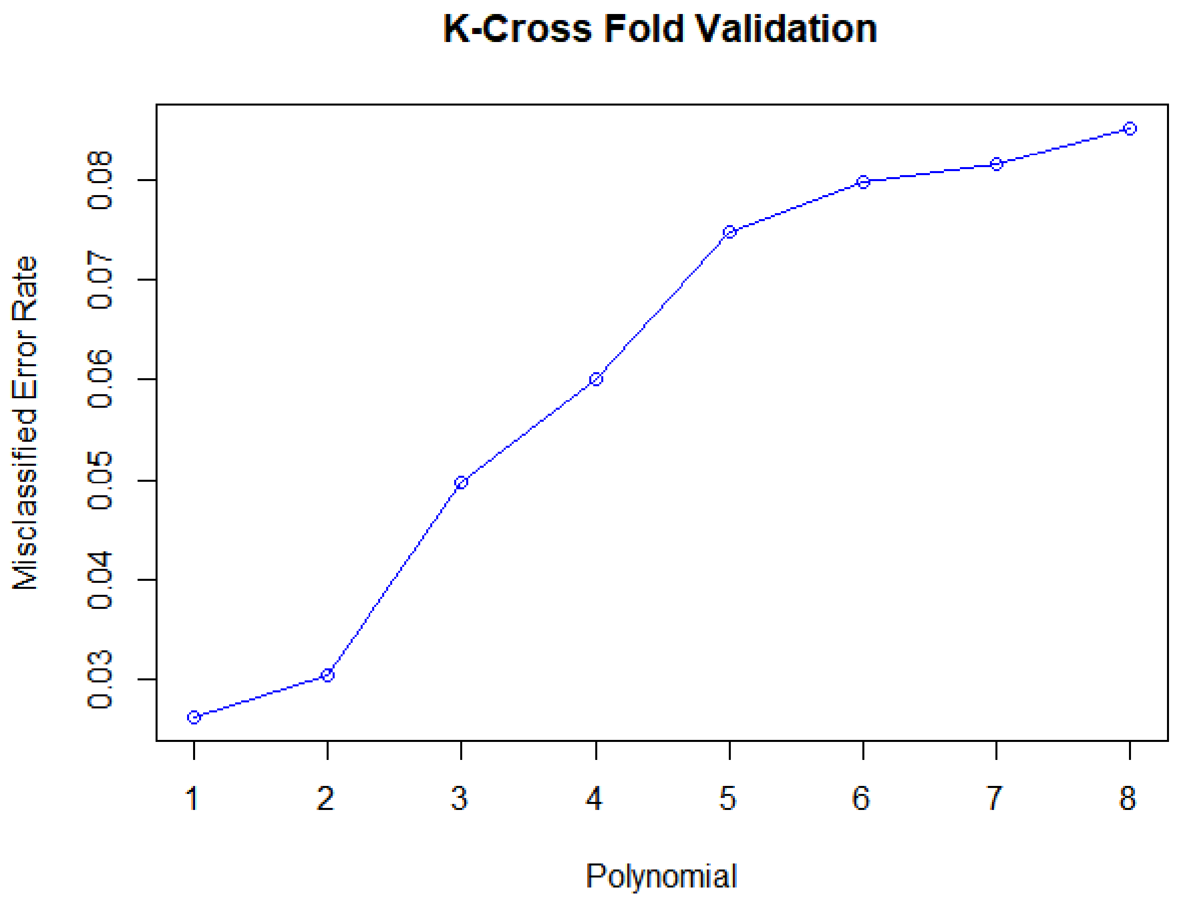
Polynomial	1	2	3	4	5	6	7	8
Error Rate	0.02516484	0.02913135	0.03356163	0.03853056	0.05124451	0.05563690	0.08198015	0.06887952

K-Cross Fold Validation
Considering LOOCV and K-Cross Fold Validation are very similar methods of validation, the script between them does not differentiate too much. The only difference is adding a modifier to include a *K* value that we wish to use in the **cv.glm()** function. Just as before, we create an empty variable called **KCrossValid_Errors** to hold our Error Rates.

```
KCrossValid_Errors = c()

for(i in 1:8){

glm.fit=glm(Class~poly(Clump.Thickness,i)+poly(Uniformity.of.Cell.Size
,i)+poly(Uniformity.of.Cell.Shape,i)+poly(Marginal.Adhesion,i)+poly(Si
ngle.Epithelial.Cell.Size,i)+poly(Bare.Nuclei,i)+poly(Bland.Chromatin,
i)+poly(Normal.Nucleoli,i)+poly(Mitoses,i),data=MyData,family=binomial
)
  KCrossValid_Errors[i]=cv.glm(MyData,glm.fit,K=5)$delta[1]
}
```

The **plot()** function is used again to create a graph of the Error Rates.

```
plot(KCrossValid_Errors, type = "o", col = "blue", xlab ="Polynomial",
ylab ="Misclassified Error Rate", main = "K-Cross Fold Validation")
```



K-Cross Fold Validation seems to have a different overall shape in comparison to LOOCV, in that the increase in Error Rate is much sharper as the polynomial increase, and the 8th degree is now relatively higher in Error Rate when it was not before. Both LOOCV and the Validation Set plots show a decreasing Error Rate at the 8th degree, but K-Cross Fold Validation does not reflect this shape. Since K-Cross Fold Validation does have a degree of randomness in how the observations are sampled, the team ran the same script multiple times on the same seed, to find some plots that did not conform to the general shape seen in LOOCV and Validation Set plots. We believe this is due to the lack of variability in some variables. However, we again note that adding flexibility to *all* predictors did not result in a decrease of the Error Rate, but in fact caused it to increase over all the models generated using K-Cross Fold Validation.

Bootstrap
We start by creating a function, called **boot.fn()** that will return the coefficient estimates of a dataset using Logistic Regression, given a dataset and indices.

```
boot.fn=function(data,index){

return(coef(glm(MyData$Class~.,data=MyData,subset=index,family=binomia
l)))
}
```

Here we test the function to randomly sample the observations and create a bootstrap estimate for each one of our variable coefficients using our newly created **boot.fn()** function.

```
boot.fn(MyData,1:683)
```

```
(Intercept)          Clump.Thickness      Uniformity.of.Cell.Size
-10.103942243        0.535014068          -0.006279717

Uniformity.of.Cell.Shape        Marginal.Adhesion
0.322706496                     0.330636915

Single.Epithelial.Cell.Size     Bare.Nuclei          Bland.Chromatin
0.096635417                     0.383024572          0.447187920

Normal.Nucleoli      Mitoses
0.213030682          0.534835631
```

Next, we want to resample our dataset and determine the standard error of bootstrap coefficient estimates to compare against the standard error estimates made by the standard **glm()** logistic regression model**.** To do this we use the **boot()** function, passing it our dataset, the function used to estimate our desired parameter (the coefficients in our case), and the number of times we wish to resample the dataset (1,000 times).

```
boot(MyData,boot.fn,1000)

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = MyData, statistic = boot.fn, R = 1000)

Bootstrap Statistics :
           original   bias       std. error
t1*  -10.103942243 -1.07637810   2.0334067
t2*   0.535014068  0.07406231    0.2043794
t3*    -0.006279717  0.02403870   0.2781853
t4*   0.322706496  0.03463111    0.2345281
t5*   0.330636915  0.02921998    0.1871320
t6*   0.096635417 -0.03980964    0.2118373
t7*   0.383024572  0.06468686    0.1638490
t8*   0.447187920  0.03916607    0.2338407
t9*   0.213030682  0.03686898    0.1622063
t10*   0.534835631  0.08210735   0.3552904
```

Finally, we compare the Bootstrap estimates against our standard logistic regression model, considering it had the lowest Error Rate throughout most of our validation methods. We simply use the **glm()** function and use all of our variables to generate a logistic regression model and then view the model's summary by using **summary()**, which contains the standard error estimates.

```
glm.fit = glm(MyData$Class~.,data=MyData,family=binomial)
summary(glm.fit)
```

```
Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)                -10.10394  1.17488  -8.600  < 2e-16 ***
Clump.Thickness            0.53501    0.14202   3.767 0.000165 ***
Uniformity.of.Cell.Size    -0.00628   0.20908  -0.030 0.976039
Uniformity.of.Cell.Shape   0.32271    0.23060   1.399 0.161688
Marginal.Adhesion          0.33064    0.12345   2.678 0.007400 **
Single.Epithelial.Cell.Size   0.09663 0.15659   0.617 0.537159
Bare.Nuclei                0.38303    0.09384   4.082 4.47e-05 ***
Bland.Chromatin            0.44719    0.17138   2.609 0.009073 **
Normal.Nucleoli            0.21303    0.11287   1.887 0.059115 .
Mitoses                    0.53484    0.32877   1.627 0.103788
```

For convenience, we have created a table that makes it easy to compare the bootstrap estimates of the standard error against the standard error estimated by the logistic regression model.

Coefficient Standard Error Estimates

Variable	Bootstrap Standard Error Estimate	Logistic Regression Standard Error Estimate
Intercept	2.0334067	1.17488
Clump.Thickness	0.2043794	0.14202
Uniformity.of.Cell.Size	0.2781853	0.20908
Uniformity.of.Cell.Shape	0.2345281	0.23060
Marginal.Adhesion	0.1871320	0.12345
Single.Epithelial.Cell.Size	0.2118373	0.15659
Bare.Nuclei	0.1638490	0.09384
Bland.Chromatin	0.2338407	0.17138
Normal.Nucleoli	0.1622063	0.11287
Mitoses	0.3552904	0.32877

The standard errors are higher on the Bootstrap expected, as it does not make the same assumptions as Logistic Regression. However, the standard errors could be closer. It would be beneficial to test the version of the model constructed in the previous deliverable with only the significant terms squared that seemed to improve accuracy against the bootstrapped estimates. Overall, the standard errors are somewhat close, possibly indicating that the model fit is adequate but could be improved

Deliverable 5

Model Selection
For this section of the deliverable the team tested Forward Selection, Backward Selection, Ridge Regression, Lasso and compared them to find the best model based on the results of each model's 5-fold cross validation error rates.

Forward Selection
Before we begin performing model selection we import the various libraries utilized throughout this deliverable by using the **library()** function. As usual, we also import our dataset from our working directory using the **read.csv()** function and ensure R reads our outcome variable as a categorical variable using the **as.factor()** function. Finally, we set the seed for the session using the **set.seed()** function to receive reproducible results.

```
library(leaps)
library(boot)
library(caret)
library(mlbench)
library(splines)
library(gam)

MyData <- read.csv("Breast Cancer Prediction_Clean.csv", header=T)

MyData$Class <- as.factor(MyData$Class)

set.seed(5)
```

Now we can begin performing forward selection by using the **regsubsets()** function which automatically performs best subset selection on a given model and its predictors. However, since we are interested in Forward Selection specifically, we set the method modifier to "forward" to indicate our preference. The default number of variables the regsubsets() function considers is 8, but since we have 9 total predictors and may want to compare the performance of the entire model against fewer predictor models within the regsubsets object, we set the nvmax modifier to 9 to ensure a model will be created with all models. After running the Forward Selection, we can use the **summary()** function to view the best model for each size.

```
regfit.fwd = regsubsets(Class~.,data=MyData,nvmax=9,method="forward")
summary(regfit.fwd)
```

The output of the summary is formatted in a manner that is hard to fit in a document, so we have formatted it into a visual table. The original text output can be viewed here.

Forward Selection

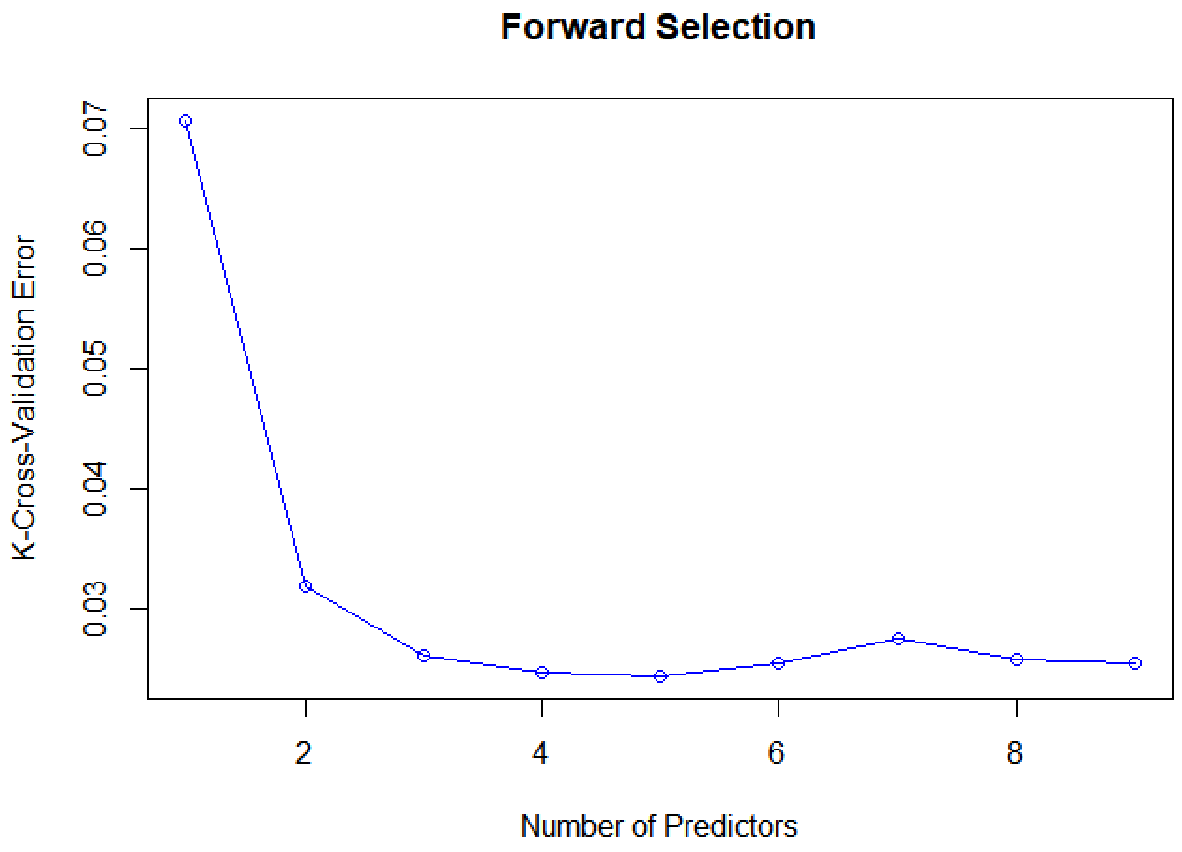
	Clump.Thickness	Uniformity.of.Cell.Size	Uniformity.of.Cell.Shape	Marginal.Adhesion	Single.Epithelial.Cell.Size	Bare.Nuclei	Bland.Chromatin	Normal.Nucleoli	Mitoses
1						x			
2		x				x			
3	x	x				x			
4	x	x				x		x	
5	x	x				x	x	x	
6	x	x	x			x	x	x	
7	x	x	x	x		x	x	x	
8	x	x	x	x	x	x	x	x	
9	x	x	x	x	x	x	x	x	x

Now we can compare each model within Forward selection by creating 9 different models, each one with the predictors listed above. After constructing each model, we perform 5-fold cross validation and retrieve and plot the error rate to observe which model performed the best. As an example, we provide the code for a 6 variable model below - which consists of the predictors: Clump.Thickness, Uniformity.of.Cell.Size, Uniformity.of.Cell.Shape, Bare.Nuclei, Bland.Chromatin, and Normal.Nucleoli. First we create a vector to hold the error rates from all our models, called Error.rates, which is filled with 0 nine times using the **rep()** function. Next we use the **glm()** function with the family modifier set to “binomial” to indicate we would like to perform logistic regression. After the model is constructed we can perform k-cross fold validation using the **cv.glm()** function, setting the k value to 5 since we want to perform 5-fold cross validation. Finally, we put the error rate of the 6 variable model in the 6th index of the Error.rates vector. *This process is performed for every one of our nine models.*

```
Error.rates=rep(0,9)

glm.fit=glm(Class~Clump.Thickness+Uniformity.of.Cell.Size+Uniformity.o
f.Cell.Shape+Bare.Nuclei+Bland.Chromatin+Normal.Nucleoli,data=MyData,f
amily=binomial)

error.rate=cv.glm(MyData,glm.fit,K=5)$delta[1]
Error.rates[6]=error.rate
```

To plot the Error Rates from all of our models, we employ the **plot()** function, passing it our Error.rates vector. We add our own labels and change the plot type to “o” to indicate we want to see both points and lines overlapped on our plot.

```
plot(Error.rates, type = "o", col = "blue", xlab ="Number of
Predictors", ylab ="K-Cross-Validation Error", main="Forward
Selection")
```



For comparison purposes we also provide a numerical output of the Error rates. Here we observe that the model with 5 predictors outperformed all other models. However, we also note that it was only marginally better than the model that used 4 predictors. Given that the results are so close, we would rather choose the simpler 4 predictor model as the ideal model from Forward Selection.

Number of Predictors	Error Rate
1	0.07055592
2	0.03190405
3	0.02612898
4	0.02461881
5	0.02440758
6	0.02540416
7	0.02743437
8	0.02581996
9	0.02550862

Backward Selection
The code for backward selection is very similar to the one used for Forward Selection. The only difference is instead of specifying “forward” for the method modifier in the **regsubsets()** function, we specify “backward.” Again we can use the summary() function to see which variables are selected from backward selection.

```
regfit.bwd = regsubsets(Class~.,data=MyData,nvmax=9,method="backward")
summary(regfit.bwd)
```

Similar to forward selection, because the output is hard to format into a text document we have created a visual to display the result. The original text output can be viewed here. We note that in our case, the models that result at every size are identical. However, considering we are using k-cross fold validation to understand predictive performance, there is an aspect of randomness because of how the observations are resampled for comparison. We wanted to see how drastic our results would change if we performed validation again.

Backward Selection

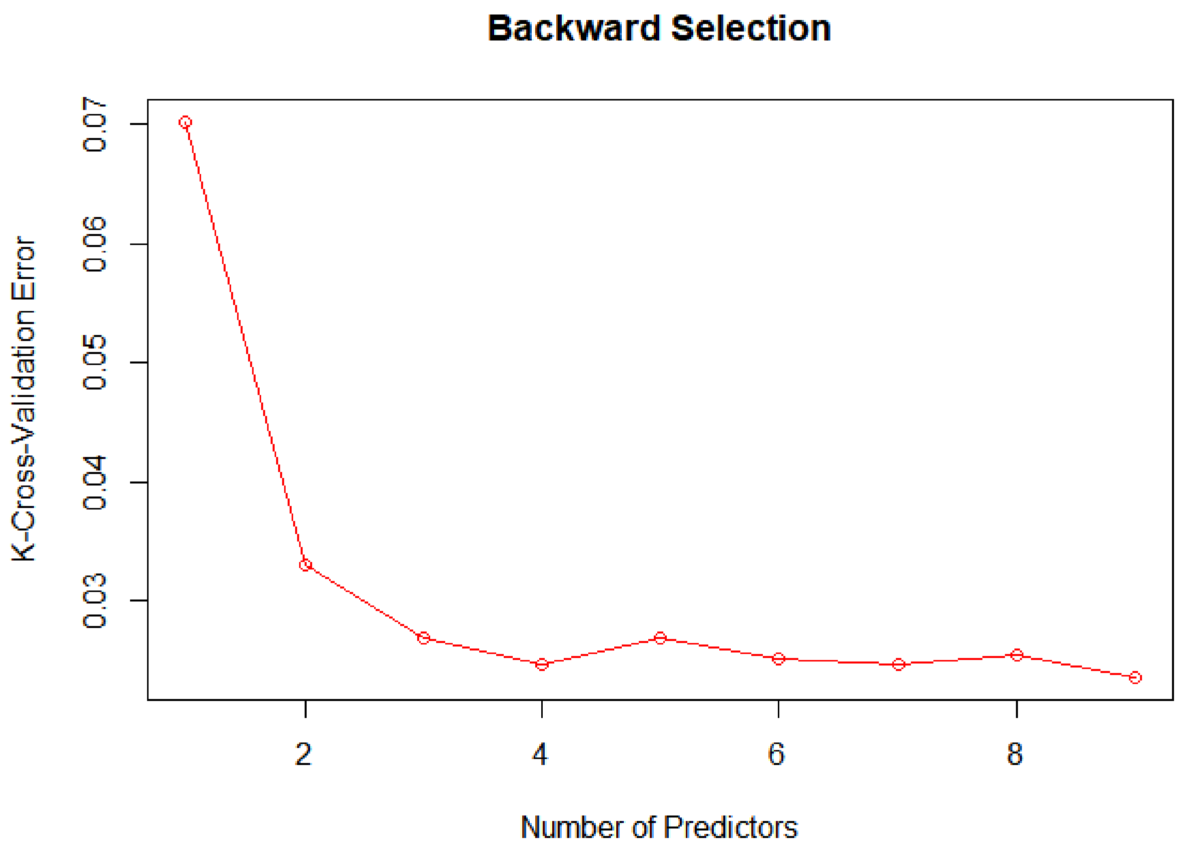
	Clump.Thickness	Uniformity.of.Cell.Size	Uniformity.of.Cell.Shape	Marginal.Adhesion	Single.Epithelial.Cell.Size	Bare.Nuclei	Bland.Chromatin	Normal.Nucleoli	Mitoses
1						x			
2		x				x			
3	x	x				x			
4	x	x				x		x	
5	x	x				x	x	x	
6	x	x	x			x	x	x	
7	x	x	x	x		x	x	x	
8	x	x	x	x	x	x	x	x	
9	x	x	x	x	x	x	x	x	x

We performed validation the exact same way as before. We create a vector called Error.rates2 that will hold the Error rates of our models during the second run of validation. Again, we provide an example of how a single model was constructed. The code below displays how our 4 variable model was constructed and how the error rate is stored in the Error.rates2 vector.

```
glm.fit=glm(Class~Bare.Nuclei+Uniformity.of.Cell.Size+Clump.Thickness+
Normal.Nucleoli,data=MyData,family=binomial)

error.rate=cv.glm(MyData,glm.fit,K=5)$delta[1]
Error.rates2[4]=error.rate
```

We employ the **plot()** function to create a line graph of our Backward Error rates.

```
plot(Error.rates2, type = "o", col = "red", xlab ="Number of
Predictors", ylab ="K-Cross-Validation Error", main="Backward
Selection")
```

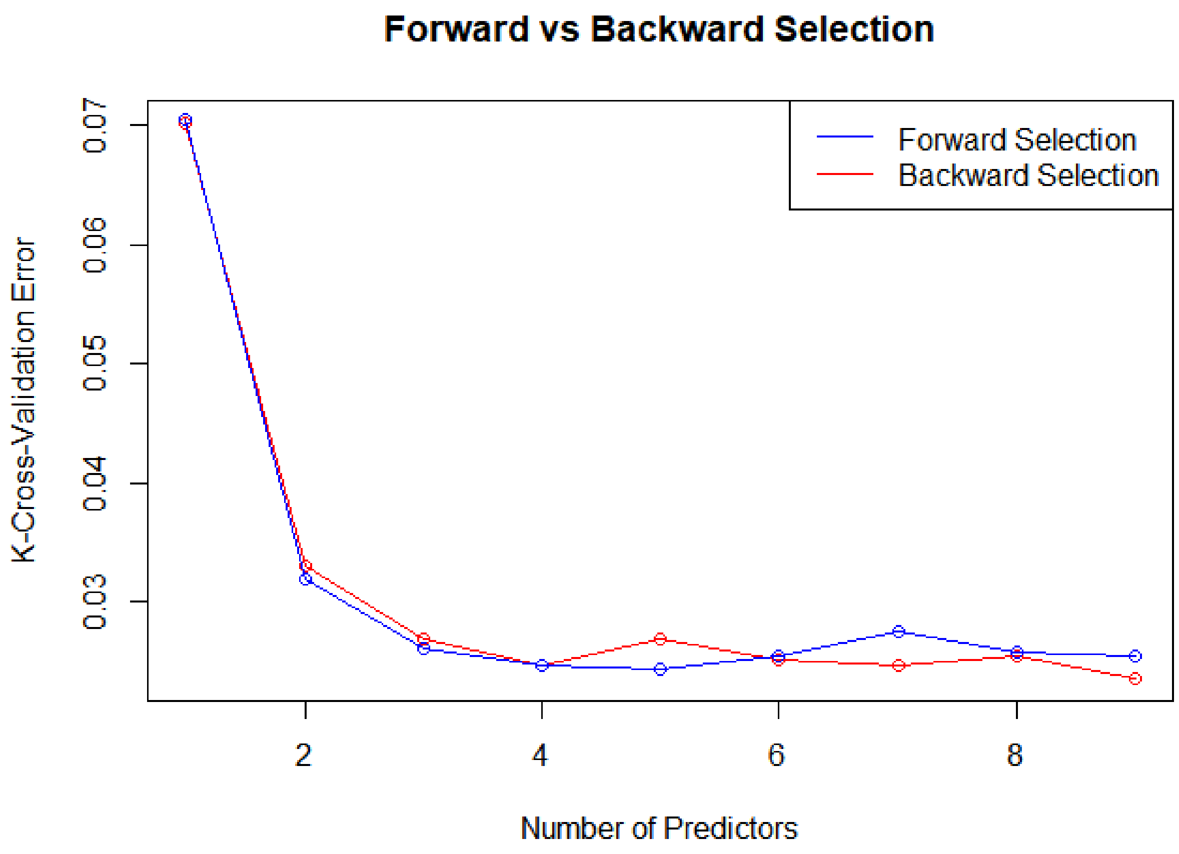


For this run we observed that the 9 variable model outperformed all other models, including the ones from the initial run of the models. We assume this is because of the randomness introduced into validation by k-cross fold validation. However, we once again note that the Error rate drops sharply after including 3 variables into the model and “steadies” out once the model has greater than or equal to 4 variables. Thus once again we choose the 4 variable model from the backward selection models (which were identical to the forward selection models) because it is the simplest model for which we achieve a significantly lower error rate relative to all other models.

Number of Predictors	Error Rate
1	0.07018557
2	0.03304954
3	0.02694649
4	0.02463429
5	0.02686422
6	0.02508366
7	0.02467253
8	0.02547250
9	0.02361484

For comparison's sake, we also construct a plot that displays the error rates of both backward and forward selection overlaid on top of each other. We create this by using the plot() function in an identical manner to how the Backward Selection plot was constructed. We then use the **lines()** function to superimpose the Forward Selection error rates overtop the plotted Backward Selection error rates. For better interpretation we also add a legend by using the **legend()** function which simply serves to identify each line. In both cases, we again note that the error rate seems to "steady" around the *4 variable model*, which would seem to indicate that it is the ideal model to use based on the results of Forward & Backward Selection.

```
plot(Error.rates2, type = "o", col = "red", xlab ="Number of
Predictors", ylab ="K-Cross-Validation Error", main="Forward vs
Backward Selection")

lines(Error.rates, type = "o", col = "blue")

legend("topright",legend=c("Forward Selection", "Backward Selection"),
col=c("blue","red"), lty = 1:1)
```



Ridge Regression

For this portion, and the Lasso portion we utilized the caret package as it was easier to employ regularization with k-cross fold validation. Caret was already imported using the **library()** function at the beginning of this deliverable. We start by defining a train control object using the **trainControl()** function. This essentially allows us to specify the methodology that will be used to determine which model performs the best from various lambda values. We set the method modifier to “cv” as we want to perform cross validation, and the number modifier to 5 as we specifically want to perform 5-fold cross validation as we did with Forward & Backward Selection.

```
fitController = trainControl(method = "cv", number = 5,
savePredictions = T)
```

After the trainControl object has been defined, we can define our Ridge Regression model by using the **train()** function. This function is not the generic train function found in R, it is the one used in the caret package. Here we define our outcome variable and set the method modifier to “glmnet.” We can also pass in the trainControl object we created before so the model utilizes 5-fold cross validation. The tuneGrid parameter is where we set alpha to 0 to indicate we want to perform Ridge Regression, and define the various values we wish to test for lambda. We want to test for a fairly wide range of lambda so we pass in `10^seq(-10,10, length=100)` or 100 sequential values ranging from 1e-10 (10^-10) to 10,000,000,000 (10^10). Finally, because our outcome variable is categorical and we wish to utilize Logistic Regression we set the family modifier to “binomial.”

```
fitController = trainControl(method = "cv", number = 5,
savePredictions = T)

RidgeReg.model = train(Class~., data=MyData, method = "glmnet",
trControl =
fitController,tuneGrid=expand.grid(.alpha=0,.lambda=10^seq(-10,10,
length=100)), family = "binomial")
```

We can view the lambda value that corresponds to the best model by simply accessing the bestTune from our train object. Here we can observe that the most optimal value was a lambda of 0.05336699, which is fairly close to zero relative to all of the possible values we tested for.

```
RidgeReg.model$bestTune
      alpha     lambda
7    0          0.05336699
```

We also constructed a confusion matrix that resulted from the best model to better understand how accurate our best model is and which observations it typically miscategorized. To do this we have to first create a subset from our train object so we can observe specifically the model that performed the best. To do this we use the subset() function and pass in all the predictions made in the model indicated by `RidgeReg.model$pred` but we are only interested in the predictions made by the best model, so we only retain the observations made from the model with the best lambda value. Finally, we construct a simple table using the **table()** function which compares all predictions against all observations. From this table we can calculate Accuracy by adding together the correctly classified observations and dividing by the total sum of observations predicted. To determine our Error Rate, we simply perform 1 minus our Accuracy. We received 0.9692533 or 96.92% accuracy with an error rate of 0.03074671 or 3.07%, which is comparable to the Error Rate of the 3 variable model from Backward and Forward Selection, however the 4 variable model did perform marginally better and is less complex.

```
sub.ridge=subset(RidgeReg.model$pred,RidgeReg.model$pred$lambda==Ridge
Reg.model$bestTune$lambda)

confMatrix = table(sub.ridge$pred,sub.ridge$obs)
Ridge.Acc = (confMatrix[[1,1]] + confMatrix[[2,2]]) / sum(confMatrix)
Ridge.Error = 1 - Ridge.Acc

Ridge.Acc
0.9692533

Ridge.Error
0.03074671

confMatrix
              Benign     Malignant
  Benign      435        12
  Malignant   9          227
```

Finally, we can also view the Ridge Regression coefficients by simply using the **coef()** function and specifying the model we wish to view by using the optimal lambda value. Considering a relatively small value was chosen for lambda, we do not believe our predictors require a lot of standardization to achieve optimal accuracy.

```
coef(RidgeReg.model$finalModel, RidgeReg.model$bestTune$lambda)
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                                 s1
(Intercept)                      -5.6701745
Clump.Thickness                  0.2126656
Uniformity.of.Cell.Size          0.1427720
Uniformity.of.Cell.Shape         0.1664654
Marginal.Adhesion                0.1375216
Single.Epithelial.Cell.Size      0.1448648
Bare.Nuclei                      0.1919641
Bland.Chromatin                  0.1973242
Normal.Nucleoli                  0.1319703
Mitoses                          0.1449949
```

Lasso
The code for performing Lasso is extremely similar to Ridge Regression. We simply set the alpha value to 1 when creating our models so the **train()** object knows to use Lasso as a means to minimize the coefficients. The rest of the code is mostly identical to Ridge Regression. The result of Lasso's predictive performance was very similar to Ridge Regression but slightly worse.

```
fitController = trainControl(method = "cv", number = 5,
savePredictions = T)

Lasso.model = train(Class~., data=MyData, method = "glmnet", trControl
= fitController,
tuneGrid=expand.grid(.alpha=1,.lambda=10^seq(-10,10, length=100)),
family = "binomial")

Lasso.model$bestTune

     alpha            lambda
41   1                0.01204504

sub.ridge=subset(Lasso.model$pred,Lasso.model$pred$lambda==Lasso.model
$bestTune$lambda)

confMatrix = table(sub.ridge$pred,sub.ridge$obs)
Lasso.Acc = (confMatrix[[1,1]] + confMatrix[[2,2]]) / sum(confMatrix)
Lasso.Error = 1 - Lasso.Acc

Lasso.Acc
[1] 0.966325

Lasso.Error
[1] 0.03367496

confMatrix

                Benign          Malignant
  Benign        435             14
  Malignant     9               225
```

To see if Lasso removed any variables by setting a coefficient value to zero, we again used the **coef()** function to view the coefficient values. Here we found that none of the predictors were eliminated which we assume is the reason the results were so similar. Notably, both Ridge Regression and Lasso did not achieve the best accuracy observed by the 4 variable model - 0.97538119. Considering Lasso and Ridge Regression are considerably more complex than simply removing variables from our model and still did not result in significantly better results, we consider the 4 variable mode the champion of model selection and continue to experiment with it in the next section with GAMs.

```
coef(Lasso.model$finalModel, Lasso.model$bestTune$lambda)
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                                      s1
(Intercept)                           -6.76538363
Clump.Thickness                       0.35775897
Uniformity.of.Cell.Size               0.09708097
Uniformity.of.Cell.Shape              0.22663211
Marginal.Adhesion                     0.14551627
Single.Epithelial.Cell.Size           0.06740412
Bare.Nuclei                           0.30752979
Bland.Chromatin                       0.25849266
Normal.Nucleoli                       0.13978791
Mitoses                               0.05446824
```

Accommodating Non-Linearity

For this section, the team experimented with splines techniques and relaxed the assumption of linearity. We present six total models, two benchmark models that utilize no non-linearity and serve as numerical guides for model performance, and four non-linear models that use splines on some or all of the variables in the model.

The 4 variable model

The team was a bit eager to test and optimize our 4 variable model since it had already achieved great accuracy in the previous section. Since we want our experiments to only specifically test if non-linearity performed better, we thought it best to remove as much randomness from our predictive performance results as possible and utilize Validation Test Set, as K-Cross Fold Validation often resulted in error values that were too similar to definitely attribute performance loss and gains to a model instead of the randomness that comes with resampling. To start we split our dataset into a train and test set using the **sample.split()** function provided by the caTools library. These two subsets will be used for all of the models we construct in this section.

```
split = sample.split(MyData$Class, SplitRatio = 0.7)
train = subset(MyData, split == TRUE)
test = subset(MyData, split == FALSE)
```

To start we fit a benchmark for the 4 variable model by fitting a simple Logistic Regression model using only the four variables from Forward & Backward Selection: Bare.Nuclei, Uniformity.of.Cell.Size, Clump.Thickness, and Normal.Nucleoli. We also construct a confusion matrix and calculate the Accuracy and Error Rate of the model, like we did in the previous section. We see an accuracy of 0.9804878 or 98.05% which is the highest we've observed thus far. A more flexible model will have to beat that accuracy to be considered better.

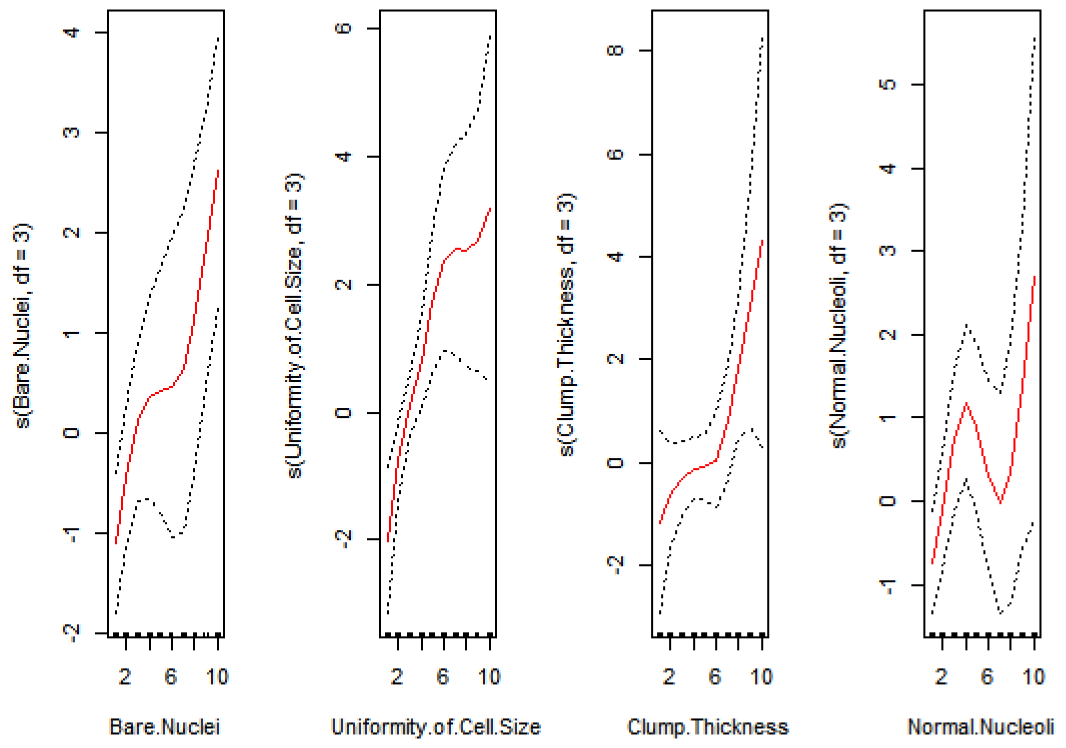
```
glm.fit=glm(Class~Bare.Nuclei+Uniformity.of.Cell.Size+Clump.Thickness+
Normal.Nucleoli,data=train,family=binomial)

glm.probs = predict(glm.fit,test,type="response")

glm.pred = rep("Benign",205)
glm.pred[glm.probs>0.5] = "Malignant"

confmatrix <- table(glm.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
ErrorRate = 1 - accuracy

accuracy
[1] 0.9804878

ErrorRate
[1] 0.0195122
```

To fit our 4 variable model with splines we make use of the **gam()** function which allows us to give each predictor within our model its own function. We started out fairly simple by using the **s()** function and giving each predictor 3 degrees of freedom by using the df parameter. We use the same methodology as the benchmark model to create a confusion matrix and calculate our Accuracy and Error Rate. The accuracy of this model is not considerably better than the simple Logistic Regression model.

```
gam.lr=gam(Class~s(Bare.Nuclei,df=3)+s(Uniformity.of.Cell.Size,df=3)+s
(Clump.Thickness,df=3)+s(Normal.Nucleoli,df=3), family=binomial,
data=train)

gam.probs = predict(gam.lr,test,type="response")

gam.pred = rep("Benign",205)
gam.pred[gam.probs>0.5] = "Malignant"

confmatrix = table(gam.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9707317

ErrorRate
[1] 0.02926829
```

However, we can also view the plots of each predictor variable against its transformed values to see if we observe any non-linearity. We can do this by simply using the **plot()** function and passing in our model, and setting the SE parameter to T in order to view our confidence intervals. Here we see that Normal.Nucleoli is not as linear as the other three variables. So we decide to create another model in which Normal.Nucleoli is transformed, but the other variables are not.

```
par(mfrow=c(1,4))
plot(gam.lr,se=T,col="red")
```



In this model we leave Bare.Nuclei, Uniformity.of.Cell.Size, and Clump.Thickness untransformed, while transforming Normal.Nucleoli to see if it has a positive or negative impact on model performance. We again initialize a new model with **gam()** but this time, we only use **s()** on the Normal.Nucleoli variable, giving it 5 degrees of freedom. This model provides us with our best accuracy from any model yet, 0.9853659 or 98.54% which does outperform our benchmark 4 variable model.

```
gam.lr=gam(Class~Bare.Nuclei+Uniformity.of.Cell.Size+Clump.Thickness+s
(Normal.Nucleoli,df=5), family=binomial, data=train)

gam.probs = predict(gam.lr,test,type="response")

gam.pred = rep("Benign",205)
gam.pred[gam.probs>0.5] = "Malignant"

confmatrix = table(gam.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9853659

ErrorRate
[1] 0.01463415
```

The 9 variable model (all variables)
For the sake of experimentation the team also wanted to perform the same set of tests constructed for the 4 variable model on the complete 9 variable model to both ensure the superior performance of the 4 variable model and be as thorough as possible with our tests. So again we constructed a benchmark logistic regression model that used all of the variables we had (9). This model resulted in an accuracy of 0.9609756, which was already below the benchmark of the 4 variable model. Nonetheless, we continued to experiment.

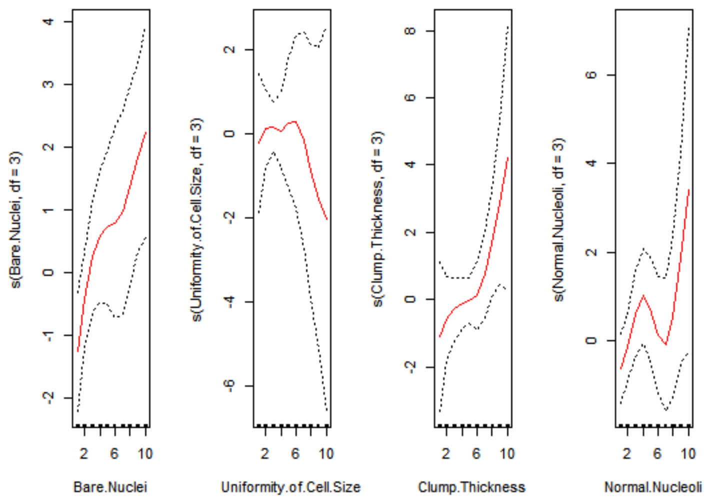
```
glm.fit=glm(Class~.,data=train,family=binomial)

glm.probs = predict(glm.fit,test,type="response")

glm.pred = rep("Benign",205)
glm.pred[glm.probs>0.5] = "Malignant"

confmatrix <- table(glm.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9609756

ErrorRate
[1] 0.03902439
```

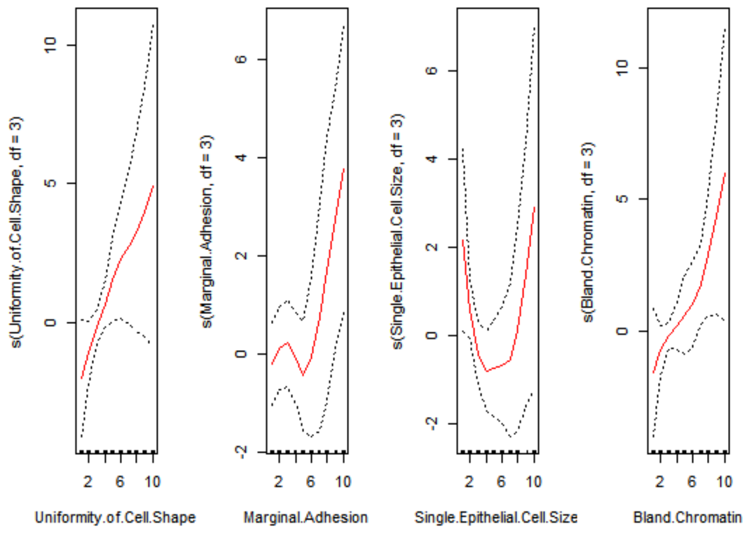
For the next 9 variable model, we gave each predictor a spline with 3 degrees of freedom using the **gam()** and **s()** functions. This model performed better than the benchmark 9 variable model, but importantly did not outperform either of the 4 variable models.

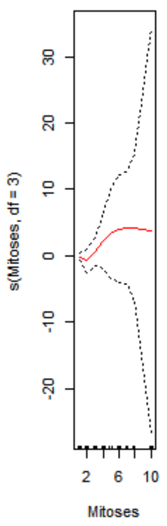
```
gam.lr=gam(Class~s(Bare.Nuclei,df=3)+s(Uniformity.of.Cell.Size,df=3)+s
(Clump.Thickness,df=3)+s(Normal.Nucleoli,df=3)+s(Uniformity.of.Cell.Sh
ape,df=3)+s(Marginal.Adhesion,df=3)+
s(Single.Epithelial.Cell.Size,df=3)+s(Bland.Chromatin,df=3)+s(Mitoses,
df=3), family=binomial, data=train)

gam.probs = predict(gam.lr,test,type="response")

gam.pred = rep("Benign",205)
gam.pred[gam.probs>0.5] = "Malignant"

confmatrix = table(gam.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9658537

ErrorRate
[1] 0.03414634
```

To observe our predictors and their transformations, we again plotted them to observe for any patterns of non-linearity using the **plot()** function. We noted non-linearity in Uniformity of Cell.Size, Normal.Nucleoli, Marginal.Adhesion, and Single.Epithelial.Cell.Size.

```
par(mfrow=c(1,4))
plot(gam.lr,se=T,col="red")
```







s(Mitoses, df = 3)
30
20
10
0
-10
-20
2
6
10
Mitoses

Based on our observations from the plots generated by the previous model, we constructed our last model by giving the predictors where we observed nonlinearity splines with 5 degrees of freedom, while keeping all other variables normal. The accuracy of this model was 0.9560976 which does not even beat the benchmark 9 variable model, and thus is not a useful model to implement.

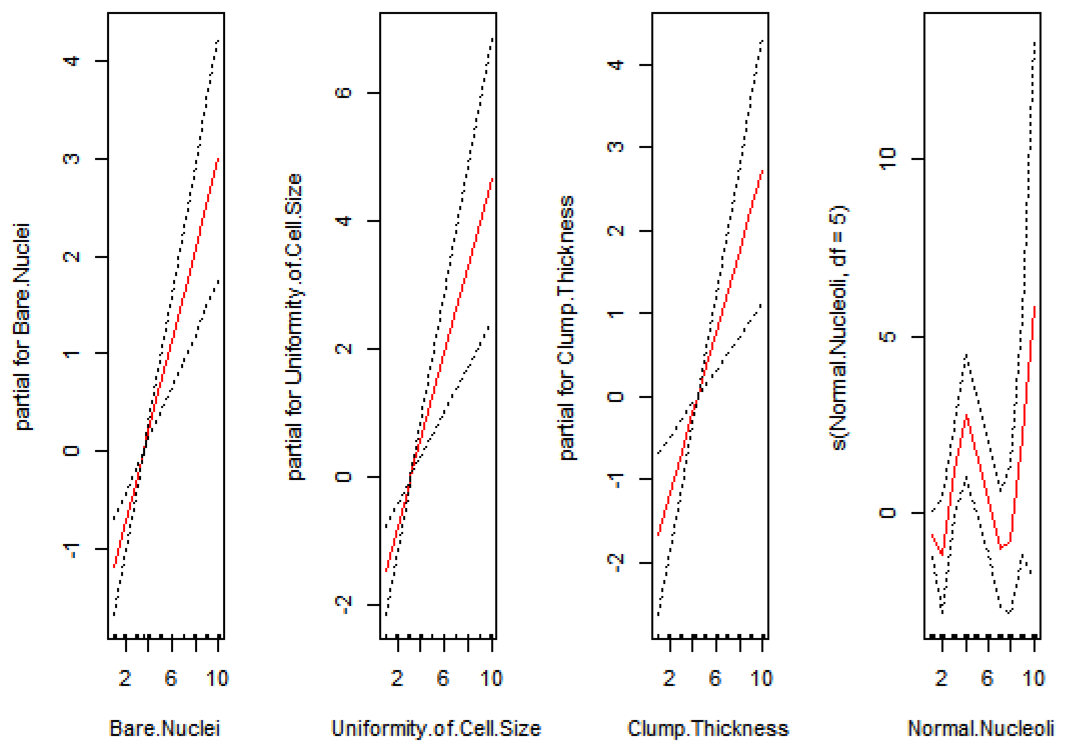
```
gam.lr=gam(Class~Bare.Nuclei+s(Uniformity.of.Cell.Size,df=5)+Clump.Thi
ckness+s(Normal.Nucleoli,df=5)+Uniformity.of.Cell.Shape+s(Marginal.Adh
esion,df=5)+s(Single.Epithelial.Cell.Size,df=5)+Bland.Chromatin+Mitose
s, family=binomial, data=train)

gam.probs = predict(gam.lr,test,type="response")

gam.pred = rep("Benign",205) #There are 205 observations in the Test
Subset
gam.pred[gam.probs>0.5] = "Malignant"

confmatrix = table(gam.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9560976

ErrorRate
[1] 0.04390244
```

For convenience and comparison we have constructed a table listing the various error rates from all the models tested. As the table shows, the best model was the 4 variable model w/ Spines for some predictors as it had the highest accuracy. Thus, we claim it as our final model. Additionally, we attach the plots of the predictors and their transformations by using the **plot()** function along with the code specifically for the final model.

Accuracy Comparison of Generated Models

	4 Variable Model Accuracy	9 Variable Model Accuracy
Benchmark Model (Logistic Regression)	0.9804878	0.9609756
Model w/ Splines for all Predictors	0.9707317	0.9658537
Model w/ Splines for Some Predictors	0.9853659	0.9560976

```
gam.lr=gam(Class~Bare.Nuclei+Uniformity.of.Cell.Size+Clump.Thickness+
s(Normal.Nucleoli,df=5), family=binomial, data=train)

gam.probs = predict(gam.lr,test,type="response")

gam.pred = rep("Benign",205)
gam.pred[gam.probs>0.5] = "Malignant"

confmatrix = table(gam.pred,test$Class)

accuracy = (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)

ErrorRate = 1 - accuracy

accuracy
[1] 0.9853659

ErrorRate
[1] 0.01463415

par(mfrow=c(1,4))
plot(gam.lr,se=T,col="red")
```


partial for Bare.Nuclei
Bare.Nuclei
partial for Uniformity.of.Cell.Size
Uniformity.of.Cell.Size
partial for Clump.Thickness
Clump.Thickness
s(Normal.Nucleoli, df = 5)
Normal.Nucleoli