622_HW1

```r
library(naivebayes)
library(class)
library(gmodels)
library(tidyverse)
library(caret)
library(pROC)
library(kableExtra)
library(class)
library(ROCR)
```

```r
df <- read.csv('https://raw.githubusercontent.com/san123i/CUNY/master/Semester4/622-MachineLearning/Ass
df$label <- as.factor(df$label)
df$Y <- as.factor(df$Y)

summary(df)
```

```
##        X         Y       label
##  Min.   : 5   a:6   BLACK:22
##  1st Qu.:19   b:6   BLUE :14
##  Median :43   c:6
##  Mean   :38   d:6
##  3rd Qu.:55   e:6
##  Max.   :63   f:6
```

```r
str(df)
```

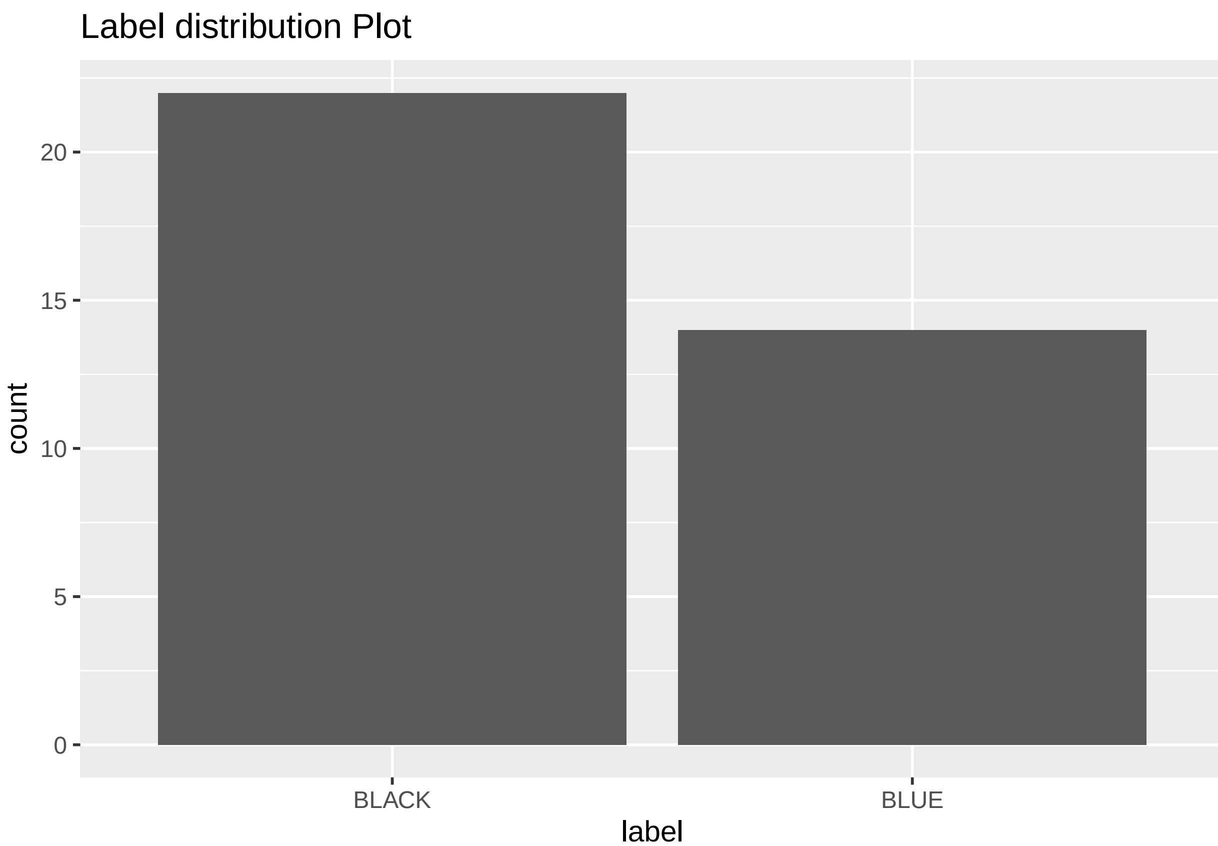
```
## 'data.frame':    36 obs. of  3 variables:
##  $ X    : int  5 5 5 5 5 5 19 19 19 19 ...
##  $ Y    : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6 1 2 3 4 ...
##  $ label: Factor w/ 2 levels "BLACK","BLUE": 2 1 2 1 1 1 2 2 2 2 ...
```

```r
head(df)
```

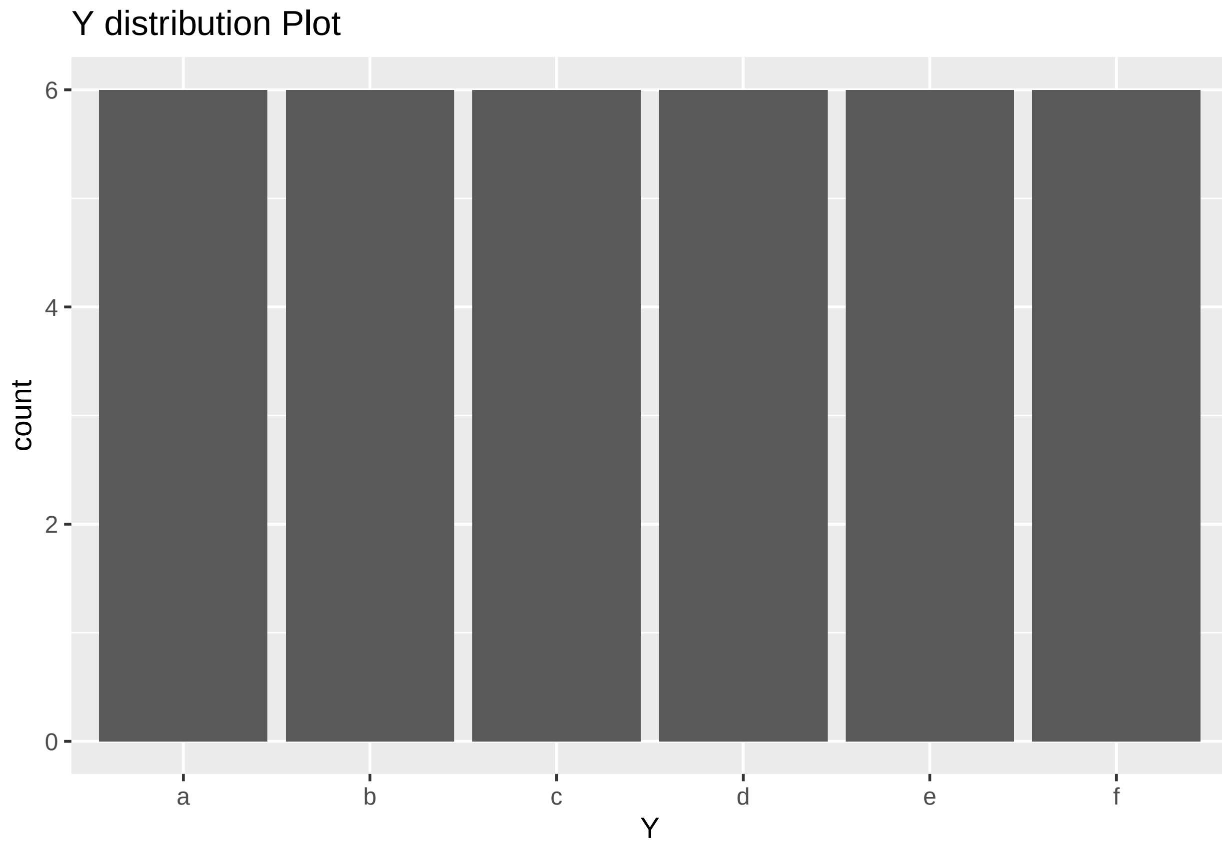
```
##   X Y label
## 1 5 a  BLUE
## 2 5 b BLACK
## 3 5 c  BLUE
## 4 5 d BLACK
## 5 5 e BLACK
## 6 5 f BLACK
```

Explorative Data Analysis

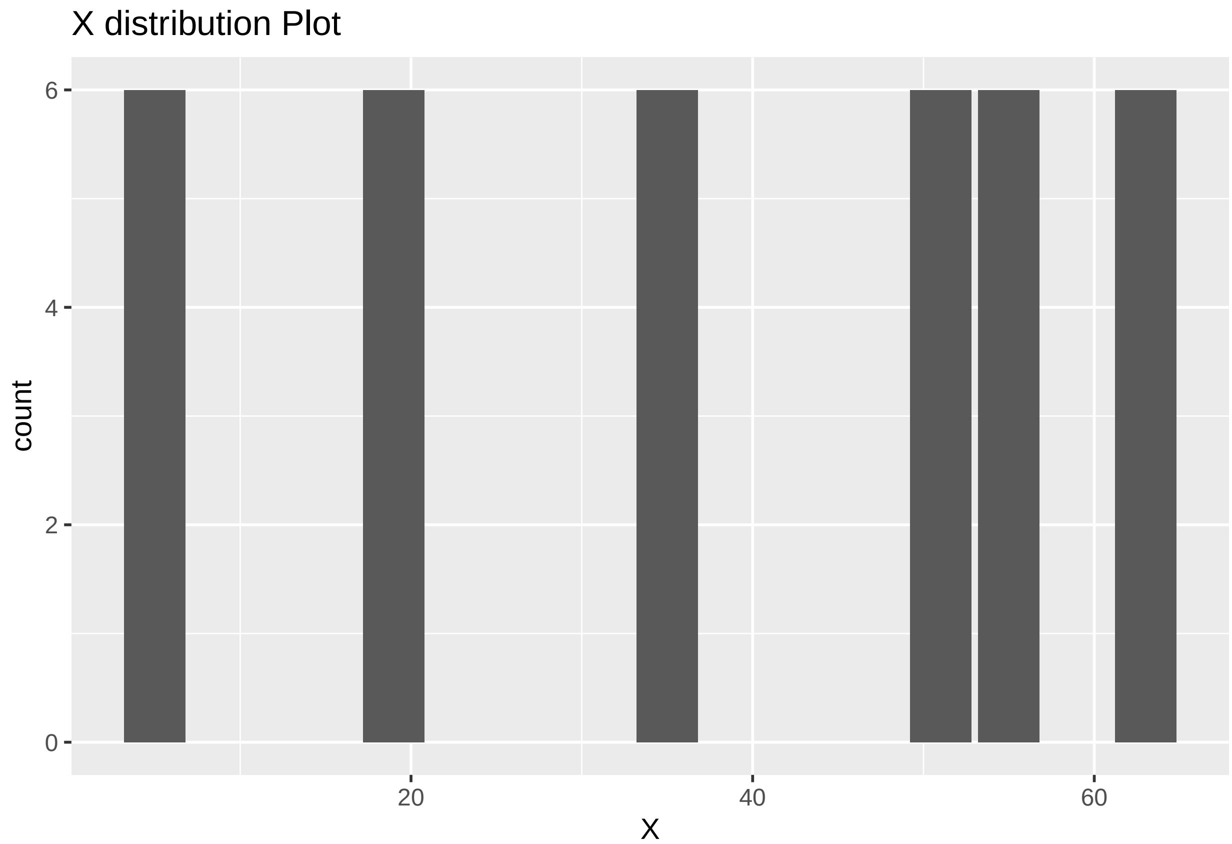
```r
df %>% ggplot(aes(x=label)) +
        geom_bar() +
  ggtitle("Label distribution Plot")
```



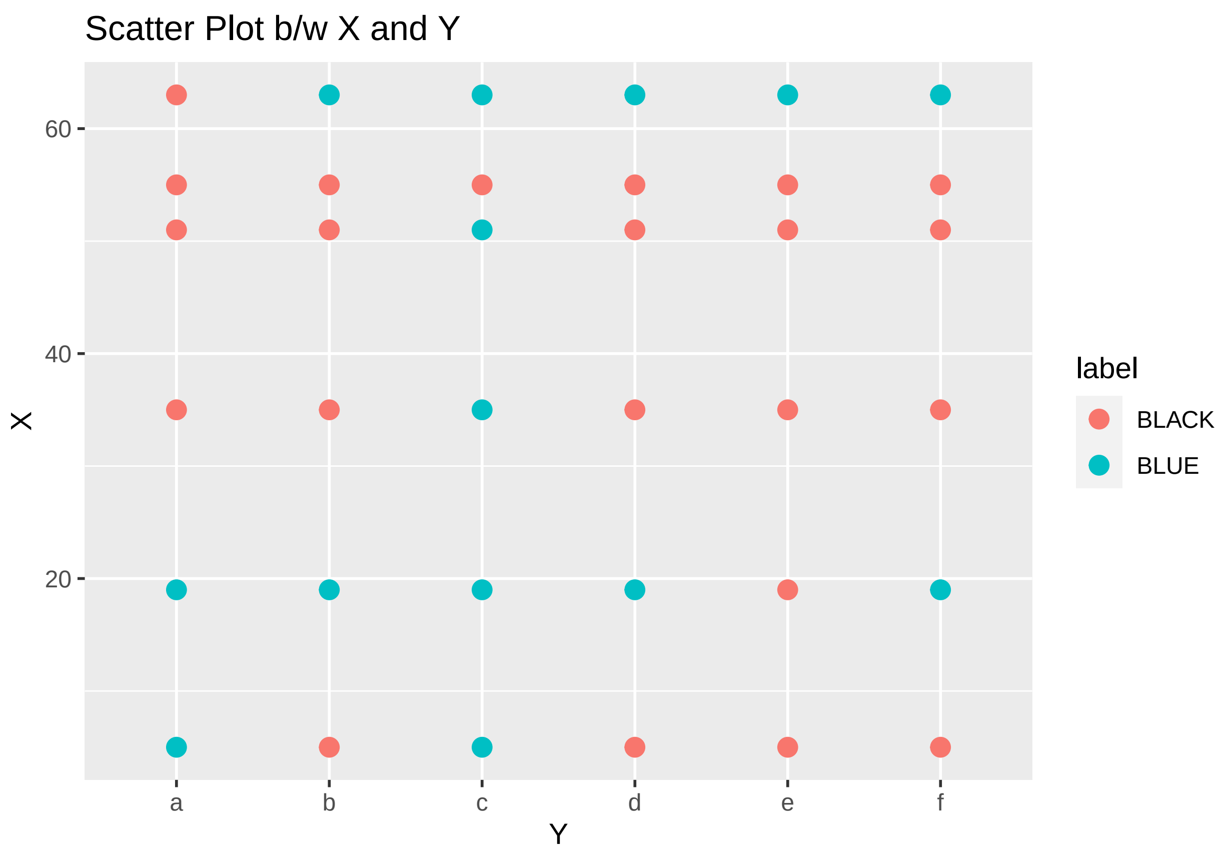

```r
df %>% ggplot(aes(x=Y)) +
        geom_bar() +
  ggtitle("Y distribution Plot")
```




```
df %>% ggplot(aes(x=X)) +
        geom_bar() +
  ggtitle("X distribution Plot")
```

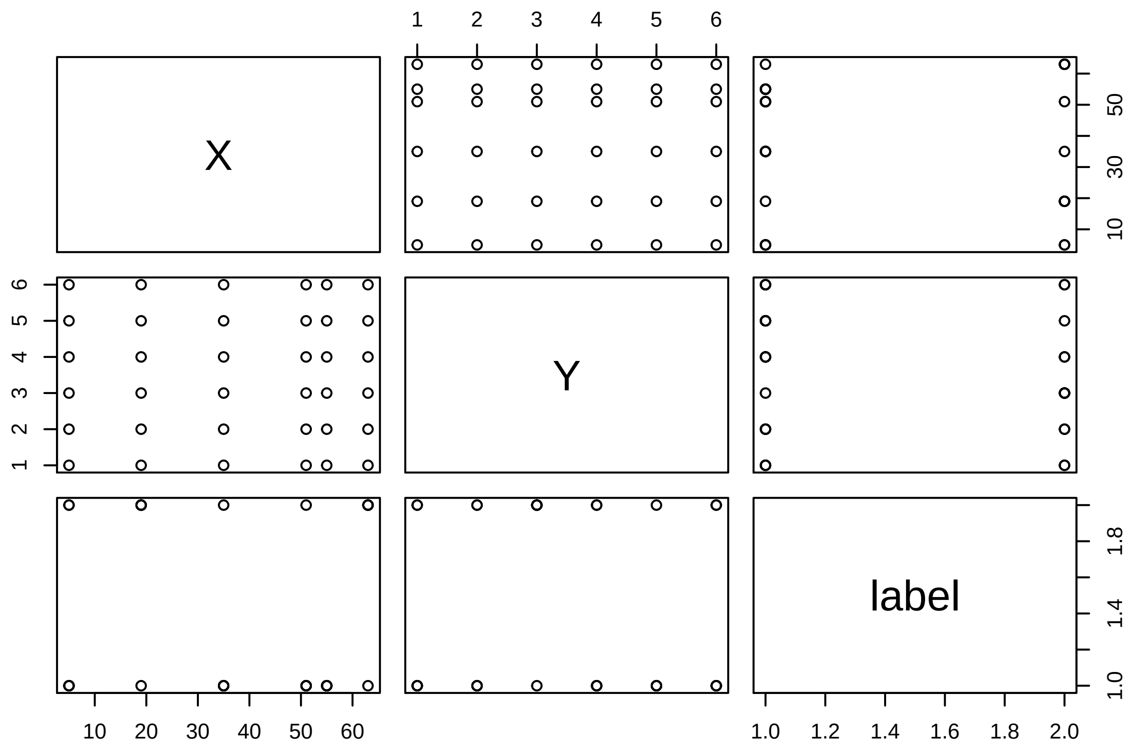



```
ggplot() + geom_point(data = df, aes(x = Y, y = X, color = label), size=3) + ggtitle("Scatter Plot b/w
```



Correlation variables

```
pairs(df)
```



Split the data

```r
set.seed(1234)

ind <- sample(2, nrow(df), replace = T, prob=c(0.7,0.3))
train <- df[ind ==1,]
test <- df[ind==2,]

train
```

```
##      X Y label
## 1    5 a  BLUE
## 2    5 b BLACK
## 3    5 c  BLUE
## 4    5 d BLACK
## 6    5 f BLACK
## 7   19 a  BLUE
## 8   19 b  BLUE
## 9   19 c  BLUE
## 10  19 d  BLUE
## 11  19 e BLACK
## 12  19 f  BLUE
## 13  35 a BLACK
## 15  35 c  BLUE
```

```
## 17 35 e BLACK
## 18 35 f BLACK
## 19 51 a BLACK
## 20 51 b BLACK
## 21 51 c  BLUE
## 22 51 d BLACK
## 23 51 e BLACK
## 24 51 f BLACK
## 25 55 a BLACK
## 27 55 c BLACK
## 30 55 f BLACK
## 31 63 a BLACK
## 32 63 b  BLUE
## 33 63 c  BLUE
## 34 63 d  BLUE
## 35 63 e  BLUE
```

```
test
```

```
##     X Y label
## 5   5 e BLACK
## 14 35 b BLACK
## 16 35 d BLACK
## 26 55 b BLACK
## 28 55 d BLACK
## 29 55 e BLACK
## 36 63 f  BLUE
```

NaiveBayes Classifier

```
nb_model <- naive_bayes(label~., data=df)
summary(nb_model)
```

```
##
## ================================== Naive Bayes ==================================
##
## - Call: naive_bayes.formula(formula = label ~ ., data = df)
## - Laplace: 0
## - Classes: 2
## - Samples: 36
## - Features: 2
## - Conditional distributions:
##     - Categorical: 1
##     - Gaussian: 1
## - Prior probabilities:
##     - BLACK: 0.6111
##     - BLUE: 0.3889
##
## ---------------------------------------------------------------------------------
```

```r
trainPredict <- predict(nb_model, train)

#ConfusionMatrix on Train data
ConfusionMatrix <- table(trainPredict, train$label)
ConfusionMatrix
```

```
##
## trainPredict BLACK BLUE
##        BLACK    15    8
##        BLUE      1    5
```

```r
model_metrics_df <- data.frame(Type=NA, Algo=NA, AUC=NA, ACCURACY=NA,TPR=NA,FPR=NA,TNR=NA,FNR=NA)

model_metrics_df <- gather_metrics_func('Train', model_metrics_df, 'NB',trainPredict, train)

testPredict <- predict(nb_model, test)

#ConfusionMatrix on Test data
ConfusionMatrix <- table(testPredict, test$label)
ConfusionMatrix
```

```
##
## testPredict BLACK BLUE
##       BLACK     6    1
##       BLUE      0    0
```

```r
model_metrics_df <- gather_metrics_func('Test', model_metrics_df, 'NB',testPredict, test)
```

Since the model is successfully predicting values on test dataset(i.e., new data/unseen data) more than training set accuracy, we can say that it is generalizable.

Logistic Regression

```r
lm_model <- glm(label~., data=train, family=binomial(link="logit"))
summary(lm_model)
```

```
##
## Call:
## glm(formula = label ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7967  -0.8274  -0.5873   1.0835   1.8081
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.15613    1.14121  -0.137   0.8912
## X           -0.01451    0.02031  -0.714   0.4751
## Yb           0.65673    1.33991   0.490   0.6240
```

```
## Yc           2.34610    1.41185   1.662   0.0966 .
## Yd           0.65673    1.33991   0.490   0.6240
## Ye          -0.34758    1.45593  -0.239   0.8113
## Yf          -0.77402    1.43062  -0.541   0.5885
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 39.892  on 28  degrees of freedom
## Residual deviance: 33.119  on 22  degrees of freedom
## AIC: 47.119
##
## Number of Fisher Scoring iterations: 4
```

```r
predict_lr_train <- predict(lm_model, newdata=train, type = "response")
predict_lr_train<- ifelse(predict_lr_train<0.5,"BLACK","BLUE" )
predict_lr_train <- as.factor(predict_lr_train)

model_metrics_df <- gather_metrics_func('Train', model_metrics_df, 'LR',predict_lr_train, train)

predict_lr_test <- predict(lm_model, newdata=test, type = "response")
predict_lr_test<- ifelse(predict_lr_test<0.5,"BLACK","BLUE" )
predict_lr_test <- as.factor(predict_lr_test)

model_metrics_df <- gather_metrics_func('Test', model_metrics_df, 'LR',predict_lr_test, test)
```

KNN - 3

```r
train_knn <- train[,c(1,2)]
test_knn <- test[,c(1,2)]

train_labels <- train[,3]
test_labels <- test[,3]

train_knn$Y = as.numeric(train_knn$Y)
test_knn$Y = as.numeric(test_knn$Y)


knn_3 = knn3(label ~ ., data = train, k = 3)

predict_knn3_train <- predict(knn_3, train, type = "class")

predict_knn3_test <- predict(knn_3, test, type = "class")

model_metrics_df <- gather_metrics_func('Train', model_metrics_df, 'KNN3',predict_knn3_train, train)
model_metrics_df <- gather_metrics_func('Test', model_metrics_df, 'KNN3',predict_knn3_test, test)
```

KNN - 5

```r
knn_5 = knn3(label ~ ., data = train, k = 5)

predict_knn5_train <- predict(knn_5, train, type = "class")

predict_knn5_test <- predict(knn_5, test, type = "class")

model_metrics_df <- gather_metrics_func('Train', model_metrics_df, 'KNN5',predict_knn5_train, train)
model_metrics_df <- gather_metrics_func('Test', model_metrics_df, 'KNN5',predict_knn5_test, test)
```

Training set stats - Ability to Learn

	Type	Algo	AUC	ACCURACY	TPR	FPR	TNR	FNR
2	Train	NB	0.6611	0.6897	0.9375	0.6154	0.3846	0.0625
3	Train	LR	0.6755	0.6897	0.8125	0.4615	0.5385	0.1875
5	Train	KNN3	0.7837	0.7931	0.875	0.3077	0.6923	0.125
7	Train	KNN5	0.8221	0.8276	0.875	0.2308	0.7692	0.125

Testing set stats - Ability to generalize

	Type	Algo	AUC	ACCURACY	TPR	FPR	TNR	FNR
21	Test	NB	0.5	0.8571	1	1	0	0
4	Test	LR	0.5	0.8571	1	1	0	0
6	Test	KNN3	1	1	1	0	1	0
8	Test	KNN5	1	1	1	0	1	0

Observations

From the above stats, we can observe that KNN(with k=5) performed the better in both training and testing datasets. Therefore, we can say that it is able to learn as well as generalize better than other data models.

Understanding the algorithms (In simple client language)

<table>
<tr><td colspan="2"></td><th colspan="2">Actual class</th></tr>
<tr><td colspan="2"></td><th>P</th><th>N</th></tr>
<tr><th rowspan="2">Predicted class</th><th>P</th><td>TP</td><td>FP</td></tr>
<tr><th>N</th><td>FN</td><td>TN</td></tr>
</table>

where: P = Positive; N = Negative; TP = True Positive; FP = False Positive; TN = True Negative; FN = False Negative.

Accuracy - Ability to predict the result accurately Sensitivity - Proportion of true positives correctly identified Specificity - Proportion of true negatives correctly identified

The above 3 parameters are very critical in choosing the right algorithm

NB(Naive Bayes): Naive Bayes algorithm predicts the output based on probabilities, and it has predicted with high accuracy in both training and test results, but specificity was lagging.

Logistic Regression: Logistic regression calculates the probability based on regression output, and it too had similar accuracy predictions as NB, and was lagging in specificity.

KNN: KNN algorithm calculates the probability based on the nearest neighbors of the identified data point. It was able to predict much accurately than NB and LR, when we used 5 neighbors to classify the data point.

Result: Since KNN had best predictions and accuracy, we can choose KNN for this dataset.