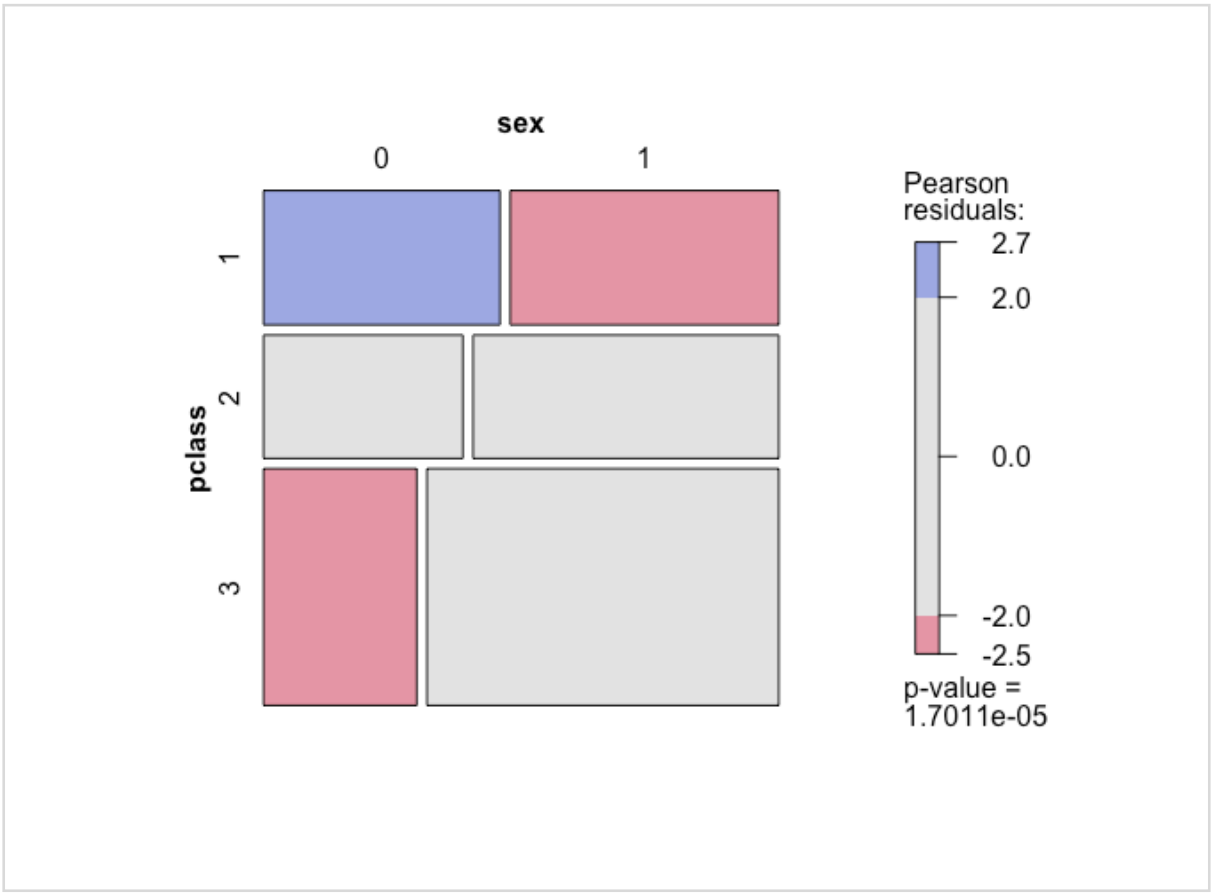
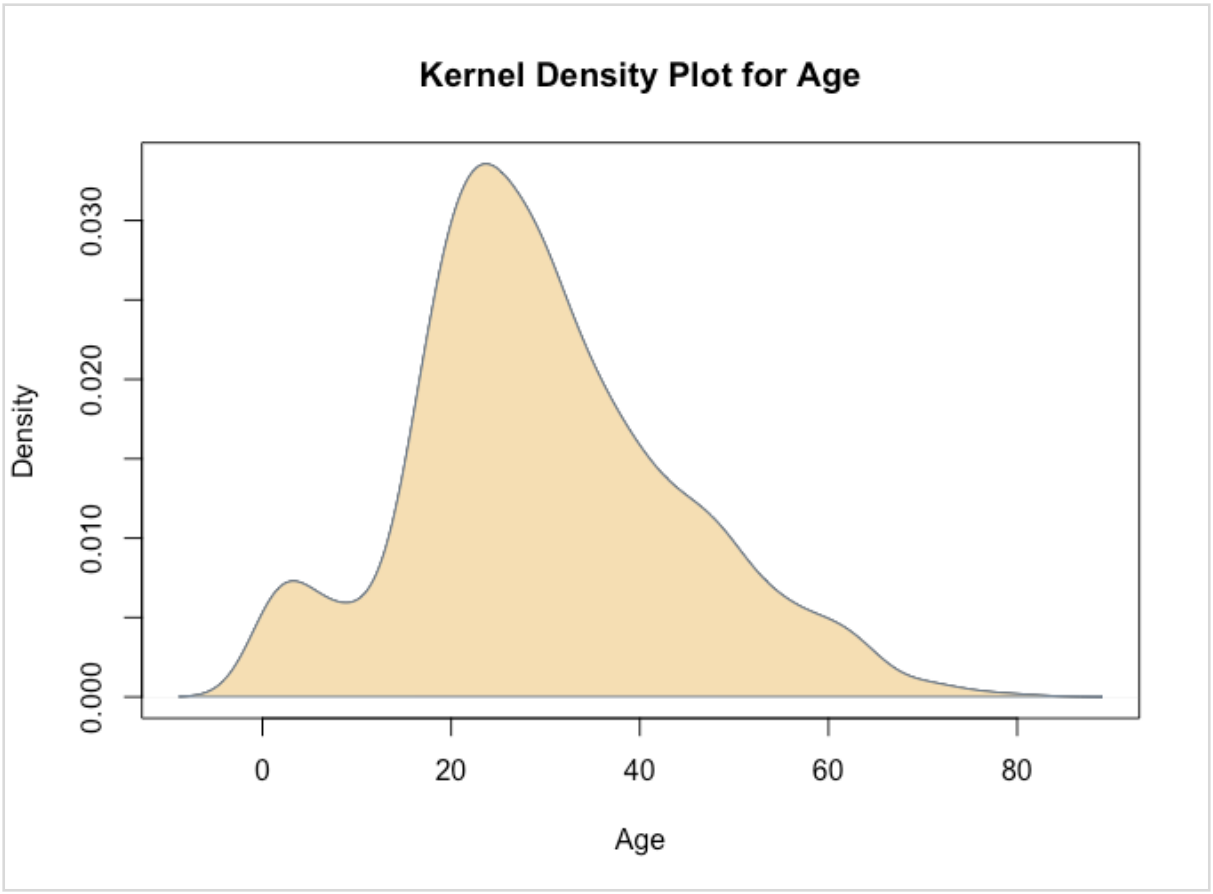
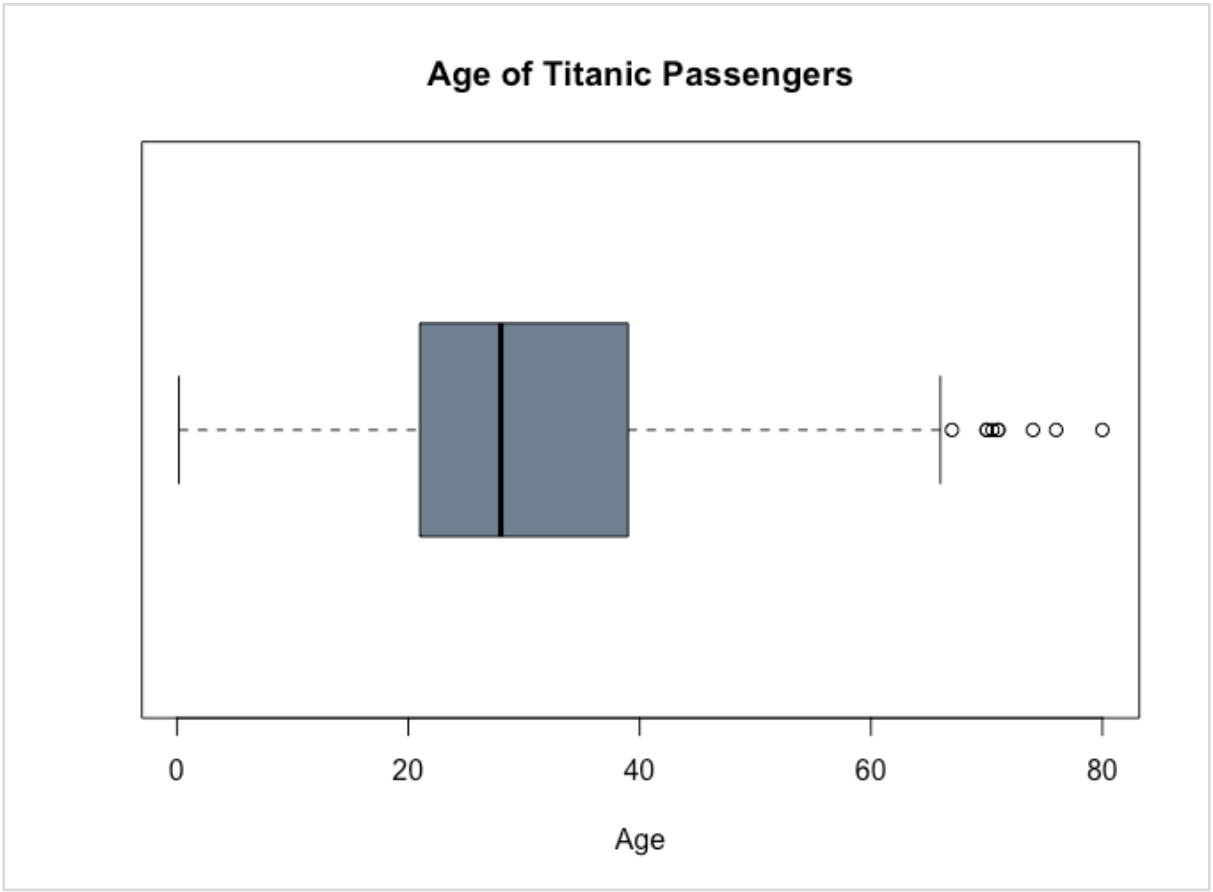
Project1 Report

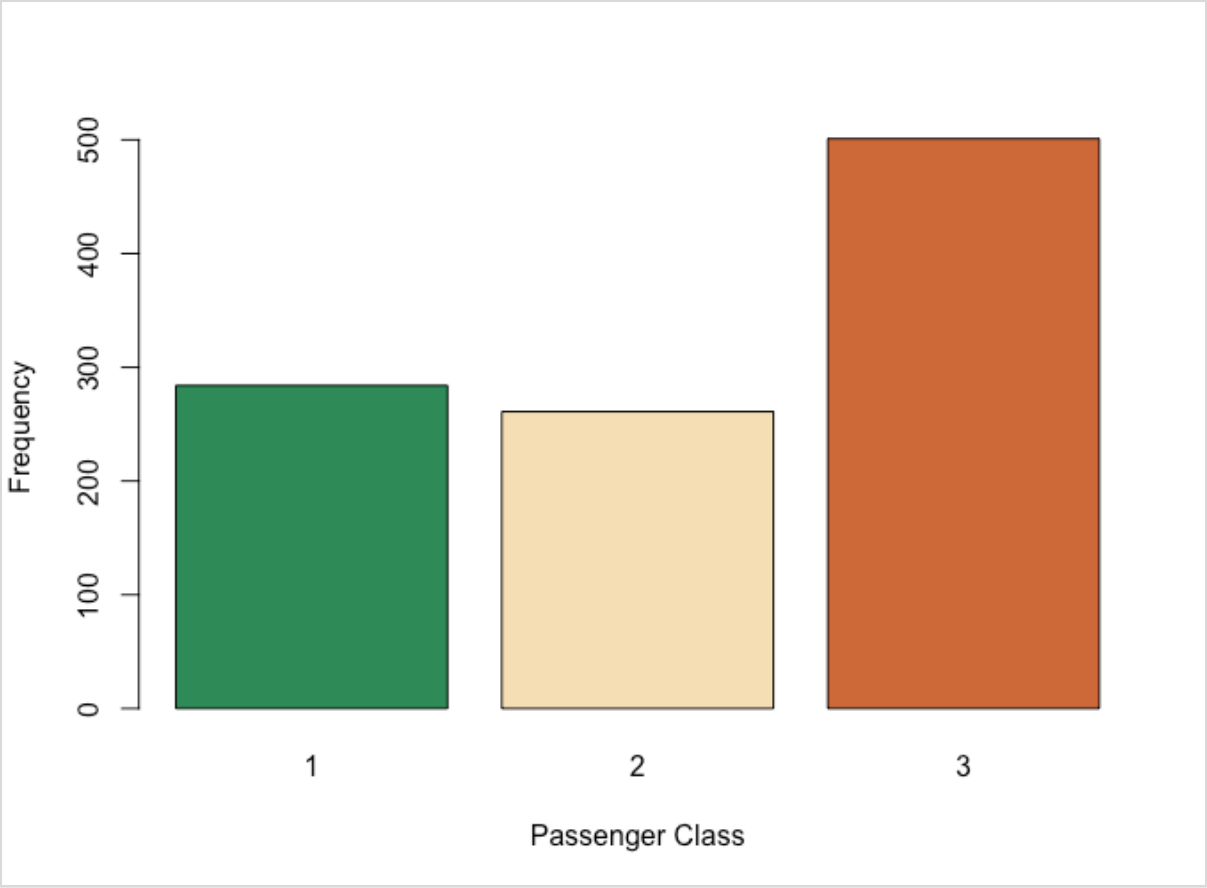
I used a struct containing vectors as my data structure for both Logistic Regression and Naive Bayes.

Data Exploration Graphs




Kernel Density Plot for Age
Density
0.000
0.010
0.020
0.030
0
20
40
60
80
Age


Age of Titanic Passengers
0
20
40
60
80
Age



Logistic Regression Report Section

Write a summary of the two implementations, R and C++. Did you get the same results? How do the run times compare? How did you measure execution time?

I received the same results on both the C++ and R implementation, for accuracy, sensitivity, and specificity. I implemented an early stop in the iterations of the gradient descent when the difference between the current and last weights is less than .0000001.

The run time of the C++ Code ended up being slower, .618 seconds in C++ as opposed to .415 seconds in R. I believe this is because of the way I implemented the C++ algorithm. I created my own vector matrix multiplication algorithm instead of using armadillo, which is probably why it wasn't as efficient. For the matrix, I used a vector of vectors.

The execution time was measured by starting the chrono timer right before the gradient descent loop, and ending right after the loop.

Include screen shots of the output and runtime of each program

C++

```
@"WEIGHTS AFTER GRADIENT DESCENT ITERATION 542: 1.29716 and -0.779927\r\n"
@" PROGRAM TERMINATED ON ITERATION 542 WITH WEIGHTS: 1.29716 and -0.779927\r\n"
@"OBSERVATION 0 Survival Probability: 0.260645 and Classification: 0\r\n"
@"OBSERVATION 1 Survival Probability: 0.260645 and Classification: 0\r\n"
@"OBSERVATION 2 Survival Probability: 0.626501 and Classification: 0\r\n"
@"OBSERVATION 3 Survival Probability: 0.260645 and Classification: 0\r\n"
@"OBSERVATION 4 Survival Probability: 0.626501 and Classification: 0\r\n"
@"OBSERVATION 5 Survival Probability: 0.626501 and Classification: 0\r\n"
@"OBSERVATION 6 Survival Probability: 0.626501 and Classification: 0\r\n"
@"OBSERVATION 7 Survival Probability: 0.260645 and Classification: 0\r\n"
@"OBSERVATION 8 Survival Probability: 0.626501 and Classification: 0\r\n"
@"OBSERVATION 9 Survival Probability: 0.260645 and Classification: 0\r\n"
@"\r\n"
@" METRICS -> \r\n"
@"Accuracy: 0.671233\r\n"
@"Sensitivity: 0.462687\r\n"
@"Specificity: 0.848101\r\n"
@"TOTAL RUNTime: 0.618008\r\n"
```

R

```
> print(paste("TRAINING LOGREGMODEL RUNTIME: ", runtime, "seconds"))
[1] "TRAINING LOGREGMODEL RUNTIME:  0.415715932846069 seconds"
> # print logistic regression model coefficients
> print("MODEL COEFFICIENTS: ")
[1] "MODEL COEFFICIENTS: "
> print(logRegModel$coefficients)
(Intercept)     pclass2     pclass3
  0.4997758  -0.7250224  -1.5539363
> print("confusion Matrix:")
[1] "confusion Matrix:"
> print(table(pred, test$survived))

pred  0  1
   0 67 36
   1 12 31
> ## Accuracy
> print(paste("ACCURACY:", accuracy))
[1] "ACCURACY: 0.671232876712329"
> ## Sensitivity
> print(paste("SENSITIVITY:", sensitivity))
[1] "SENSITIVITY: 0.462686567164179"
> ## Specificity
> print(paste("SPECIFICITY:", specificity))
[1] "SPECIFICITY: 0.848101265822785"
```

Naive Bayes Report Section

Write a summary of the two implementations, R and C++. Did you get the same results? How do the run times compare? How did you measure execution time?

I received the same results on both the C++ and R implementation, for accuracy, sensitivity, and specificity.

The runtime of the C++ code ended up being vastly faster than the R code, with only .000173 seconds against the R code which took .721 seconds.

The execution time was measured by starting the timer right before starting the iteration through the test set and ending right after.

Include screen shots of the output and runtime of each program

C++

I measured execution time by starting the clock as soon as the train and test data was split, including all the calculations that are required to prepare for calculating the likelihood.

```
@"Num Survived vs Perished: 360 540\r\n"
@"Prior Probabilities: 0.4 0.6\r\n"
@"PCLASS LIKELIHOODS: 0.416667 0.263889 0.319444\r\n"
@"SEX LIKELIHOODS: 0.159259 0.840741\r\n"
@"MEAN AND VAR PERISHED: 30.4168 201.603\r\n"
@"MEAN AND VAR SURVIVED: 28.9206 227.098\r\n"
@"RAWPROBS FOR OBS 0: 0.365132 0.634868\r\n"
@"RAWPROBS FOR OBS 1: 0.892183 0.107817\r\n"
@"RAWPROBS FOR OBS 2: 0.648374 0.351626\r\n"
@"RAWPROBS FOR OBS 3: 0.89139 0.10861\r\n"
@"RAWPROBS FOR OBS 4: 0.646372 0.353628\r\n"
@"RUNTime: 0.000173\r\n"
@"\r\n"
@"METRICS -> \r\n"
@"Accuracy: 0.760274\r\n"
@"Sensitivity: 0.626866\r\n"
@"Specificity: 0.873418\r\n"
```

R

```
> print(paste("TRAINING NAIVEBAYES MODEL RUNTIME:", runtime))
[1] "TRAINING NAIVEBAYES MODEL RUNTIME: 0.721612930297852"
> print("naiveBayesModel:")
[1] "naiveBayesModel:"
> print(naiveBayesModel)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
  0   1
0.6 0.4

Conditional probabilities:
   pclass
Y           1         2         3
  0 0.1685185 0.2203704 0.6111111
  1 0.4166667 0.2638889 0.3194444

   sex
Y           0         1
  0 0.1592593 0.8407407
  1 0.6944444 0.3055556

   age
Y       [,1]     [,2]
  0 30.41682 14.21185
  1 28.92060 15.09074
```

```
> print("Confusion Matrix")
[1] "Confusion Matrix"
> print(confusionMatrix(pred, as.factor(test$survived)))
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 69 25
         1 10 42

               Accuracy : 0.7603
                 95% CI : (0.6827, 0.827)
    No Information Rate : 0.5411
    P-Value [Acc > NIR] : 3.612e-08

                  Kappa : 0.5089

 Mcnemar's Test P-Value : 0.01796

            Sensitivity : 0.8734
            Specificity : 0.6269
         Pos Pred Value : 0.7340
         Neg Pred Value : 0.8077
             Prevalence : 0.5411
         Detection Rate : 0.4726
   Detection Prevalence : 0.6438
      Balanced Accuracy : 0.7501

       'Positive' Class : 0

> print(paste("ACCURACY", accuracy))
[1] "ACCURACY 0.76027397260274"
> print(paste("SENSITIVITY", sensitivity))
[1] "SENSITIVITY 0.626865671641791"
> print(paste("SPECIFICITY", specificity))
[1] "SPECIFICITY 0.873417721518987"
```