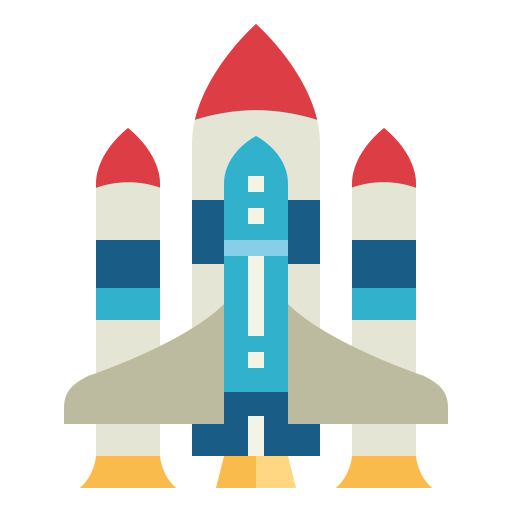
***CS 4661 Final Progress Report(Space Ship Titanic)***

*By: Kobe Martinez, Daniel Rodas, Gregory Celestino, Angel Serrano, Santiago Bautista*

**Main Goal:** *To experiment with various machine learning algorithms and find out which algorithm creates the best accuracy in predicting who got transported and who did not.*

**Project Description:**

*We are using data about the Space Titanic. We need to figure out who was transported or not by using the training data set for accuracy measurement and testing data. One of the goals for this project is to make use of all the algorithms we have learned throughout the semester to work correctly and in the most efficient way possible. Secondly, we must test the accuracy with many values (i.e., Kvalues, maxiter, random state, etc.).Thirdly, we must use the testing data frame and the algorithms we created to predict who got transported. Lastly, we must illustrate which was the best of all algorithms that best accurately predicted who was transported or not.*

**Team Member Responsibilities:**

* ***Leader****: Greg- Editor/ Commentor / Algorithm Efficiency/ Leader*
* *Daniel- Algorithm Efficiency/ Code Commentor*
* *Kobe- Document Manager/ Graph Creator / Algorithm Efficiency*
* *Angel- Editor / Commentor / Algorithm Efficiency*
* *Santiago- Algorithm Efficiency / Editor*

**Project Data Details:**

*The data used for our analysis of the Space Titanic came from Kaggle. We were given two sets of data, ‘train.csv’ and ‘test.csv’, but will only be using ‘train.csv.’ This is because ‘test.csv’ was unsupervised, it didn’t have a label. This means the supervised dataset, ‘train.csv,’ would be more reliable and as a result the only dataset we used for our analysis. This dataset includes*

* *PassengerId:*
  + *A unique Id for each passenger*
  + *Takes the form of ‘gggg\_pp’*
  + *‘gggg’ indicates a group the passenger is traveling with*
  + *‘pp’ is their number within the group*
* *HomePlanet*
  + *The planet the passenger departed from*
* *CryoSleep*
  + *Whether passenger chose to be put in the state of cryosleep(suspended animation) during the voyage.*
* *Cabin*
  + *Cabin number passenger is staying in.*
  + *Takes the form ‘deck/num/side’*
  + *‘Side’ can be either ‘P’ for Port or ‘S’ for Starboard*
* *Destination*
  + *Planet the passenger is heading to*
* *Age*
  + *Age of passenger*
* *VIP*
  + *Whether the passenger has VIP status or not*
* *RoomService, FoodCourt, ShoppingMall, Spa, VRDeck*
  + *Amount of money that a passenger has spent on the Spaceship Titanic’s services*
* *Name*
  + *First and last name*
* *Transported*
  + *Whether the passenger was transported to another dimension*
  + *Our target label*

*Important Note: We decided to choose the features we believe are the most important:*

* *HomePlanet*
* *CryoSleep*
* *Age*
* *VIP*

*Reason: The reason we decided to choose these specific features is that when it comes to Age, survival rates drastically change the outcome of who survives or not, logically speaking if a person is old, there are more chances they wouldn't make it out alive since old they need much more help to get back on the ship than the young passengers. When there are humans there will always be that one group of humans that discriminate where they come from, we believe that people will be selective in who receives help. If the passengers are asleep during this, we believe that it will be too late for them to be saved. Finally, VIP members we believe will receive the best treatment and assurance of survival in getting back on board.*

**Developed Methods and Algorithms:**

*With the project, the algorithms that the team decided on to meet the requirements were the ones done in the semester. That includes the KNN classifier, Decision Tree Classifier, Logistic Regression Classifier, Random Forest, and Linear Regression. With these algorithms, we also use different methods for these algorithms which are, Cross Validation, Standard Scalar, Standard Scalar with Cross Validation, Bagging and Voting, and for every method done we have created an AOC curve for it.*

*When it comes to the training sets, we train split the training and testing data uniqually before each algorithm section. For instance, we only do one train split code before using KNN algorithms, then one for the decision tree section, logistic regression, etc.*

***Tools Used:***

* *Google Collab in working together*
* *Github*
* *Jupyter Notebook*

**Developed Code and Results:**

***KNN Classifier Results***

*For the KNN classifier, we first created a for loop to read through a list of various k values and print the k values with their respective accuracies. We used a list within the range of 1 through 31 by 5 which showed us enough information in which what k value no longer benefited the algorithm. For instance, beyond the k value of 21, the accuracies started to come to a halt.*

*Accuracy on Testing Data of KNN Classifier with k=1: 0.65*

*Accuracy on Testing Data of KNN Classifier with k=6: 0.69*

*Accuracy on Testing Data of KNN Classifier with k=11: 0.72*

*Accuracy on Testing Data of KNN Classifier with k=16: 0.72*

*Accuracy on Testing Data of KNN Classifier with k=21: 0.73*

*Accuracy on Testing Data of KNN Classifier with k=26: 0.73*

*As you can see from our output, we can see the trajectory of the accuracies from their respective k values from the k value 1 being the lowest to slowly increasing up until k value 21. After the loop was finished we made sure to print the best k value with the best accuracy. Once we were finished with the code we decided to test through different values for test\_size and random\_state within the train split function*

*‘ X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=9)’*

*After vigours testing we concluded that making 80 percent of the data go towards training and the rest of the 20 percent will go towards the testing data gave us the best results for our KNN classifier. As the final run of the updated code, we concluded that the k value of 21 provided the best accuracy, 73 percent.*

***KNN with Cross Validation:***

*For the KNN with Cross Validation algorithm, we mirrored the same logical aspect of the original KNN code but in this case, we added Cross Validation. Following the same logical process we have a for loop that reads through a list of k values from a range of 1 through 31 by 5 with the same train split data and print out the various accuracies for their respective k values. To use cross validation we first called in the necessary import from the sklearn.model library:*

*‘from sklearn.model\_selection import cross\_val\_score’*

*After implementing the import we began to use the cross\_val\_score using the training data ‘X’ and testing data ‘y’ along with specifically setting the cv to 10 and the scoring attribute set to accuracy. The reason we set the cv to 10 was because we checked through various values for cv and concluded that 10 was the best value for cv.*

*‘knn\_scores = cross\_val\_score(knn\_classifier, X, y, cv=10, scoring='accuracy')’*

*Furthermore, we used the mean function to get the mean accuracy across folds before printing out the outputs. We achieved this by setting the once plain accuracy to the newly embedded accuracy equaled to the mean function:*

*‘knn\_mean\_accuracy = knn\_scores.mean()’*

*Surprisingly, we discovered that cross-validation did not improve the KNN algorithm however it did need a higher k value than the previous KNN algorithm. We discovered that the best accuracy was the same, 73 percent but the k value increased from 21 to 26. What this meant was that cross-validation needed more values before capping than the previous KNN algorithm.*

*KNN with k=1 - Accuracy with Cross Validation: 0.64*

*KNN with k=6 - Accuracy with Cross Validation: 0.69*

*KNN with k=11 - Accuracy with Cross Validation: 0.7*

*KNN with k=16 - Accuracy with Cross Validation: 0.72*

*KNN with k=21 - Accuracy with Cross Validation: 0.72*

*KNN with k=26 - Accuracy with Cross Validation: 0.73*

*KNN with k=31 - Accuracy with Cross Validation: 0.73*

*Highest Accuracy: 0.73 (k=26)*

***KNN with Standard Scalared and Cross-Validation***

*In this unique case, we used prior techniques at this point to further test to see if we could improve the results. We started by using both the KNN and KNN with cross-validation but now adding one more algorithm, Standard Scalar. To implement Standard Scalar we inserted the classifier Standard Scalar and its import to use it before proceeding further:*

*‘from sklearn.preprocessing import StandardScaler’*

*‘Scalar = StandardScalar()’*

*Importantly, we then used fit\_transform on the X\_train and transformed the X\_test using standard scalar callings such as*

*‘X\_train\_scaled = scaler.fit\_transform(X\_train)’*

*‘X\_test\_scaled = scaler.transform(X\_test)’*

*With Standard Scalar finished we simply added the previous algorithms from KNN and KNN with Cross-Validation. We discovered that the results were the same however noticed that using Standard Scalared the results improved much faster. The results showed that 26 was yet again the best k value with the best percentage of 73 percent.*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=1: 0.65*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=6: 0.7*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=11: 0.72*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=16: 0.73*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=21: 0.73*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=26: 0.73*

*Mean Cross-Validation Accuracy with using Standard Scalar of KNN Classifier with k=31: 0.73*

*Highest Accuracy: 0.73 (k=26)*

***KNN classifier with Cross-Validation, Standard Scalar using Bagging and Voting***

*To finish the KNN Test we decided to use Bagging and Voting as the last add-on combination to the already made code with KNN, Cross-Validation, and Standard Scalar. With Bagging and Voting, we had to get creative and use other imports and libraries such as ‘numpy’ and ‘make\_pipeline’ to make the cross-validation adaptable enough for further use in the code. We first made sure that only X\_train was being transformed using scalar.fit\_transform and set the number of classifiers to 10 which is enough classifiers to see the pattern of improvement or failing.*

*To begin the bagging and Voting we began with creating the variables for the loop to go through the bagging and Voting. We then used bagging calling using ‘np.zeros’ using the length of the y\_train. After that, we created a for loop that represents the base classifier calls in KNN classifier with Cross-Validation and Standard Scalar:*

*‘bagging\_predictions = np.zeros(len(y\_train))’*

*With the base we created a for loop that will call the KNN classifier and used pipelining for the Standard Scalar and the KNN classifier*

*‘knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neighbors)*

*pipeline = make\_pipeline(StandardScaler(), knn\_classifier)’*

*Now we can use the cross-validation to predict the results using pipeline, scaled x\_train, and y\_train with cv as. With this, we then create the accumulation of predictions adding on prediction to the bagging\_prediction to later get the average of the accumulation of the predictions.*

*predictions = cross\_val\_predict(pipeline, X\_train\_scaled, y\_train, cv=5)*

*bagging\_predictions += predictions*

*bagging\_predictions /= num\_classifiers*

*Finally, to end the bagging section we calculated and printed the accuracies.*

*‘accuracy\_bagging = np.mean(bagging\_predictions.round() == y\_train)’*

*‘print(f"Bagging Ensemble Accuracy: {round(accuracy\_bagging, 2)}")’*

*With Bagging finished we then do the same for Voting in creating the classifier for the predictions using ‘np.zeros’ with the length of y\_train*

*‘voting\_predictions = np.zeros(len(y\_train))’*

*We did similar work for the Cross-Validation with bagging to the Voting section using pipelining and setting the params to scalared X-train with regular y\_train and cv to 5. After this, we created the accumulation of predictions and grabbed the majority voting:*

*predictions = cross\_val\_predict(pipeline, X\_train\_scaled, y\_train, cv=5)*

*voting\_predictions += predictions*

*voting\_predictions = (voting\_predictions >= (num\_classifiers / 2)).astype(int)*

*Finally, we calculate the accuracies while creating a print to show the accuracies for the voting sections. We discovered that bagging and voting gave us the same results similar to KNN with Cross-Validation and Standard Scalared with a k value of 26 having the best accuracy of 73 percent.*

*Number of Neighbors: 1*

*Bagging Ensemble Accuracy: 0.65*

*Voting Ensemble Accuracy: 0.65*

*Number of Neighbors: 6*

*Bagging Ensemble Accuracy: 0.7*

*Voting Ensemble Accuracy: 0.7*

*Number of Neighbors: 11*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Number of Neighbors: 16*

*Bagging Ensemble Accuracy: 0.73*

*Voting Ensemble Accuracy: 0.73*

*Number of Neighbors: 21*

*Bagging Ensemble Accuracy: 0.73*

*Voting Ensemble Accuracy: 0.73*

*Number of Neighbors: 26*

*Bagging Ensemble Accuracy: 0.73*

*Voting Ensemble Accuracy: 0.73*

*Number of Neighbors: 31*

*Bagging Ensemble Accuracy: 0.73*

*Voting Ensemble Accuracy: 0.73*

***Decision Tree Results:***

*For our Decision Tree results, we split the data into training and testing sets (test\_size=0.25, random\_state=40). We created a decision tree classifier with the current random state, trained the classifier on the training data, and made predictions on the testing data. With all of this, we calculated the accuracy on our testing data, and our results are seen below:*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=1: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=2: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=3: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=4: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=5: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=6: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=7: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=8: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=9: 0.74*

*Accuracy on Testing Data of Decision Tree Classifier with random\_state=10: 0.74*

*As you can see, the accuracies from all our data points with the decision tree classifier were the same so the prediction for this method is 74%.*

***Decision Tree with Cross Validation Results:***

*For decision tree with cross-validation, we used the same logic as the original test. You can see the results below:*

*Mean Accuracy of Decision Tree Classifier with random\_state=1: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=2: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=3: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=4: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=5: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=6: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=7: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=8: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=9: 0.72*

*Mean Accuracy of Decision Tree Classifier with random\_state=10: 0.72*

*As you can see, the prediction is less accurate than without Cross-validation.*

***Decision Tree Classifier Using Standard Scalar and Cross-Validation:***

*For this method, we use the same logic, but we add to it. Now we also use the following:*

*‘from sklearn.preprocessing import StandardScaler’*

*‘Scalar = StandardScalar()’*

*We initialize the standard scaler, fit and transform the scaler on the training data, and initialize variables to store the highest accuracy and random state value. After getting our results, this is what it looks like:*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=1: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=2: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=3: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=4: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=5: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=6: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=7: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=8: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=9: 0.71*

*Mean Cross-Validation Accuracy of Decision Tree Classifier with random\_state=10: 0.71*

*Our results only yielded a 71% accuracy.*

***Decision Tree with Cross-Validation and Standard Scalar using Bagging and Voting:***

*To finish the Decision Tree testing we use bagging and voting as the last few functions added to the already-made Decision Tree with cross-validation using Standard Scalar.*

*For this method, we had to initialize the Standard Scaler, fit and transform the scaler on the training data, and define the number of base classifiers(num\_classifiers=10). We had to bag the classifier and loop through each base classifier by initializing the decision tree classifier, creating a pipeline with standard scaler, using cross-validation for getting predictions on the training data, and accumulating the predictions. Then we repeat for voting the classifier. Below are our results:*

*Random State: 1*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 2*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 3*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 4*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 5*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 6*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 7*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 8*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 9*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*Random State: 10*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*As you can see, our results are lower using this method, only getting a 71% accuracy. Same as with just the standard scaler.*

***Logistic Regression Results:***

*For the Logistic Regression Results, we have defined and train splits the data using*

*‘ X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)’*

*With this in mind, we create and list of values to iterate from 100 iterating to 525 by a value of 25 and as we iterate through these values we record the accuracy of the prediction values with test data wanting the highest accuracy and the iteration that gave us the values which produce the value 0.7 with the first iteration of 100 which is than produced in every iteration from then on.*

*Accuracy on Logistic Regression Classifier with max\_iter=100: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=125: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=150: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=175: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=200: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=225: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=250: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=275: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=300: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=325: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=350: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=375: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=400: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=425: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=450: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=475: 0.7*

*Accuracy on Logistic Regression Classifier with max\_iter=500: 0.7*

*We can see with the output we have the accuracies are all the same from the lowest iteration value to the highest so with this the prediction of who got transported and who didn't is 0.7 percent.*

***Logistic Regression Results With Cross Validation:***

*For Logistic Regression with cross-validation, we used the same logic as the original logistic regression but using a higher iteration number of 200 instead due to cross-validation needing a higher value of iteration*

*maxIter\_values = list(range(200, 525, 25))*

*Using the new iteration value list to iterate through and test the mean accuracy of Logistic regression. We utilize cross\_val\_score to train the data to find the accuracy using the current iteration as the classifier. The reason we set the cv to 10 was because we checked through various values for cv and concluded that 10 was the best value for cv. Then with the information given, we calculate the accuracy score and record it*

*logistic\_scores = cross\_val\_score(lr\_classifier, X, y, cv=10, scoring='accuracy')*

*With the mean accuracy recorded we find the corresponding highest iteration and mean accuracy attached the results are*

*Logistic Regression with max\_iter=200 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=225 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=250 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=275 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=300 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=325 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=350 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=375 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=400 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=425 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=450 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=475 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=500 - Mean Accuracy: 0.71*

*We see that results give us a marginally better mean accuracy at a 0.71 rather than a 0.7 so we find that cross-validation holds a higher accuracy than a regular Logistic Regression.*

***Logistic Regression with Standard Scalared and Cross-Validation***

*In this case of standard scalar and cross-validation, we test further to so if we can improve the results given. We add to the algorithms in cross-validation testing, a standard scalar operation, we insert*

*‘from sklearn.preprocessing import StandardScaler’*

*‘Scalar = StandardScalar()’*

*By adding the function of scalar we create a pipeline that applies standard scalar and logistic regression and we use it to perform the k folds for cross-validation and accuracy score calculation*

*pipeline = make\_pipeline(StandardScaler(), lr\_classifier)*

*logistic\_scores = cross\_val\_score(pipeline, X, y, cv=10, scoring='accuracy')*

*With these functions added and changed we can perform the rest of the cross-validation as normal finding the accuracy mean for each iteration starting from 200 and we get the results.*

*Logistic Regression with max\_iter=200 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=225 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=250 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=275 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=300 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=325 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=350 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=375 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=400 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=425 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=450 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=475 - Mean Accuracy: 0.71*

*Logistic Regression with max\_iter=500 - Mean Accuracy: 0.71*

*With this we see that there is no difference in accuracy from the Logistic regression Cross Validation with or without Scalar the mean remains the same at 0.71.*

***Logistic Regression with Cross-Validation, Standard Scalar using Bagging and Voting***

*To finish the Logistic regression testing we use bagging and voting as the last few functions added to the already-made Logistic Regression with cross-validation using Standard Scalar. Bagging and Voting, we do some setup we fit and transform by re-training the X\_train using scaler and set a number classifier for the base classifiers and set it to 10*

*scaler = StandardScaler()*

*num\_classifiers = 10*

*With this set up we use the same iteration values as before till we get to creating and using bagging and voting where we create the variable for the loop to go through the bagging and voting. We use the bagging call np.zeros using y\_train after which we the rest of the logistic regression classifier with standard scalar and cross-validation*

*bagging\_predictions = np.zeros(len(y\_train))*

*We then use the logistic regression classifier and a pipeline for standard scalar*

*lr\_classifier = LogisticRegression(max\_iter=max\_iter\_value)*

*pipeline = make\_pipeline(StandardScaler(), lr\_classifier)*

*predictions = cross\_val\_predict(pipeline, X\_train\_scaled, y\_train, cv=5)*

*We then proceed to the predictions for the cross value and accumulate those predictions to the baggingh\_prediction to later average them*

*bagging\_predictions += predictions*

*bagging\_predictions /= num\_classifiers*

*accuracy\_bagging = np.mean(bagging\_predictions.round() == y\_train)*

*Then we end the bagging by calculating and printing the accuracy*

*‘accuracy\_bagging = np.mean(bagging\_predictions.round() == y\_train)’*

*‘print(f"Bagging Ensemble Accuracy: {round(accuracy\_bagging, 2)}")’*

*With Bagging finished we then do the same for Voting in creating the classifier for the predictions using ‘np.zeros’ with the length of y\_train*

*‘voting\_predictions = np.zeros(len(y\_train))’*

*We did similar work for the Cross-Validation with bagging to the Voting section using pipelining and setting the params to scalared X-train with regular y\_train and cv to 5. After this, we created the accumulation of predictions and grabbed the majority voting:*

*predictions = cross\_val\_predict(pipeline, X\_train\_scaled, y\_train, cv=5)*

*voting\_predictions += predictions*

*voting\_predictions = (voting\_predictions >= (num\_classifiers / 2)).astype(int)*

*Finally, we get these results*

*Max Iterations: 200*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 225*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 250*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 275*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 300*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 325*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 350*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 375*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 400*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 425*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 450*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 475*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*Max Iterations: 500*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*With this, we discover that we once again gain a slightly better accuracy for logistic regression at 0.72 rather than the previous 0.7 and 0.71 so the better of these three is Logistic Regression using Standard Scalar and Bagging and Voting.*

***Random Forest****:*

*Firstly, we imported the Random Forest classifier and train split for the random forest set to test\_size equal to .25 and random state to 5. This signifies that 25 percent of the data will be for testing while 85 of the rest of the data will be for training data*

*from sklearn.ensemble import RandomForestClassifier*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5) # splits*

*When it came to Random Forest we needed to adjust and measure the parameters of Random Forest which are the n\_estimators and the random state. We began to create the for loops for both the essential parameters to illustrate the accuracies with various values. In this case, we made the estimators in the range of 1 to 50 by 10 with the random state range from 1 to 6*

*for n\_estimators in range(1, 50, 10):*

*for random\_state in range(1, 6):*

*Once we finished the for loop and created the fits of x\_train and y\_train, the prediction of our X\_test, and the accuracy for the y\_test and y\_predict; we created the print statement for the accuracies of each combination. After that similar to what we did for prior algorithms we made an if statement to check if the current model has a better accuracy than the others.*

*if score > best\_accuracy:*

*best\_accuracy = score*

*best\_params = {'n\_estimators': n\_estimators, 'random\_state': random\_state}*

*We made sure that the print statement after the first for loop was finished we created a print statement that will print the best accuracy with its respective parameters and their values.*

*n\_estimators=1, random\_state= 1, Accuracy: 0.71*

*n\_estimators=1, random\_state= 2, Accuracy: 0.7*

*n\_estimators=1, random\_state= 3, Accuracy: 0.7*

*n\_estimators=1, random\_state= 4, Accuracy: 0.7*

*n\_estimators=1, random\_state= 5, Accuracy: 0.7*

*n\_estimators=11, random\_state= 1, Accuracy: 0.71*

*n\_estimators=11, random\_state= 2, Accuracy: 0.71*

*n\_estimators=11, random\_state= 3, Accuracy: 0.71*

*n\_estimators=11, random\_state= 4, Accuracy: 0.71*

*n\_estimators=11, random\_state= 5, Accuracy: 0.71*

*n\_estimators=21, random\_state= 1, Accuracy: 0.71*

*n\_estimators=21, random\_state= 2, Accuracy: 0.71*

*n\_estimators=21, random\_state= 3, Accuracy: 0.71*

*n\_estimators=21, random\_state= 4, Accuracy: 0.71*

*n\_estimators=21, random\_state= 5, Accuracy: 0.71*

*n\_estimators=31, random\_state= 1, Accuracy: 0.71*

*n\_estimators=31, random\_state= 2, Accuracy: 0.71*

*n\_estimators=31, random\_state= 3, Accuracy: 0.71*

*n\_estimators=31, random\_state= 4, Accuracy: 0.71*

*n\_estimators=31, random\_state= 5, Accuracy: 0.71*

*n\_estimators=41, random\_state= 1, Accuracy: 0.71*

*n\_estimators=41, random\_state= 2, Accuracy: 0.71*

*n\_estimators=41, random\_state= 3, Accuracy: 0.71*

*n\_estimators=41, random\_state= 4, Accuracy: 0.71*

*n\_estimators=41, random\_state= 5, Accuracy: 0.71*

*We discovered that the accuracy scores for the estimators with 1 and a random state higher than 1 would be less accurate than the first estimator with the random state 1. For n\_estimators 11 we concluded that beyond this n\_estimator there was no further increase nor decrease in accuracy. However, the best possible value was n\_estimators 21 with random\_state 5 finished with an accuracy of 71 percent although every other estimator and random state received similar results. It is safe that 71 percent is the highest random forest can go.*

***Random Forest with Cross-Validation***

*With Cross-Validation included in our Random Forest, we mirrored the prior formatting of the first Random Forest but added Cross-validation by using the necessary import to retrieve the cross-validation score. With the import, we simply continued with the double for loop.*

*from sklearn.model\_selection import cross\_val\_score, train\_test\_split*

*my\_RandomForest = RandomForestClassifier(n\_estimators=n\_estimators, bootstrap=True, random\_state=random\_state)*

*After this is done, we continue in creating the actual cross-validation sequence using the necessary cross-validation function inerting the random forest variable with the X\_train and y \_train. Furthermore, we gathered all the mean accuracies across folds and then printed out the variables with their respective accuracies.*

*cv\_scores = cross\_val\_score(my\_RandomForest, X\_train, y\_train)*

*mean\_score = cv\_scores.mean()*

*print(f'n\_estimators={n\_estimators}, random\_state={random\_state}, Mean Accuracy: {round(mean\_score, 2)}')*

*Similar to what we did for the first Random Forest code we implemented an if statement in which if the placed variable mean score is greater than the best accuracy, then make the best accuracy equal to the mean score. With this, we can print the values for n\_estimators and their respective random states.*

*if mean\_score > best\_accuracy:*

*best\_accuracy = mean\_score*

*best\_params = {'n\_estimators': n\_estimators, 'random\_state': random\_state}*

*Once we retrieve the output for all the values for the n\_estimators and random states with their accuracies, we finally print out the best accuracy along with its respective n\_estimator and random\_state. We discovered that the best accuracy was 73 percent with the values of n\_estimaor of 41 and random state of 3.*

*n\_estimators=1, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=1, random\_state=2, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=3, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=4, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=5, Mean Accuracy: 0.7*

*n\_estimators=11, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=3, Mean Accuracy: 0.73*

*n\_estimators=21, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=3, Mean Accuracy: 0.73*

*n\_estimators=41, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=5, Mean Accuracy: 0.72*

*Best Mean Accuracy: 0.73 with n\_estimators=41 and random\_state=3*

*However, we also encountered that there was another accuracy that matched the chosen accuracy but with different values. We decided to accept both the combos because they shared the same accuracy and 73 percent is the confirmed highest possible accuracy score.*

***Random Forest using Standard Scalar and Cross-Validation***

*To include standard scalar we simply implemented the import necessary to use standard scalar functions and libraries. Once we inserted the necessary import we immediately fitted the X\_train creating the scaled X\_train.*

*from sklearn.ensemble import RandomForestClassifier*

*scaler = StandardScaler()*

*X\_train\_scaled = scaler.fit\_transform(X\_train)*

*We copied over the code we did prior for the Random Forest and only changed what was being cross-validated. Before with the Random Forest with Cross-Validation, we had X-train as the second parameter but now we are using the scaled version of the X\_train.*

*cv\_scores = cross\_val\_score(my\_RandomForest, X\_train\_scaled, y\_train)*

*Just like prior we continued with the code and set the mean accuracy across folds then printed the accuracy for each combination of n\_estimators and random\_states. The if statement for printing all combinations and their respective accuracies is the same as the previous random forest code. Finally, we print out the best combination with the highest possible accuracy after the loop is done. Unfortunately, we discovered that the results were lower than previous results for Random Forests.*

*n\_estimators=1, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=1, random\_state=2, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=3, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=4, Mean Accuracy: 0.71*

*n\_estimators=1, random\_state=5, Mean Accuracy: 0.7*

*n\_estimators=11, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=11, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=21, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=31, random\_state=5, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=1, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=2, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=3, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=4, Mean Accuracy: 0.72*

*n\_estimators=41, random\_state=5, Mean Accuracy: 0.72*

*Best Mean Accuracy: 0.72 with n\_estimators=41 and random\_state=3*

*The accuracies came to a halt once the estimator was higher than 11, if lower than 11, the accuracy got worse. We concluded that 72 percent was the highest accuracy that can be achieved using this version of Random Forest.*

***Random Forest with Cross-Validation, Standard Scalared, and using Bagging and Voting***

*To achieve this version of Random Forest we first created the imports for pipelining and numpy. We then started to insert the base classifier section of the bagging method via a for loop.*

*from sklearn.pipeline import make\_pipeline*

*import numpy as np*

*Before the loop we created the bagging classifier by manually using the zeros function with the length of the y-train. With this, we then create the loop that will run through each base classifier.*

*bagging\_predictions = np.zeros(len(y\_train))*

*for \_ in range(num\_classifiers):*

*rf\_classifier = RandomForestClassifier(n\_estimators=n\_estimators\_value, bootstrap=True, random\_state=random\_state\_value)*

*Later we use cross\_val\_predict for getting the predictions on the training data. We do this by using the random Forest classifier with the inputs of the X\_train and y\_train scaled. We then accumulate the predictions and ensemble predictions by averaging. We take the bagging predictions and divide them by the number of classifiers.*

*predictions = cross\_val\_predict(rf\_classifier, X\_train\_scaled, y\_train, cv=5)*

*bagging\_predictions += predictions*

*bagging\_predictions /= num\_classifiers*

*Now, we calculate the accuracy using the mean functions with the bagging predictions rounded equal to y\_train. We then print the Bagging ensemble accuracy.*

*accuracy\_bagging = np.mean(bagging\_predictions.round() == y\_train)*

*print(f"Bagging Ensemble Accuracy: {round(accuracy\_bagging, 2)}")*

*With this we finished bagging and moved on the voting; with voting, we did what we did for bagging in the beginning using the voting classifier manually with the zeros function. After we made the voting classifier, similar to Bagging we created the cross-val prediction using the random forest classifier with x and y train scaled.*

*voting\_predictions = np.zeros(len(y\_train))*

*predictions = cross\_val\_predict(rf\_classifier, X\_train\_scaled, y\_train, cv=5)*

*We then create the accumulation of the predictions and ensemble them by majority voting. By doing the ensembling we check if the number of positive votes is greater than or equal to half of the total number of classifiers.*

*voting\_predictions += predictions*

*voting\_predictions = (voting\_predictions >= (num\_classifiers / 2)).astype(int)*

*We finally calculate the accuracy using the mean function and print out the rounded accuracy at the end of the whole process.*

*accuracy\_voting = np.mean(voting\_predictions == y\_train)*

*print(f"Voting Ensemble Accuracy: {round(accuracy\_voting, 2)}")*

*We concluded that 72 percent was the maximum this version of the algorithm can achieve.*

*n\_estimators: 1, random\_state: 1*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 1, random\_state: 2*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*n\_estimators: 1, random\_state: 3*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*n\_estimators: 1, random\_state: 4*

*Bagging Ensemble Accuracy: 0.71*

*Voting Ensemble Accuracy: 0.71*

*n\_estimators: 1, random\_state: 5*

*Bagging Ensemble Accuracy: 0.7*

*Voting Ensemble Accuracy: 0.7*

*n\_estimators: 11, random\_state: 1*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 11, random\_state: 2*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 11, random\_state: 3*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 11, random\_state: 4*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 11, random\_state: 5*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 21, random\_state: 1*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 21, random\_state: 2*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 21, random\_state: 3*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 21, random\_state: 4*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

*n\_estimators: 21, random\_state: 5*

*Bagging Ensemble Accuracy: 0.72*

*Voting Ensemble Accuracy: 0.72*

***Linear Regression:***

*To begin our Linear Regression algorithm we first implemented the necessary imports to access the functions to make the algorithm work. Once we implemented the imports we created a list of values starting with values for the for loop to iterate from.*

*from sklearn.linear\_model import LinearRegression*

*import matplotlib.pyplot as plt*

*lin\_reg\_values = list(range(1, 11))*

*lowest\_mse = float('inf')*

*best\_max = 0*

*Just like with previous algorithms, we made a list in a range from 1 to 11. Furthermore, we created a positive infinity and a best max measurement variable to ensure that the variables are ready to track the lowest MSE and its parameter value.*

*mse = mean\_squared\_error(y\_test, y\_prediction)*

*print(f"MSE on Testing Data of Linear Regression={j}: {round(mse, 2)}")*

*if mse < lowest\_mse:*

*lowest\_mse = mse*

*best\_max = j*

**When it comes to Linear Regression it captures the MSE which to get the MSE for all values iterating through the loop, we captured the mean squared after the prediction of each value using y\_test and y\_prediction. Later we made an if statement that will grab the best result that the selected value made.**

*MSE on Testing Data of Linear Regression=1: 0.19*

*MSE on Testing Data of Linear Regression=2: 0.19*

*MSE on Testing Data of Linear Regression=3: 0.19*

*MSE on Testing Data of Linear Regression=4: 0.19*

*MSE on Testing Data of Linear Regression=5: 0.19*

*MSE on Testing Data of Linear Regression=6: 0.19*

*MSE on Testing Data of Linear Regression=7: 0.19*

*MSE on Testing Data of Linear Regression=8: 0.19*

*MSE on Testing Data of Linear Regression=9: 0.19*

*MSE on Testing Data of Linear Regression=10: 0.19*

*Lowest MSE: 0.19 for value=1*

**Unexpectedly we noticed that the MSE was very low which meant the algorithm was very successful in regards to getting the true values. With this in mind, this may be the best algorithm we had through the testing portion of the project.**

***Linear Regression with Cross-Validation***

***Adding Cross-Validation we simply imported the necessary libraries and instead of a small range list we needed to create a much bigger list to gain better accuracy, with that we decided to make the list in the range from 200 to 525 by 25.***

*from sklearn.model\_selection import cross\_val\_score*

*from sklearn.linear\_model import LinearRegression*

*import numpy as np*

*maxIter\_values = list(range(200, 525, 25))*

***We then added the cross-validation score using linear regression variable***

***classifier and X, y with cs = 10 and scoring set to negative mean squared error. Since Linear regression is set to get positive values we made cross val score negative to preserve the goal of minimizing the mean square error. After, we create a code that will calculate the mean squared error across folds.***

linreg\_errors = -cross\_val\_score(lr\_model, X, y, cv=10, scoring='neg\_mean\_squared\_error')

linreg\_mean\_error = np.mean(linreg\_errors)

**Once we finish that part of the code we simply make a code that will print the MSE and then right after create an if statement to capture the best value with the best results in the list.**

*Linear Regression with max\_iter=200 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=225 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=250 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=275 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=300 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=325 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=350 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=375 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=400 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=425 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=450 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=475 - Mean Squared Error: 0.19*

*Linear Regression with max\_iter=500 - Mean Squared Error: 0.19*

*Lowest Mean Squared Error: 0.19 (max\_iter=200)*

***After retirevigning the output the best possible value with its respective MSE was 200 an MSE of .19. With this score it is tied with the previous Linear Regression algorithm.***

***Linear Regression with Standard Scalar and Cross Validation***

***Adding Standard Scalar we simply copied previous techniques in which we scared the x train data using fit transform. The imports were also implemented for the necessary libraries to use standard scalar especially.***

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

*X\_scaled = scaler.fit\_transform(X)*

***We copied over the code we used prior for the Linear regression and changed what is being cross-validated. We change the r2 score cross\_val\_score by changing the X scaled***

*r2\_scores = cross\_val\_score(my\_linreg, X\_scaled, y, cv=5, scoring='r2')*

***`***

***Just like the previous code we find to take the mean of the cross-validated r2 scores and print the r2 score from each iteration then print out the current highest r2 score from each iteration taking the highest one.***

*Mean R2 Score on Cross-Validation Data of Linear Regression=1: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=2: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=3: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=4: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=5: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=6: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=7: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=8: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=9: 0.23*

*Mean R2 Score on Cross-Validation Data of Linear Regression=10: 0.23*

*Highest Mean R2 Score: 0.23 for max\_iter=1*

***The results we get from this show that the highest mean score comes from the first iteration and is maintained through all 10 iterations. The highest accuracy is 23 percent using this method of Linear Regression***

***Linear Regression with Cross-Validation, Standard Scalared, and using Bagging and Voting***

***Using this version of Linear Regression we import the pipelining and numpy***

*from sklearn.pipeline import make\_pipeline*

*import numpy as np*

***We create the bagging classifier before the loop using zeros function with the length of the y\_train, with this we can iterate through the loop that will run each base classifier***

*bagging\_predictions = np.zeros(len(y))*

***We then use cross\_val\_predict to get the predictions for the training data, using linear regressing we scale the X as X\_scaled we then accumulate the predictions after which we divide by the number of classifiers***

*predictions = cross\_val\_predict(my\_linreg, X\_scaled, y, cv=5)*

*bagging\_predictions += predictions*

*bagging\_predictions /= num\_classifiers*

***We then calculate the R2 score by using the mean of the baggining predictions subtracting the y\_train in the equation we then print the bagging ensemble accuracy***

*r2\_score = np.mean((bagging\_predictions - y) \*\* 2)*

*print(f"\nMax Iterations: {j}")*

*print(f"Bagging Ensemble Accuracy: {round(r2\_score, 2)}")*

***With this finished bagging we move to voting as we implement the voting classifier manually with with zeros function we create the cross\_val\_prediction using linear regression with the X-scaled***

*voting\_predictions = np.zeros(len(y))*

*predictions = cross\_val\_predict(my\_linreg, X\_scaled, y, cv=5)*

***We then calculate the voting accumulation from the predictions added***

*predictions = cross\_val\_predict(my\_linreg, X\_scaled, y, cv=5)*

*voting\_predictions /= len(lin\_reg\_values)*

***We then calculate the accuracy using the mean function and print out the R2 score for voting.***

*r2\_score\_voting = np.mean((voting\_predictions - y) \*\* 2)*

*print(f"\nMax Iterations: {round(r2\_score\_voting, 2)}")*

*print(f"Voting Ensemble Accuracy: {round(r2\_score\_voting, 2)}")*

*print(f"\nHighest Mean R2 Score(Bagging):{round (highest\_r2\_score\_bagging ,2)} for max\_iter={best\_max\_bagging}")*

***We then iterate through all instances and print out the highest mean R2 score***

*Max Iterations: 1*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 2*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 3*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 4*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 5*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 6*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 7*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 8*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 9*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 10*

*Bagging Ensemble Accuracy: 0.19*

*Max Iterations: 0.19*

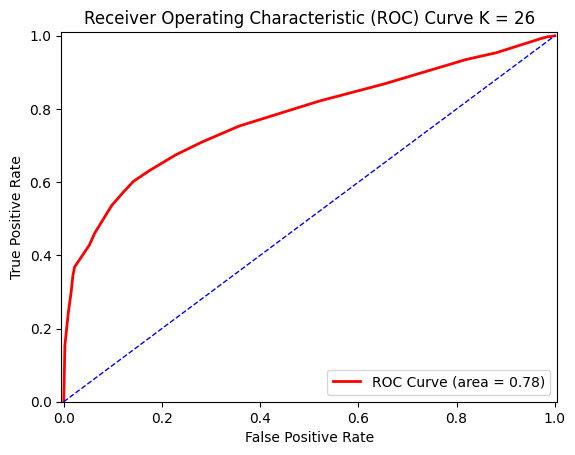
*Voting Ensemble Accuracy: 0.19*

*Highest Mean R2 Score (Bagging): 0.19 for max\_iter=1*

***The results we get from this show that the highest mean score comes from the first iteration and is maintained through all 10 iterations. The highest accuracy is 19 percent using this method of Linear Regression which is less than the previous version we used.***

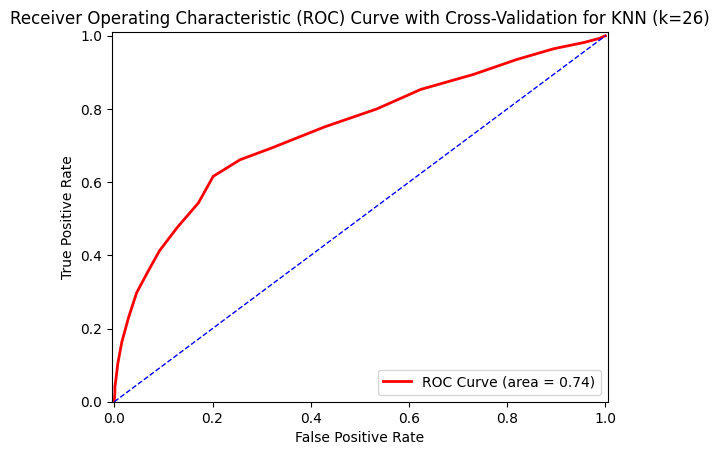
**Curve Performances:**

**KNN ROC:**

****

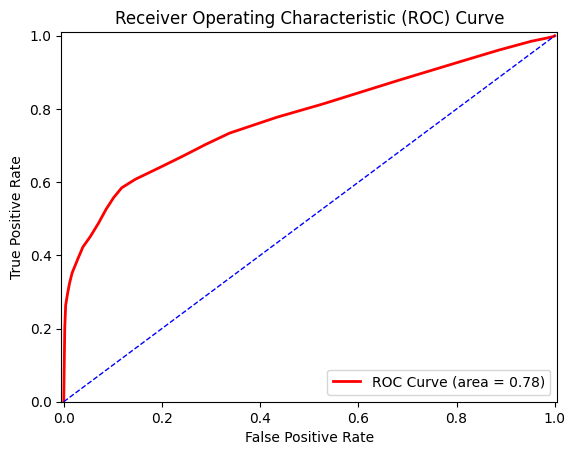
Description: In this ROC curve analysis we set K to 26 by initializing the train to make predictions in the testing. The ROC helps visualize the trade-off between true positive and false positive rates at different thresholds, while the AUC score summarizes the overall performance of the classifier.

**KNN with Cross Validation ROC:**

****

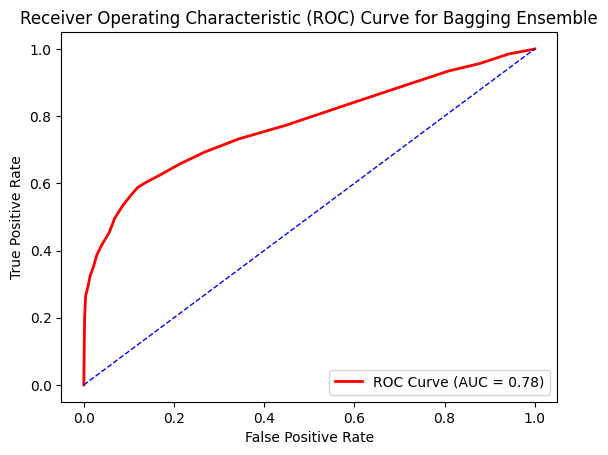
Description: Set k to 26, initialize and train KKN classifier to use Cross-Validation to receive prediction then use that for each fold to calculate ROC curve.

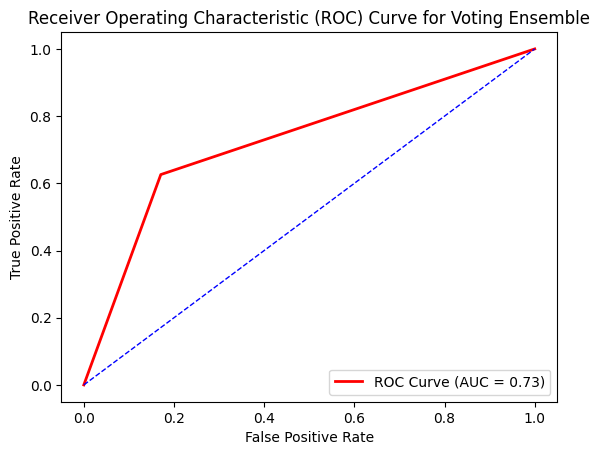
**KNN using Standard Scalar and Cross-Validation ROC:**

****

Description: We set K value to 26, then with using the pipeline function to get each prediction fold with Cross-Validation and Standard Scalar. Calculating the ROC and AUC curve with the classifier predictions by using Matplotlib.

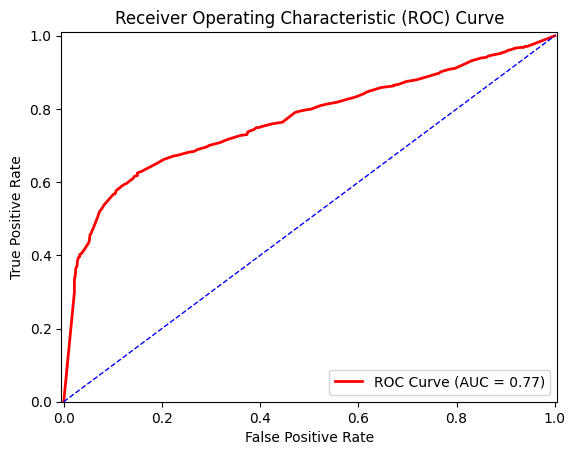
**KNN Classifier with Cross-Validation and Standard Scalar using Bagging and Voting ROC:**

****

****

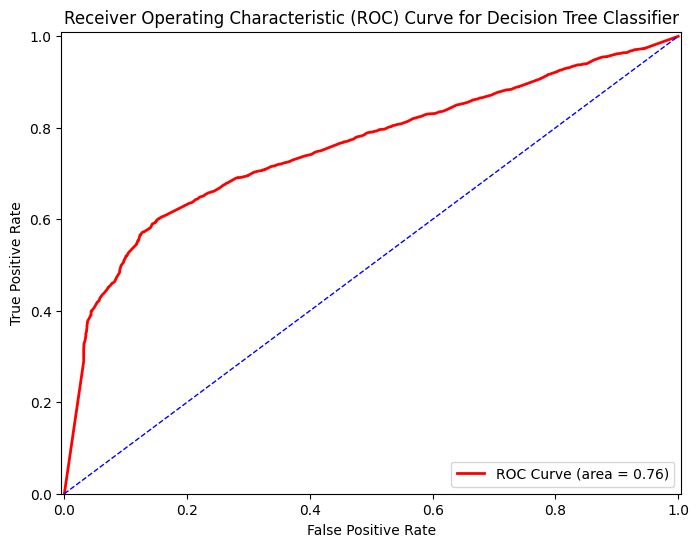
Description: Use a list of 1 - 31 for every 5 which identify for the K values then use that for the number of neighbors to create an ROC and AUC curve for both Bagging and Voting.

**Decision Tree Classifier ROC:**

****

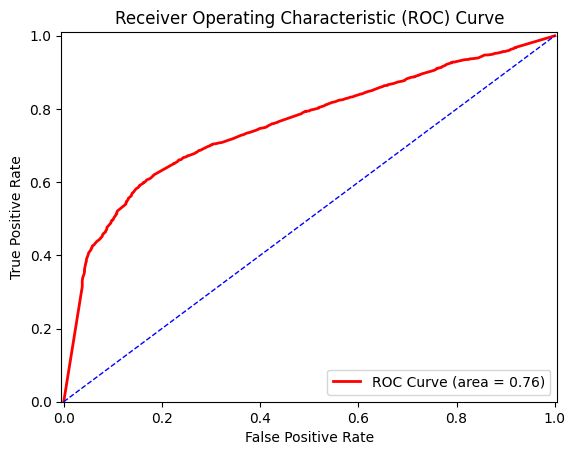
Description: ROC curve represents the trade-off between true positive rate and false positive rate for different thresholds of the Decision Tree Classifier. The AUC value provides a quantitative measure of the classifier’s ability to discriminate between positive and negative instances.

**Decision Tree Classifier with Cross-Validation ROC:**

****

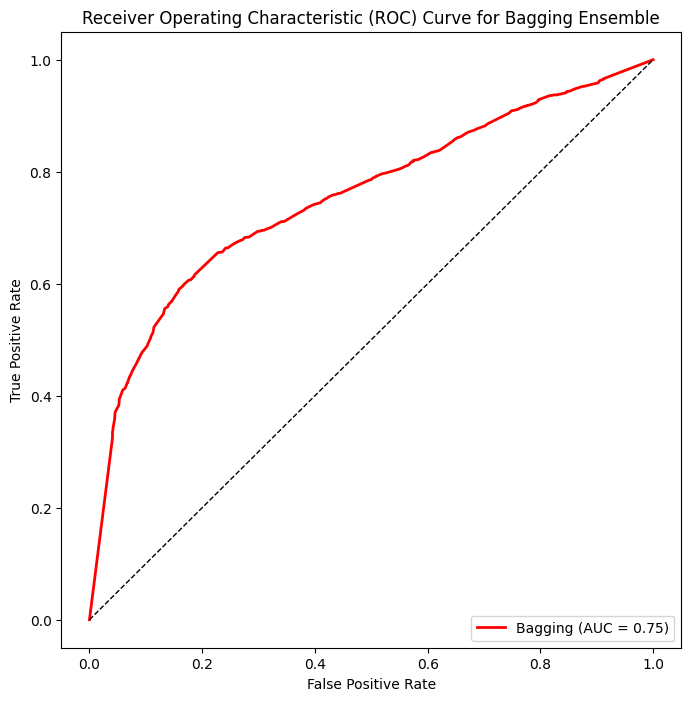
Description: Decision Tree Classifier with a random state of 3 and a Cross-validation value of 10 to calculate the mean accuracy which is then used to calculate the ROC and AUC.

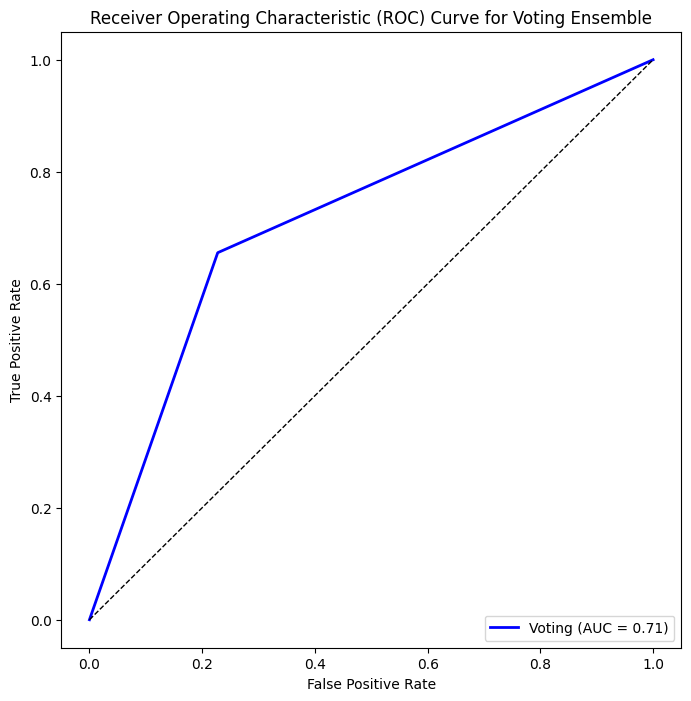
**Decision Tree Classifier with Cross-Validation and Standard Scalar ROC**

****

Description: Hyperparameter with the use of Cross-validation with iterated random state of 1-11 then using pipeline that comes with Standard Scalar. Giving the ROC curve with the random state value with its mean accuracy.

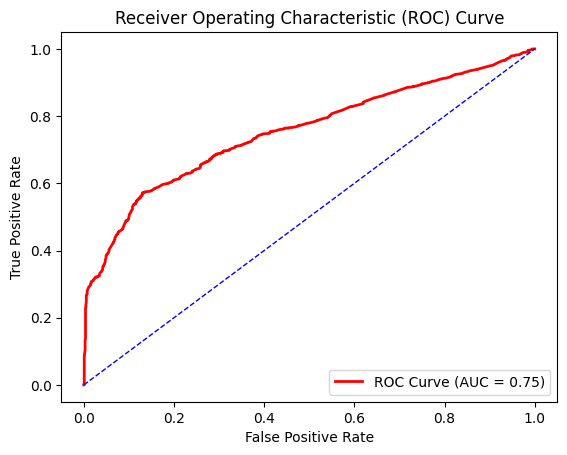
**Decision Tree Classifier with Cross-Validation and Standard Scalar using Bagging and Voting ROC:**

****

****

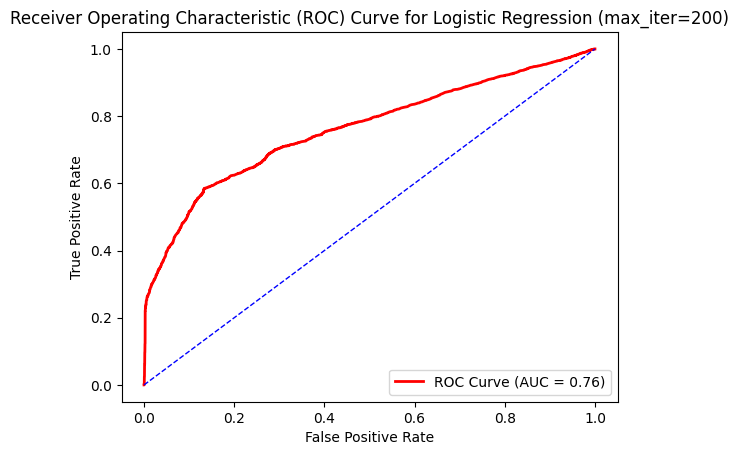
Description: Unlike the last one that iterates with pipeline, this time it iterates over Bagging and Voting with the random states being from 1 - 11. Standard Scalar is still used to fit and transform the training data but uses the result of accuracy for the AUC values in order to predict the ROC curve.

**Logistic Regression Classifier ROC:**

****

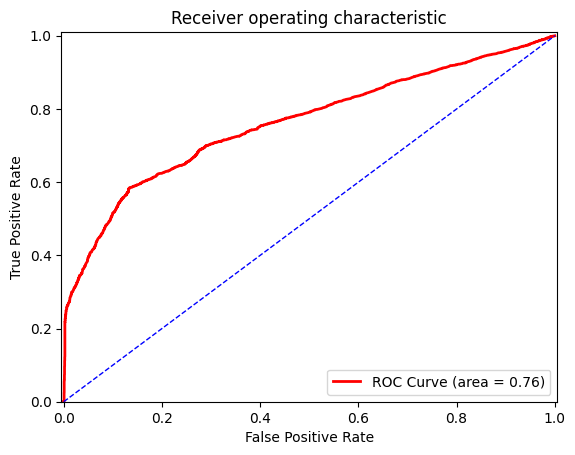
Description: Using the max iteration of a set list of values for the training data and using that to evaluate the accuracy of the test data. In this case, set max\_iteration to 100 to calculate the accuracy of the ROC curve. ROC Curve came to 0.75.

**Logistic Regression Classifier with Cross-Validation ROC:**

****

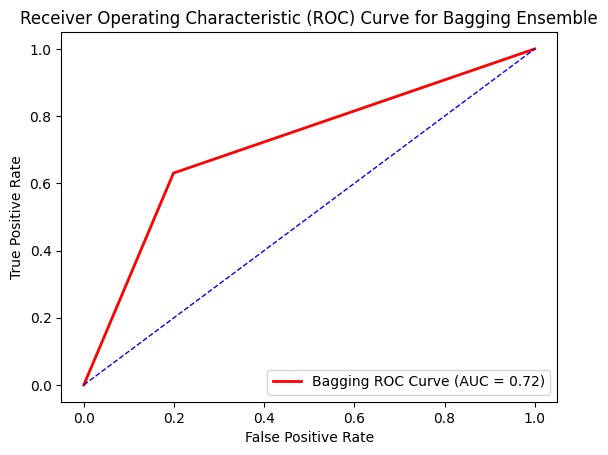
Description: Using Cross\_Validation for the whole dataset to predict the probability and to calculate the AUC when max\_iteration was set to 200. Predictions are made on the test data when the classifier is done on training data.

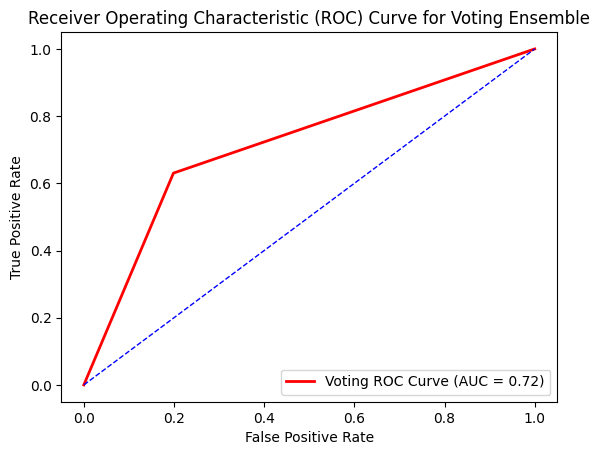
**Logistic Regression Classifier with Standard Scalar and Cross-Validation ROC:**

****

Description: Iterates through the list of values and pipeline using Standard Scalar then use Cross-Validation to calculate accuracy score to evaluate the mean accuracy for each max\_iteration. In this case, max\_iteration of 200 to build pipeline and ROC.

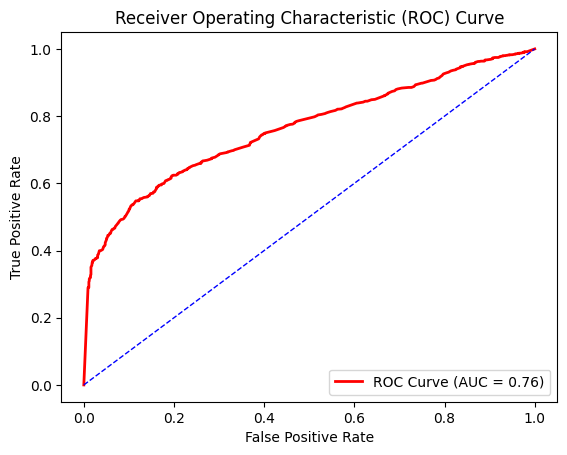
**Logistic Regression Classifier with Cross-Validation and Standard Scalar using Bagging and Voting ROC:**

****

**` **

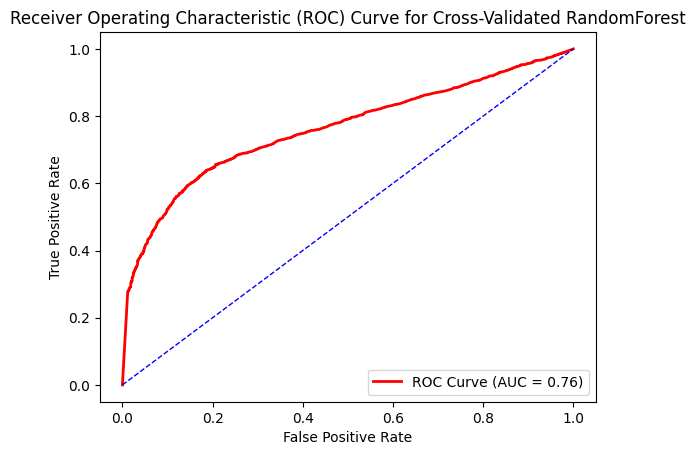
Description: Used all data to initialize the Standard Scalar for the training data to use Bagging and Voting for max\_iteration being 200. Iterates each base classifier for each prediction to calculate the accuracy for the AUC of Bagging and Voting.

**Random Forest ROC:**

****

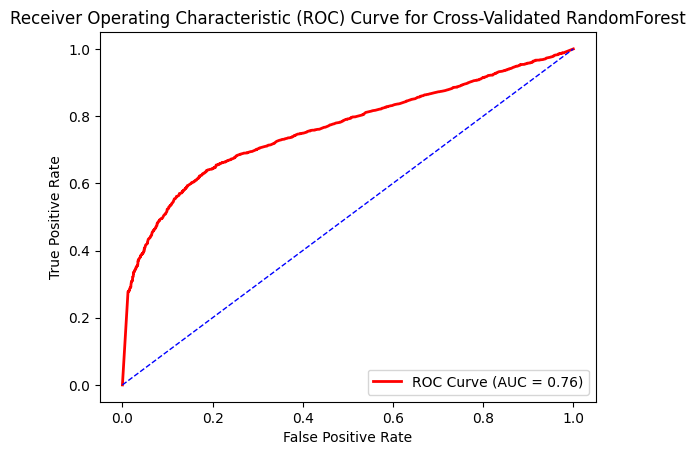
Description: With a random state of 5 and ‘n’ estimator of 21, creates and fits the classifier and then makes the predictions to calculate the accuracy for the AUC score to plot the ROC.

**Random Forest with Cross-Validation ROC:**

****

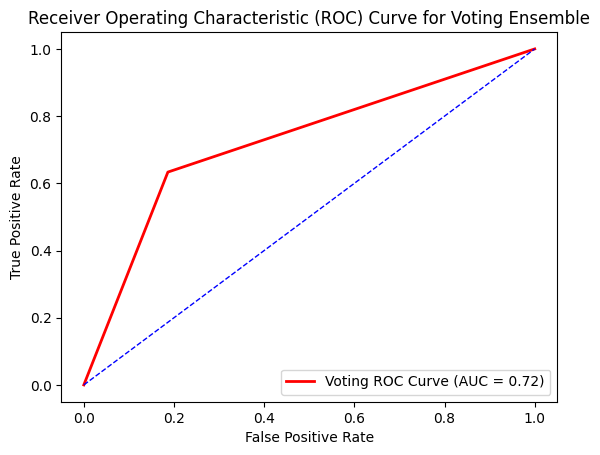
Description: We set the n\_estimator with a value of 41 and a random state of 3. With this, Cross-Validation will be used to get the mean across folds and accuracy for the specified combinations and will use that to calculate AUC and plot the ROC.

**Random Forest using Standard Scalar and Cross-Validation ROC:**

****

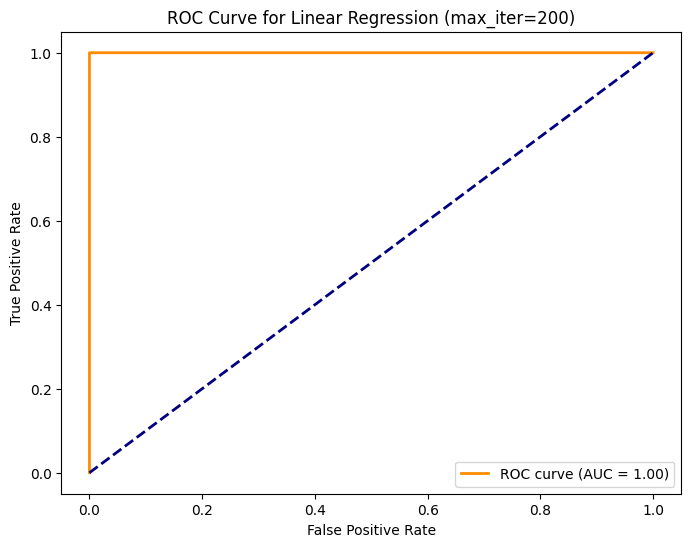
Description: Using the ‘n\_estimator\_value’ for 41 and ‘random\_state\_value’ for 3, we create and fit the classifier and use the Cross\_Validation for the predicted probabilities and accuracy scores. By doing this, we then calculate the mean accuracy across folds and combinations. Then calculate AUC and plot ROC.

**Random Forest with Cross-Validation and Standard Scalar using Bagging and Voting ROC:**

****

Description: First initialize the Standard Scalar in order to fit and transform to the training data. We set the base classifiers to 10, ‘n\_estimator\_value’ to 11, and ‘random\_state\_value’ to 2 then use Bagging and Voting with Cross-Validation to get predictions on the training data for both ensembles and use the Bagging and Voting ensemble for AUC and ROC of the ensembles.

**Linear Regression with Cross Validation, Standard Validation, AUC, FPR, and TPR into ROC:**

****

Description: Let max\_iterations to 200 and used Standard Scalar for scale features, Cross-Validation for predictions, and depending on the median we used it to binarize the prediction which in the end, was not a good outcome. Overall, not great in results compared to other ROC/AUC but included anyway.

**Note ROC: Could not do ROC/AUC curve for Linear Regression besides using it with standard and cross-validation ROC**

**Final Results:**

After all of our testing, we found that the range of accuracies that we got was mostly in the low 70s for percentage. This means that most of our methods have a solid chance of actually finding the missing passengers. Here is our resulting data**(Note: This does not include the MSE of linear regression)**:

l

***Highest accuracy:*** When reviewing our data, I found that the highest accuracy was the Decision Tree Classifier Method. The Decision Tree Classifier accuracy score was 74%, which is the highest standard accuracy score that we got in all of our tests. This is without cross-validation or Standard scaler. When the Decision Tree Classifier is running with Standard Scalar and/or cross-validation, the accuracy score drops to 72% or 71%.

***Lowest Accuracy:*** The lowest accuracy we found from all our testing was found in Logistic Regression. The Logistic Regression Classifier had an accuracy of only 70%, which doesn’t sound like a big downgrade from 74%, which was our highest, but since we’re dealing with people’s lives, 4% makes a big difference. Unlike with the Decision Tree Classifier where the accuracy score dropped when we ran it with cross-validation and standard scalar, the Logistic Regression Classifier went up by 1 or 2% when we ran it with cross-validation and standard scalar.

***Other Results:*** We used a variety of methods to check what had the highest and lowest accuracy scores. We already know what the highest and lowest accuracy scores are as we discussed earlier, but here are the other methods and accuracy scores that we got:

* ***KNN:*** KNN was the first method we started testing, and unlike some of the other methods, where the accuracy score was the same for all the different classifiers, KNN had a variety of accuracy scores ranging from 65 to 73%. This is why even if KNN had the highest accuracy score, we would still be reluctant to choose it as our method for finding the passengers because 65% is much lower than anything else we tested. The same goes for when we ran the data with cross-validation and standard scalar. The accuracy score range even dropped to a minimum of 64% and the maximum stayed at 73%.
* ***Linear Regression:*** When we were testing for linear regression, we found that it was different from the other classifiers in the sense that we tested the MSE instead of an accuracy score. The MSE we got for linear regression was 0.19, which is pretty good, but we’re not 100% confident about how this compares to the other tests we did, so we didn’t want to use it as our highest or lowest method. We didn’t think it would be wise or safe to use linear regression when we’re not too familiar with our results.
* ***Random Forest:*** The last testing method we used was Random Forest, which cultivated an accuracy score of 71%. This is our second-weakest testing method, only beating logistic regression. With cross-validation, Random Forest fluctuated between 71 and 73 %, which is an improvement. With Standard Scalar, only up to 72%.

With this information, we can decide when we’re looking for missing passengers across the universe, we should use the Decision Tree Classifier as that has the highest accuracy score, and thus, the highest chance of success at finding the missing passengers.

**Sources linked:**

**Data**

* <https://www.kaggle.com/competitions/spaceship-titanic>

**Techniques**

* <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>
* <https://stephenallwright.com/cross_val_score-sklearn/>