Our strategy was that we would utilize RandomForestRegressor and, primarily, several neural networks to help predict the outputs of the test data. We also wanted to see how well we could utilize feature selection to improve our models by way of maintaining consistent accuracy as well as improving efficiency, memory used, and model runtimes. We tested several different configurations of neural networks primarily by testing different regularizers: BatchNormalization, which can have a regularizing effect, Dropout layers, L1 regularizers / Lasso regression, and Elastic net regression. We also tested out two different optimizers - primarily Adam, and SGD - and used two different activation functions - ReLu and Leaky ReLu. We also utilized two methods of preprocessing the data - Label encoding and encoding via a Pipeline.

In our initial attempts to preprocess the data, we label-encoded our categorical data (i.e. the data with an 'object' data type) and then proceeded to apply a standard scaler to all of our data. This was something we didn't realize at first wouldn't work as we'd need to account for unseen categories in the test data. We went forward with our testing without realizing this error.

We also tried feature selection considering the data set was quite large and unknown. There were some categorical features along with several continuous and binary. We chose ReliefF for our feature reduction as it's considered to perform better with such data. Also, because of class imbalance in target values, gave us another reason to try ReliefF. We selected the top 125 features as ranked by ReliefF as the significance of the rest of the features were dropping after that. We then tested the features we selected by fitting a RandomForestRegressor to our newly encoded data. When using the full dataset, we had a low MSE and an R^2 of about 0.30. The model only captured about 30% of the variance in the output. This wasn't a very useful model and the R^2 decreased while the MSE increased when we utilized only the most important features. The R^2 decreased to about 12%. However, reducing the number of features we used reduced the time taken to fit the model to about 20% of the original time taken. Although reducing features did positively affect training time, it didn't seem to help much on accuracy of model predictions.

We then created a number of neural networks. These should've been better at capturing the non-linear relationships in our data. What we saw was that the neural network models made with ReLu and ADAM, whether or not we added regularizers, consistently returned models with an accuracy of about 83-84%. The major differences we'd noticed were how the validation loss and validation accuracies seemed to be changing. In the models without L1 and elastic net regularization, the validation loss might initially decrease during the first few epochs of training but would then consistently start to increase. In addition, the validation accuracy would become sporadic, increasing and decreasing with regularity. The models were becoming less stable and showing signs of overfitting.

The presence of L1 and elastic net regularizers made the validation loss curve start to decrease consistently and even reach convergence but validation accuracy at that point would peak at an early epoch and hardly change. It was indicative that, while error was decreasing and so was overfitting, our model really wasn't learning. A fact we realized late was that there was a

class imbalance in the outputs. We compared our neural network results with a Decision Tree classifier that used our label-encoded data. We got the same accuracy as our neural networks did as it mainly returned 1s. To us, this was another sign of the class imbalance. To try to address this imbalance, we tried to make a neural network model that utilized SMOTE resampling to help oversample the less represented output, 0. Unfortunately, our validation accuracy dropped to less than 80%. The resampling was only partially successful and we changed gears with how we addressed the data by revisiting our preprocessing method.

We used a new method of preprocessing the data by utilizing one-hot encoding for our object class data and assigning different scalers to different columns with a ColumnTransformer and the Pipeline class. This was slightly helpful. Our representation of 0s from our neural network outputs improved and so did our test accuracy. All further tests were conducted using the pipeline's preprocessed data. We had relatively similar results with our loss and accuracy curves when utilizing the previous neural network setups with our pipeline data, with slightly more learning occurring in those models and slightly higher test accuracies. We then shifted gears and utilized SGD to no significant improvement. Adam optimizer appeared to be doing just fine with similar efficiency and performance. Our best success occurred when we utilized a simple feedforward network with a Leaky ReLu activation function. The validation accuracy was above 85% even if the loss indicated there was overfitting.

We also analyzed one of our model's behaviors in the context of which features were significant in affecting the decision. A SHAP plot indicated that "subject" & "phase" were among the top features having a significant role in the model's predictions. One more thing we learnt that SHAP did not work properly with scaled data after using StandardScaler(). We tried LIME also with RandomForestRegressor and our Basic FNN model but LIME requires a probability score method from a model to work which these models do not provide as directly available.