Anabel Costa
Andrew Hennessey
Mila Lewis-Peinado
Aaron Shikh

Preliminary Project Report

Abstract

This research illustrates the use and structure of Logistic Regression. We first discuss Miami International Airport (MIA), which is the focus of our research, and then visualize its flight data set. We then construct a main effects model and use the Area Under Curve (AUC) to quantify the model's prediction accuracy of flight delays. Expanding on this, we also created larger models with polynomial and interaction terms, alongside variable selection methods like stepwise and lasso regression. Finally for the airport data, we compared the AUC of each model and chose which one is best suited to predict flight delays. Following this, we explored simulated data in two ways. First, we simulated the data ourselves to try to maximize or minimize the AUC of a given model. Second, we reversed this process and created a logistic regression model with a given simulated data set to achieve the same goal as previously mentioned. Finally, we created a function and shiny app to consolidate the steps used in this research to make logistic regression an even more accessible and efficient process.

Introduction and Background

When Statisticians and Data Scientists are tasked with predicting a binary outcome, they have a handful of classification methods at their disposal. Of these, Logistic Regression is best at maintaining interpretability as well as accessibility. For this reason it is incredibly useful for real world applications, which asked us to investigate an issue in everyday life: flight delays.

Flight delays are a common inconvenience for passengers and a significant challenge for airlines. Understanding the factors that contribute to these delays is crucial for improving both customer satisfaction and operational efficiency in the aviation industry. The key focus in this research was Miami International Airport (MIA). It is the second busiest airport in the United States for international passengers, and offers more flights to Latin America and the Caribbean than any other U.S. airport (About Us, c2024). Additionally, it generates $118 billion annually and contributes about 700,000 jobs to the local economy. Given its contributions to U.S. aviation and economy, it would be of great use to diagnose what contributes to flight delays at MIA. Using Logistic regression, as well as variable selection methods like stepwise and lasso regression, we set out to find the largest contributing factors to flight delays and to construct a model best suited for predicting the outcome.

Once analysis is completed for the airport data, it is demonstrated what logistic regression is capable of accomplishing. Moving beyond this, it was in our interest to peel back another layer and understand the "how" of logistic regression. In further exploring the inner workings of a statistical method, it is often of great use to simulate data. By stripping away labels or units, there are no preconceptions of what to expect based on real life experience. Instead, the understanding of the method at hand becomes the driving force behind decision making. Therefore, we've demonstrated multiple exercises in simulation, particularly achieving low and high predictive rates, in order to put logistic regression on full display.

Finally, during the course of this research many of the steps taken had to be repeated in order to achieve the results. For a non-statistician, reapplying the methods used here may be daunting or simply undesirable for this reason. To make our research repeatable or translatable into completely new work, we wanted to create a way to streamline the process of a complete logistic regression analysis. Therefore we created a function and shiny app to do exactly this, and hope that it offers those who are new to logistic regression a practical and useful way of doing so.

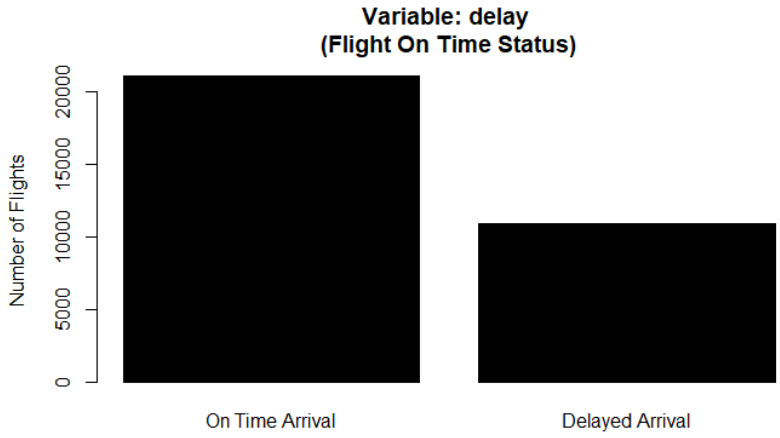
Methods and Results

Data Visualization

The specific data set from MIA that we considered contains flight information from June to September of 2023, and is composed of 11 predictors as well as the response variable, which is a binary outcome of whether or not the arrival of the flight was delayed by 15 minutes or more. The predictors include information on flight duration, departure time and origin, day of the week, month of the flight and airline carrier.

The first step towards understanding the MIA data was by simply visualizing the data in order to detect trends and patterns in the data. Keeping in mind the ultimate goal of predicting delays, we wanted specifically to create plots showing delay and other predictors' relationship with it.

Fig. 1



The first graph in Figure 1 shows a comparison between delayed and on-time arrival flights into Miami International Airport, not differentiating between the amount of delayed time, or any possible causes for the delays. The bars for each category (on-time and delayed flights) show the frequency of flights falling under these two groups. Out of approximately 32,000 flights in the dataset, 22,000 of them were on time, and 10,000 were delayed. This makes 31.25% of all arrivals into MIA delayed flights.

Fig. 2

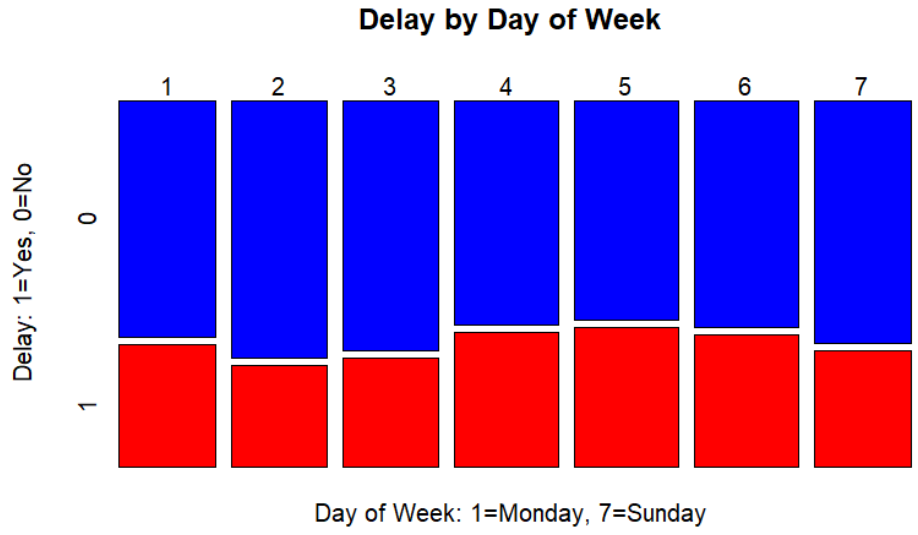


Figure 2 depicts the number of delays relative to the total number of flights visually, with the red shaded regions being delayed flight arrivals, while the blue denotes non-delayed flight arrivals. Each bar represents a different day of the week with Monday being the first bar, Tuesday being the second bar, all the way down to Sunday being the seventh bar. Figure 2 shows

that there are generally more delays on Thursdays, Fridays, and Saturdays, with the most number of delays being on Fridays.

Fig. 3

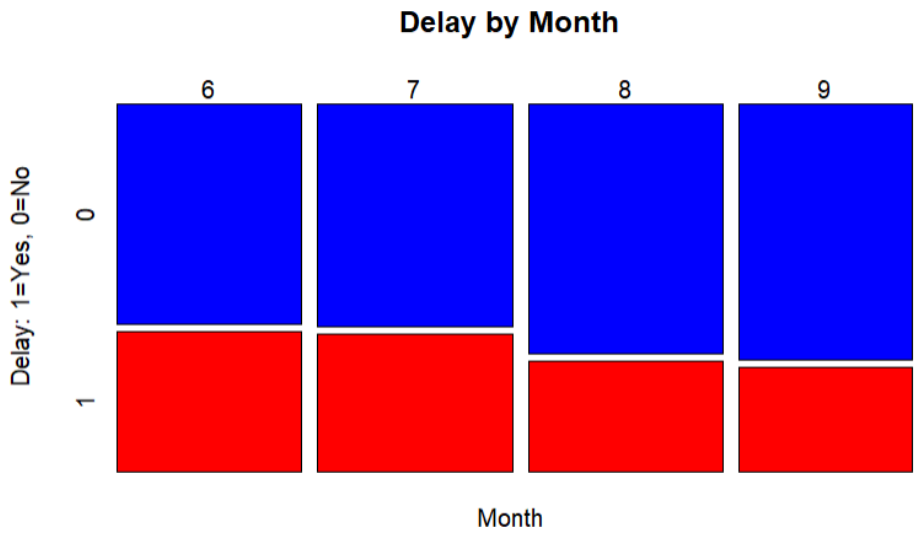


Figure 3 depicts the number of delays relative to the total number of flights visually, with the red shaded regions being delayed flight arrivals, while the blue denotes non-delayed flight arrivals. Each bar represents a different month with June being the first bar (labeled 6), July being the second (labeled 7), August being the third (labeled 8), and September being the fourth (labeled 9). In Figure 3, we can clearly see that there are generally more delays relative to total flights in June and July, with the fewest delays relative to total flights in September.

Fig. 4

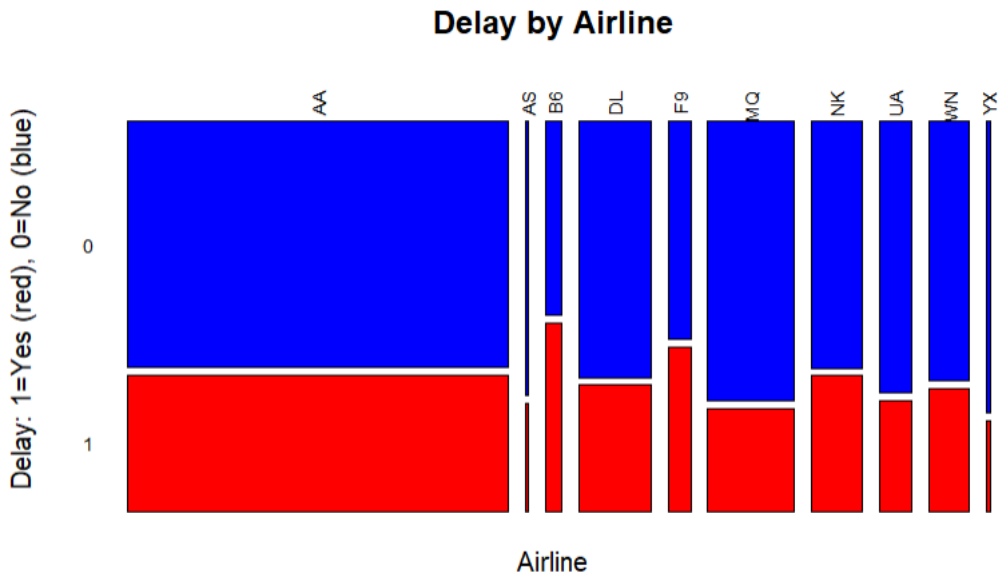
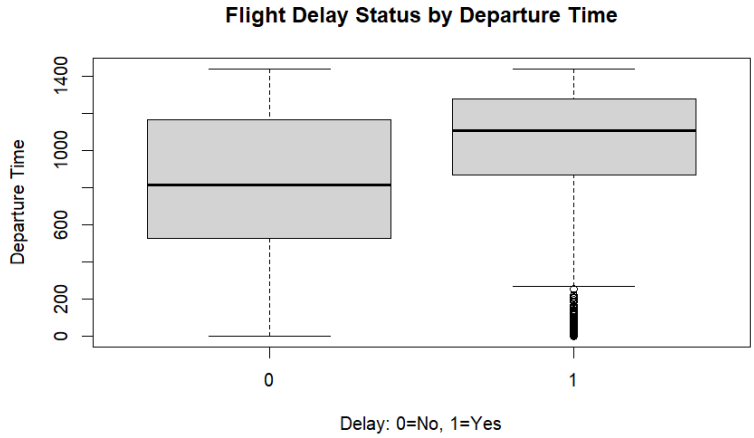


Figure 4 depicts the number of delays relative for a given carrier relative to the total number of flights. Each bar represents a different airline with the width being thicker for airlines with more flights. In Figure 4, we can clearly see that there are more total flights in AA (American Airlines), DL (Delta), and MQ (Envoy Air). One notable aspect of that chart is that American Airlines has far more flights than any of the other airlines, but has a relatively low number of delays despite this fact. Another notable aspect of this chart is that while B6 (Jetblue) has fewer flights at MIA than other airlines, it has the highest number of delays relative to its total number of flights.

Fig. 5

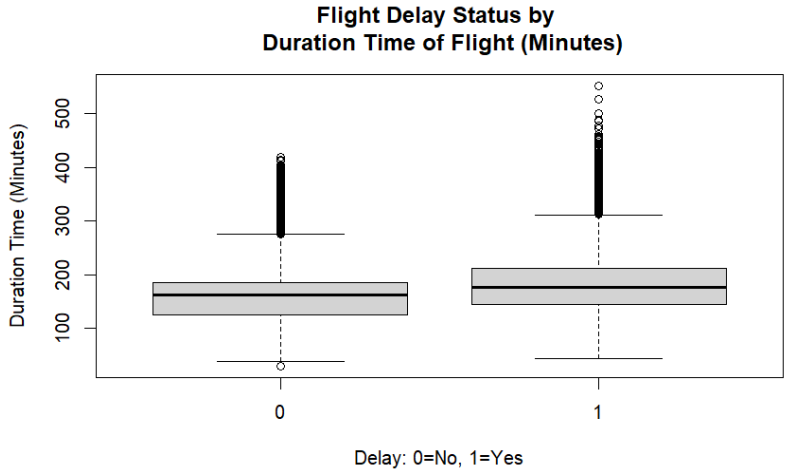


In Figure 5 there are two boxplots: the one on the left corresponds to flights that were not delayed, and on the right ones that were. Comparing the two, it is apparent that departure indeed has an effect on delay status. The median departure time for delayed flights is roughly 300

minutes (5 hours) greater than non-delayed ones. Not only this, but the first quartile of departure times for delayed flights even exceeds the median for non-delayed. Finally, while there is more variability in the departure time for on-time flights, the box is narrower for delayed ones, demonstrating that this effect is reliably significant.

To understand the significance of the delay status versus departure time relationship, we deemed it helpful to compare it to another relationship:

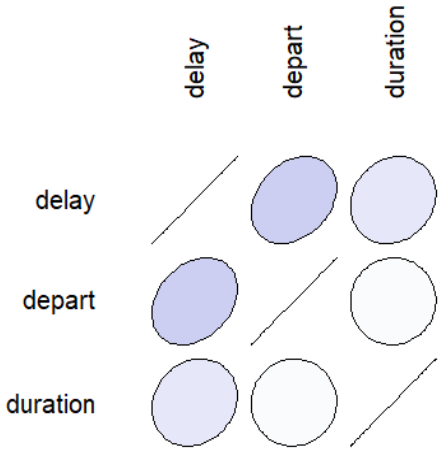
Fig. 6



In Figure 6**,** we once again have two boxplots for flights that weren't delayed and ones that were. However, this time the y–axis is the duration time of flights in minutes. As can be seen, the boxplots look fairly similar. The medians for both are quite near, and the height of the boxes reveal that their variabilities are almost the same. This then illustrates that duration time does not have a significant effect on flight delay status, and when compared to the previous plot, it reinforces the strength of the departure time relationship.

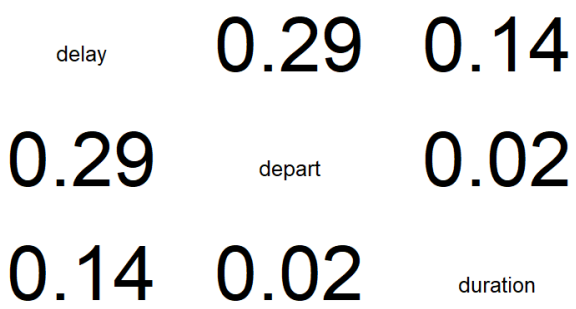
To further compare how departure time and duration time may have an effect on flight delay status, we created a correlogram. When quantifying a relationship between two variables, correlation can determine the strength and direction of said relationship. What a correlogram will then do is visualize correlations between multiple variables at once.

Fig. 7



The correlogram in Figure 7 depicts the correlations between delay status, departure time, and duration time. The darker and more "squished" an oval is, the higher its correlation and thus the stronger its relationship. Shown above, the correlation is highest for delay status and departure time with its dark blue, oblong ovals. The duration time versus delay status shows a rather weak relationship, while duration time versus departure time shows little to no relationship at all. To supplement this plot, we also generated a correlogram using numbers rather than ovals to display the actual correlations:

Fig. 8



So, the correlation for delay status and departure time is a little over double that of the correlation between delay status and flight duration. These correlograms reinforce what we inferred from the boxplots, and thus emphasize the impact of departure time on flight delays.

Main Effects Model & AUC

The next step in predicting flight delays for the MIA data was running a simple logistic regression between day, carrier, depart, duration, and month to predict whether or not there was a delay in arrival. Logistic regression works by passing a linear combination of regressors through the sigmoid function, which maps the input vector to a probability bounded between 0 and 1,

where the coefficients for the linear combination of the input vector is found through maximum likelihood estimation (Casella et al. 2002)**.** Each coefficient or beta in the regression represents the amount of change to the log odds ratio through a one unit change in the regressor, holding all other regressors constant. The log odds ratio is a measure of association between the regressor and the predictor.

To evaluate the performance of our logistic regression model, we used the AUC (the area under the ROC curve) which is a value between 0 and 1 that summarizes the overall ability of the classifier to discriminate between the two classes. An ROC curve is a graphical representation of the performance of a logistic regression classifier as its discrimination threshold is varied. In logistic regression we observe a probability that is between 0 and 1, and we can choose a discrimination threshold, which is the probability that will determine whether we pick a positive or negative class (delayed or not delayed in our case). For any given threshold, the classifier will produce two types of correct classifications and two types of errors in the confusion matrix:

1. True Positives (TP): Correctly classified positive instances.
2. False Positives (FP): Negative instances incorrectly classified as positive.
3. True Negatives (TN): Correctly classified negative instances.
4. False Negatives (FN): Positive instances incorrectly classified as negative.

Using these values from our confusion matrix, we can find the true positive rate and the false positive rate which are key components for constructing an ROC curve. The true positive rate is the proportion of positive instances that are correctly classified, providing a general sense of how well the model is identifying positive outcomes. False positive rate, on the contrary, is the proportion of negatives that were misclassified as positives. The formulas for these two metrics are shown below:

1. True Positive Rate (TPR) (Sensitivity): TP/(TP+FN)
2. False Positive Rate (FPR) (1 - Specificity): FP/(FP+TN)

The ROC curve plots the TPR (y-axis) against the FPR (x-axis) for varying threshold values. As you vary this threshold, the classifier becomes more or less strict in its predictions, altering the rates of true and false positives. The curve gives insight into the trade-off between true positives and false positives. This leads to the interpretation of an AUC, where an AUC of 1 represents a perfect classifier, while an AUC of 0.5 indicates a performance no better than randomly guessing. The AUC can also be interpreted as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

For our main effects logistic regression model of the MIA data, which included day, carrier, depart, duration, and month, we obtained the following results:

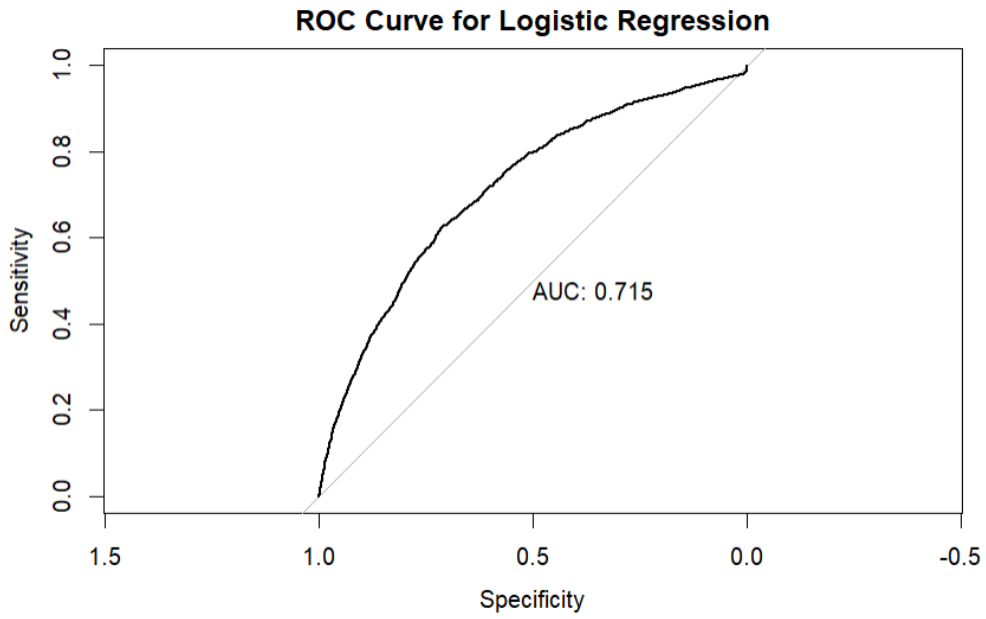
Fig. 9

```
Call:
glm(formula = delay ~ day + carrier + depart + duration + month,
    family = binomial("logit"), data = train.batch)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.250e+00  7.875e-02 -41.272  < 2e-16 ***
day2        -3.305e-01  5.473e-02  -6.040 1.54e-09 ***
day3        -2.149e-01  5.398e-02  -3.982 6.84e-05 ***
day4         1.253e-01  5.203e-02   2.409 0.015980 *
day5         1.830e-01  5.183e-02   3.530 0.000415 ***
day6         1.242e-01  5.209e-02   2.385 0.017072 *
day7        -1.333e-01  5.338e-02  -2.498 0.012505 *
carrierAS   -1.466e+00  2.356e-01  -6.223 4.87e-10 ***
carrierB6    5.243e-01  9.504e-02   5.517 3.45e-08 ***
carrierDL    2.100e-01  4.846e-02   4.333 1.47e-05 ***
carrierF9    7.279e-01  8.044e-02   9.050  < 2e-16 ***
carrierMQ   -3.001e-01  4.794e-02  -6.260 3.84e-10 ***
carrierNK    1.323e-01  5.549e-02   2.384 0.017146 *
carrierUA   -1.313e-01  7.047e-02  -1.863 0.062399 .
carrierWN    2.896e-01  6.299e-02   4.598 4.26e-06 ***
carrierYX   -7.538e-01  2.246e-01  -3.357 0.000789 ***
depart       2.175e-03  4.727e-05  46.018  < 2e-16 ***
duration     4.325e-03  2.234e-04  19.362  < 2e-16 ***
month7      -1.007e-02  3.871e-02  -0.260 0.794764
month8      -3.425e-01  3.947e-02  -8.676  < 2e-16 ***
month9      -4.399e-01  4.105e-02 -10.716  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

As seen in Figure 9, the first column lists each variable, as well as each factor level for qualitative predictors (i.e. day2, day3, etc.). Each column to the right then corresponds to each predictor. The estimate column is generally important to measure the estimated β’s, or regression coefficients, in order to understand the strength and direction of a variable’s relationship. However, for the purposes of deciding on a model, the Pr(>|z|) column, or p-values, is most important in understanding statistical significance and model fit. The significance codes at the bottom are a guide to whether a predictor is statistically significant or not. If a p-value is less than 0.05, then the predictor corresponding to it will be deemed statistically significant. Therefore, most of the predictors in this model are significant. However, with a lack of asterisks in the rows for ‘carrierUA’ and ‘month7’, it can be seen that their p-values are greater than 0.05: 0.062399 and 0.794764 respectively. Therefore, United Airlines and July are not statistically significant in this model in comparison to their baseline categories.

We first used this model on the testing data to predict whether a flight is delayed or not, and then the results obtained were compared to actual outcomes. From this, the ROC and AUC were generated:

Fig. 10



As seen from Figure 10, the AUC is 0.715. This value indicates this model is far from perfectly incorrect, and being above 0.5 shows it is better than a model simply making guesses.

Stepwise Regression

The next step for creating a strong model was to try and improve the AUC by using stepwise regression. Stepwise regression is a statistical technique used to help choose which predictors (independent variables) to include in a regression model. With this technique, you add and remove variables so that you can focus on the ones that have a stronger impact on your outcome variable. There are two different approaches to stepwise regression. The first one is called backwards stepwise regression. Backward stepwise regression starts with a full regression model that has all of the predictor variables. Then, you check the significance of each predictor and their impact on the outcome variable and remove ones that don't seem important. At the end, you should be left with predictors that are statistically significant (Sperandei, 2014). Forward

stepwise regression on the other hand, starts off with a completely empty regression model, and one by one predictors are added to the model based on their statistical significance.

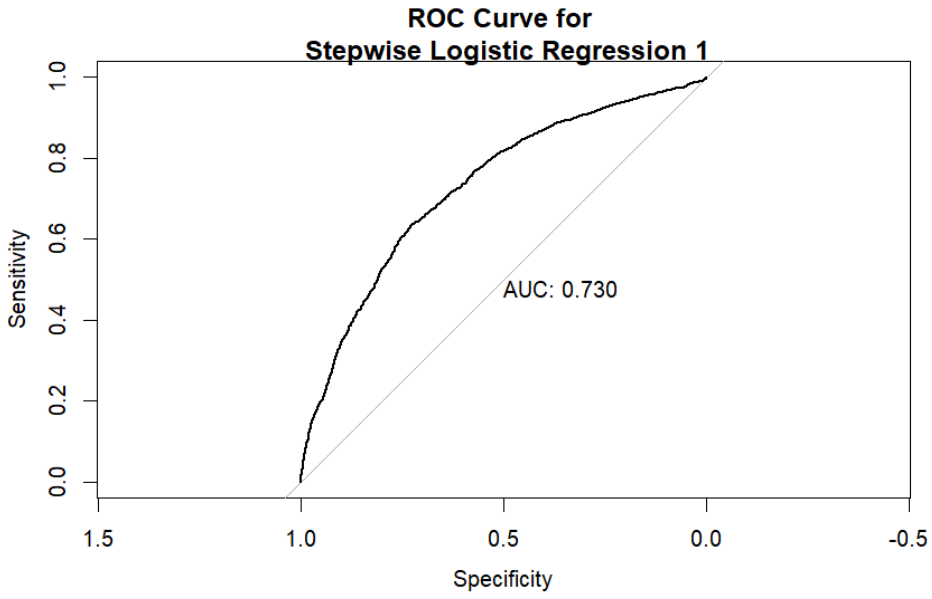
The first model we created using backward stepwise regression included the main effects as well as pairwise interactions between 'day', 'depart', 'duration', and 'month'. Using backward selection on the full model, it was computed that "duration:month" alone should be removed to minimize AIC. Below are the estimates for just statistically significant terms in this model.

Fig. 11

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-2.641880e+00	2.183876e-01	-12.097206	1.092666e-33
day3	-9.967930e-01	2.555682e-01	-3.900302	9.607294e-05
depart	1.523639e-03	1.851403e-04	8.229645	1.877696e-16
duration	2.525892e-03	8.459324e-04	2.985926	2.827209e-03
month8	-5.219139e-01	1.655182e-01	-3.153212	1.614844e-03
carrierAS	-1.502888e+00	2.307591e-01	-6.512803	7.376129e-11
carrierB6	4.848570e-01	9.391533e-02	5.162703	2.434089e-07
carrierDL	2.278860e-01	4.871638e-02	4.677811	2.899537e-06
carrierF9	6.351672e-01	8.170594e-02	7.773820	7.615420e-15
carrierMQ	-3.225881e-01	4.900913e-02	-6.582203	4.635284e-11
carrierNK	1.200209e-01	5.579772e-02	2.150999	3.147626e-02
carrierWN	2.729323e-01	6.293774e-02	4.336544	1.447407e-05
carrierYX	-7.425881e-01	2.273863e-01	-3.265756	1.091723e-03
day4:depart	7.222334e-04	1.695225e-04	4.260397	2.040638e-05
day2:duration	1.645626e-03	8.014386e-04	2.053340	4.003965e-02
day3:duration	2.013765e-03	8.072092e-04	2.494725	1.260549e-02
day2:month7	-3.488982e-01	1.507953e-01	-2.313720	2.068307e-02
day4:month7	-8.883106e-01	1.460399e-01	-6.082656	1.182081e-09
day5:month7	3.456846e-01	1.419008e-01	2.436100	1.484659e-02
day6:month7	-4.521733e-01	1.416949e-01	-3.191176	1.416952e-03
day3:month8	7.252298e-01	1.538707e-01	4.713242	2.438068e-06
day4:month8	3.465149e-01	1.468944e-01	2.358939	1.832729e-02
day5:month8	4.520447e-01	1.495299e-01	3.023105	2.501958e-03
day6:month8	-6.751085e-01	1.565011e-01	-4.313763	1.604986e-05
day6:month9	-6.401618e-01	1.520623e-01	-4.209865	2.555230e-05
depart:duration	1.543799e-06	6.671055e-07	2.314176	2.065809e-02
depart:month7	4.022886e-04	1.264942e-04	3.180294	1.471258e-03

As we can see in Figure 11, many of the categorical main effect terms are no longer significant with the inclusion of interaction terms. However, 'depart' and 'duration' as main effects are still significant in this model. Overall, we gain more information about how predictors within the model interact. To understand the predicting strength of the model, we generated an ROC curve.

Fig. 12



While the AUC of 0.730 was an improvement over the main effects model, which had an AUC of 0.715, the next step was to add polynomial regressors.

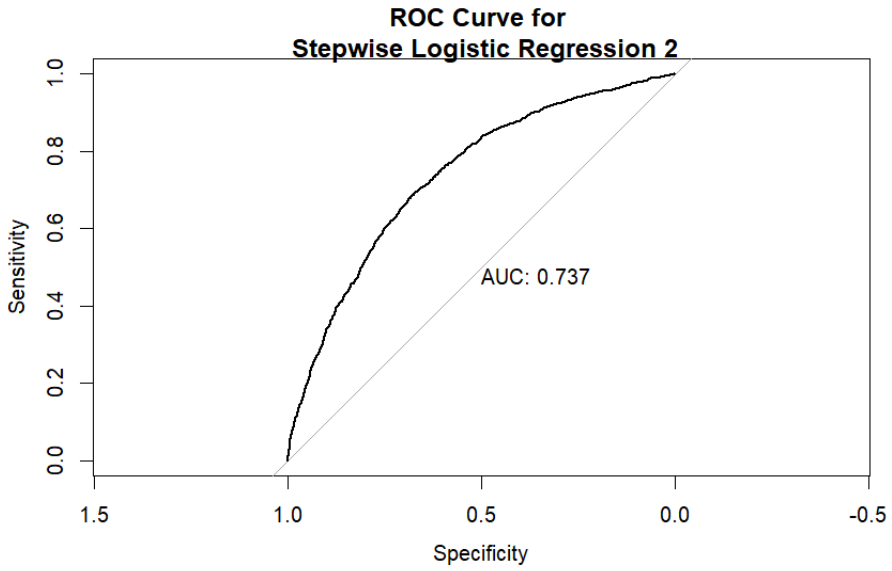
The following model includes the main effects, pairwise interaction terms, and now second and third-degree polynomial terms for 'depart' and 'duration'. Prior to fitting however, backwards selection was used and 'depart:duration' was removed to minimize the AIC. Below are the estimates for the significant terms:

Fig. 13

	Estimate	Std. Error	z value	Pr(>|z|)
day3	-8.069588e-01	2.598228e-01	-3.105804	1.897624e-03
depart	-1.482133e-02	6.651467e-04	-22.282792	5.426754e-110
duration	1.293984e-02	2.808454e-03	4.607459	4.076199e-06
carrierAS	-1.224320e+00	2.513835e-01	-4.870327	1.114134e-06
carrierB6	4.433268e-01	9.449685e-02	4.691445	2.712819e-06
carrierDL	2.862139e-01	4.996229e-02	5.728598	1.012640e-08
carrierF9	4.517707e-01	8.255557e-02	5.472322	4.441761e-08
carrierMQ	-2.733020e-01	5.030125e-02	-5.433305	5.531982e-08
carrierWN	2.618617e-01	6.437325e-02	4.067865	4.744577e-05
carrierYX	-7.560201e-01	2.329959e-01	-3.244778	1.175422e-03
poly(depart, degree = 3, raw = TRUE)[, 2:3]2	2.037848e-05	8.679526e-07	23.478788	6.719023e-122
poly(depart, degree = 3, raw = TRUE)[, 2:3]3	-7.517201e-09	3.562124e-10	-21.103142	7.442231e-99
day2:depart	-3.722887e-04	1.746033e-04	-2.132198	3.299061e-02
day4:depart	6.339908e-04	1.680199e-04	3.773307	1.610978e-04
day2:duration	1.878009e-03	8.235451e-04	2.280395	2.258425e-02
day3:duration	2.275565e-03	8.269221e-04	2.751849	5.925976e-03
day2:month7	-3.125990e-01	1.541040e-01	-2.028493	4.250993e-02
day4:month7	-8.929024e-01	1.492913e-01	-5.980942	2.218511e-09
day5:month7	3.598779e-01	1.450973e-01	2.480252	1.312897e-02
day6:month7	-3.907908e-01	1.452249e-01	-2.690936	7.125178e-03
day3:month8	7.378140e-01	1.567813e-01	4.706007	2.526157e-06
day4:month8	3.083402e-01	1.501737e-01	2.053224	4.005085e-02
day5:month8	4.302246e-01	1.525097e-01	2.820965	4.787937e-03
day6:month8	-6.671357e-01	1.599796e-01	-4.170131	3.044252e-05
day6:month9	-5.843626e-01	1.556247e-01	-3.754949	1.733769e-04
depart:month7	2.506623e-04	1.263728e-04	1.983514	4.731000e-02
duration:month7	-1.439756e-03	5.997565e-04	-2.400567	1.636970e-02
duration:month8	-1.993472e-03	6.096491e-04	-3.269869	1.075975e-03

Comparing the significance of terms from the first stepwise model, many of the same interaction terms remained. In addition, both the quadratic and cubic terms for 'depart' were significant, which gives insight into the added benefit of polynomial terms. To see if this model was an improvement for predicting delays, another ROC curve was generated:

Fig. 14



Improving from an AUC of 0.730 of the last model to 0.737 for this model, it is apparent that adding polynomial terms has increased our predictive power for delays. Yet, there is still more that could even be added upon this model, and that is three-term interactions.

A further extension of the pairwise interactions previously mentioned are the three-term interactions. Rather than observing how just two predictors interacting with each other have an effect on delays, we will now do so with three predictors interacting. Thus the third and final model for stepwise logistic regression will include all the previously mentioned terms, but with three-way interactions for 'day', 'depart', 'duration', and 'month'. Using backward selection on

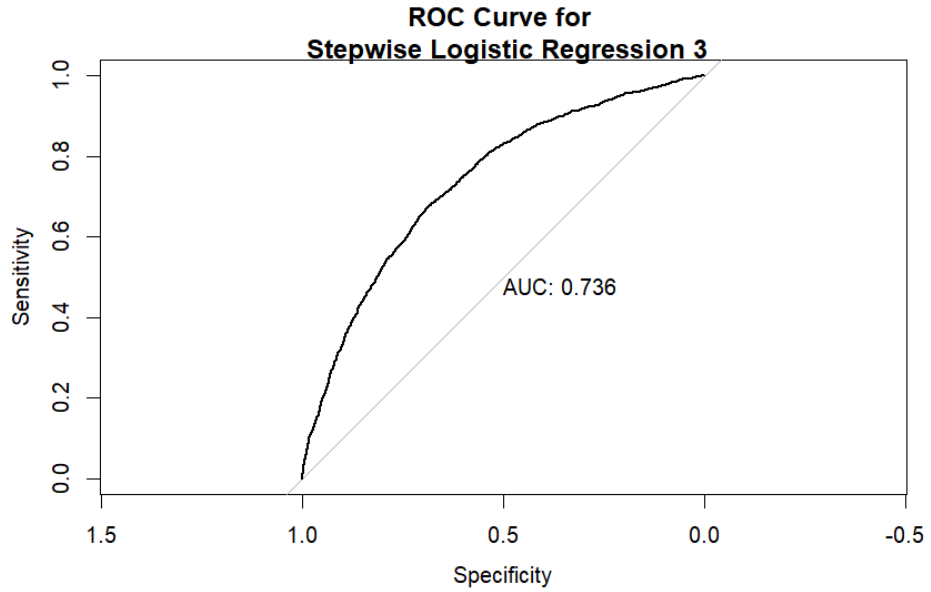
the full model, ‘day:depart:duration’ and ‘depart:duration:month’ are removed to minimize the AIC. Shown below are the significant terms:

Fig. 15

	Estimate	Std. Error	z value	Pr(>|z|)
depart	-1.543360e-02	7.054558e-04	-21.877490	4.256284e-106
duration	1.247659e-02	3.013603e-03	4.140092	3.471671e-05
carrierAS	-1.273107e+00	2.588052e-01	-4.919171	8.691172e-07
carrierB6	4.475803e-01	9.500204e-02	4.711270	2.461774e-06
carrierDL	2.778274e-01	5.006037e-02	5.549848	2.859187e-08
carrierF9	4.447282e-01	8.277096e-02	5.372998	7.743834e-08
carrierMQ	-2.834391e-01	5.046067e-02	-5.617031	1.942671e-08
carrierWN	2.519927e-01	6.451130e-02	3.906179	9.376701e-05
carrierYX	-7.817555e-01	2.374041e-01	-3.292932	9.914857e-04
poly(depart, degree = 3, raw = TRUE)[, 2:3]2	2.101763e-05	8.830645e-07	23.800783	3.277984e-125
poly(depart, degree = 3, raw = TRUE)[, 2:3]3	-7.750338e-09	3.615451e-10	-21.436712	6.075707e-102
day4:depart	6.319428e-04	3.208283e-04	1.969723	4.887018e-02
day6:depart	8.394617e-04	3.288787e-04	2.552496	1.069541e-02
day5:month8	1.327116e+00	6.462868e-01	2.053447	4.002925e-02
day6:month9	1.641669e+00	6.360424e-01	2.581069	9.849502e-03
day2:depart:month8	1.003955e-03	4.979637e-04	2.016120	4.378739e-02
day6:depart:month8	-1.255911e-03	4.955248e-04	-2.534507	1.126058e-02
day6:depart:month9	-1.816794e-03	4.756830e-04	-3.819338	1.338105e-04
day4:duration:month7	-4.843546e-03	2.208710e-03	-2.192929	2.831246e-02
day3:duration:month8	5.260138e-03	2.358770e-03	2.230034	2.574518e-02
day4:duration:month8	-4.506166e-03	2.273224e-03	-1.982280	4.744795e-02

As can be seen from Figure 15, the main effects of ‘depart’ and ‘duration’ are still significant. Additionally, the same polynomial terms from the second model have remained. Yet many of the two-term interactions are no longer significant with the arrival of three-term ones. This is perhaps because the effects explained by some two-term interactions are explained by three-term interactions in which they are nested (i.e. ‘day6:month8’ and ‘day6:depart:month8’). Nonetheless, to see if there have been improvements in prediction for this third model, a final ROC curve was generated:

Fig. 16



So, the first decrease in AUC when compared to previous models occurred.

Lasso Regression

As previously explored, multicollinearity and overfitting are two possible limitations of linear regression models. Ridge regression is a statistical technique used to counter these limitations by altering the standard least squares estimation method. Typically in standard linear regression, we want to minimize the residual sum of squares (Figure 17) where Y_i is the observed response and Y_i-hat is the predicted response. When a predictor is highly correlated with one or more predictors, the regression model estimates can be less stable thus leading to the estimated coefficients having large variances and the model overall being unreliable with new data (James et al., 2013).

Fig. 17

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

The ridge regression coefficient estimates are the values that minimize Figure 17, where the first term is the RSS and the second term is called the shrinkage penalty.

Fig. 18

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

The shrinkage penalty discourages large coefficients; it "shrinks" the coefficients, which can improve the model when multicollinearity is present (Hastie et al., 2023). The lambda is a parameter that controls the impact of these two terms on the overall ridge regression coefficient estimates. By choosing a larger lambda, the penalty gets bigger and the coefficient estimates get smaller. One key observation is that there is some bias in the coefficient estimates that can come from ridge regression, but the variance is reduced so the tradeoff is deemed as productive (Hastie et al., 2023; James et al., 2013). Another observation is that unlike stepwise regression, ridge regression includes all predictors in the model which can be helpful if all the predictors have an effect on the outcome variable. Ridge regression can also be beneficial because it can be made computationally efficient using algorithms that involve matrix algebra, which makes it more feasible for large datasets (Hastie et al., 2023). The "glmnet" package in R can be used to fit ridge regression models.

One disadvantage of ridge regression however is that it will include all p predictors in the final model, unlike other methods previously seen such as backward stepwise regression and forward selection (James et al., 2013). This can create issues in larger models with many predictors and decrease its interpretability. Lasso regression is a variation of ridge regression that works similarly, but aims to allow for variable selection.

Fig. 19

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|$$

The shrinkage penalty works the same way as in ridge regression, but instead of minimizing the squares of the coefficients it aims to minimize the absolute value of the coefficients. This method allows for not only the minimization of coefficients but also allows them to reach zero (James et al., 2013).

Prior to using lasso regression on our logistic models, we wanted to find the λ best suited for each model. Keeping in mind the bias-variance tradeoff, we'd ideally like to keep both small for our models. Therefore the "best" λ would be one which minimizes Mean Squared Error (MSE), which is the sum of bias and variance. To find this, we used k–Fold Cross-Validation. What this method does is split our training set into k sets of approximately the same size, fit our model using k-1 groups, then finally test on the group not used (validation set) and calculate the MSE. This process is reiterated k times for each possible validation set, and thus a number of k MSE's are generated. Therefore for every possible λ for our lasso regression models, k-Fold Cross-Validation can produce its average MSE:

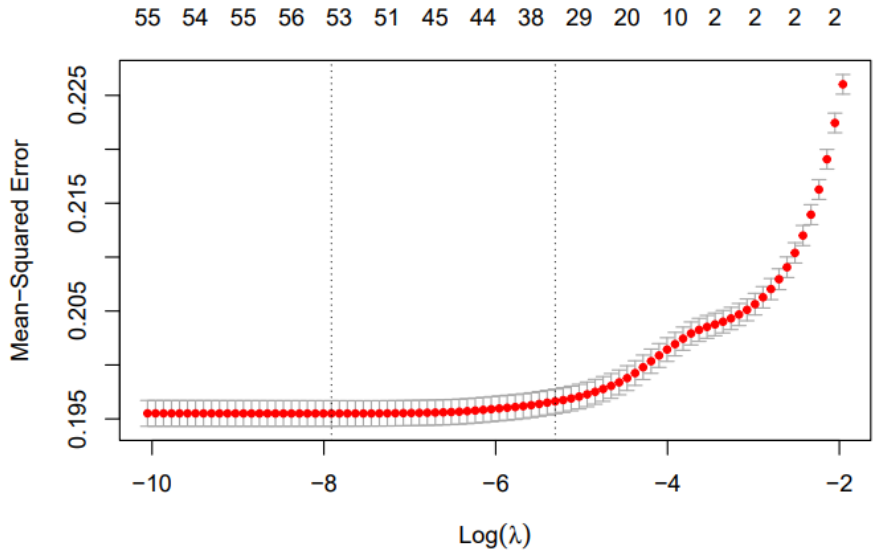
Fig. 20

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i.$$

From there we wanted to find the λ with the minimum average cross-validated MSE (James et al., 2013). For each model fitted below, we used a 10-fold CV.

The first model we created found the best lambda for is the logistic regression model including main effects of day, carrier, depart, duration, and month, as well as pairwise interaction between those same predictors save carrier. Performing CV on this model, it was calculated that λ = 3.66e-04 was the best choice for lasso regression. Below in figure 21 is a visualization of the average MSE for each tested λ in red, as well as error bars for each:

Fig. 21



As indicated by the dotted-line on the left, log(3.66e-04) = -7.91 is the location of the λ with the smallest MSE. After calculating this, we performed lasso regression on the first model. Using the best lambda, lasso produced the coefficients below:

Fig. 22

```
## 58 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)     -2.950659e+00
## day2            -2.511407e-01
## day3            -4.505731e-01
## day4            -3.840735e-02
## day5             1.013705e-01
## day6             5.429713e-01
## day7             .
## depart           1.770400e-03
## duration         2.779023e-03
## month7           .
## month8          -2.222784e-01
## month9          -3.622145e-01
## carrierAS       -1.456499e+00
## carrierB6        5.252402e-01
## carrierDL        1.954304e-01
```

It is important to note that Figure 22 displays partial results, but there is enough here to see lasso regression at work. The coefficients are interpreted the same as they were in logistic regression,

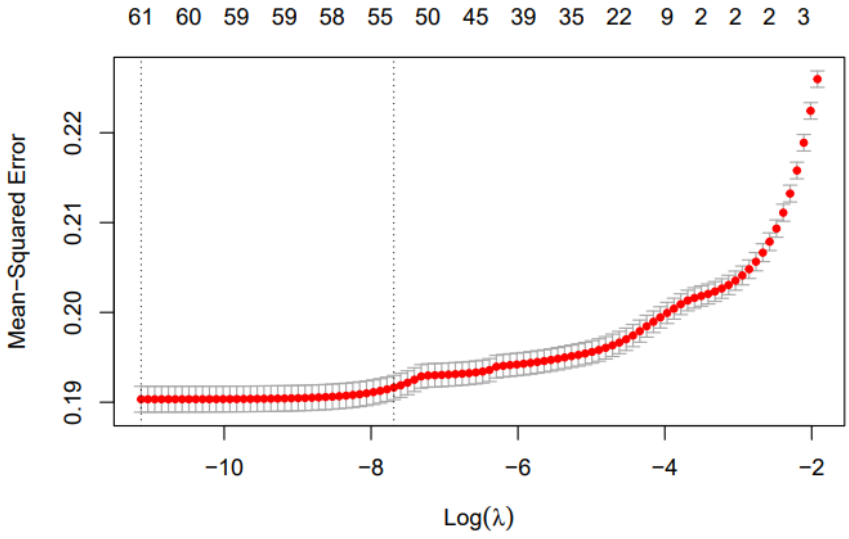
except now for each predictor we can see the new "shrunken" coefficients given the penalty of λ. The values indicated by "." are the coefficients which have been shrunk completely to zero, which displays the desirable effect of removing predictors in lasso regression. Below is the full list of these removed predictors:

Fig. 23

```
##                    s1
## day7                0
## month7              0
## day3:depart         0
## day6:depart         0
## day4:duration       0
## depart:month8       0
## duration:month9     0
```

Shown above, 7 coefficients were shrunken to zero, leaving 51 non-zero terms in the lasso regression model. It seems there is no coincidence that all of these coefficients up to "depart:month8" were not statistically significant in the logistic regression model of previous analysis. Additionally, "duration:month9" was removed by backward selection. The second model we analyze has all the same predictors as the first, but now with the addition of second and third-degree polynomials for depart and duration. Performing CV on this model finds that the λ minimizing MSE is λ=1.46e-05, which can be seen in Figure 24 below:

Fig. 24



Once again, it can be seen from the left-most dotted line in Figure 24 the location of log(1.46e-05) = -11.3. Using this best lambda, the fitted lasso regression for the second model could be

computed. Below are partial results for the coefficients of model 2:

Fig. 25

```
## 62 x 1 sparse Matrix of class "dgCMatrix"
##                                                       s1
## (Intercept)                                 -4.154754e-02
## day2                                        -4.951308e-01
## day3                                        -5.426603e-01
## day4                                        -3.638292e-01
## day5                                         8.485017e-02
## day6                                         3.994313e-01
## day7                                        -2.911416e-02
## depart                                      -1.258614e-02
## duration                                     9.325743e-03
## month7                                       4.490101e-02
## month8                                      -1.993413e-01
## month9                                      -3.189635e-01
## carrierAS                                   -1.159116e+00
## carrierB6                                    4.895816e-01
## carrierDL                                    2.451328e-01
```

Unlike in model 1's partial results, we did not yet have a glimpse of any coefficients being shrunk to zero. However, filtering for this we saw that there were only two for model 2:

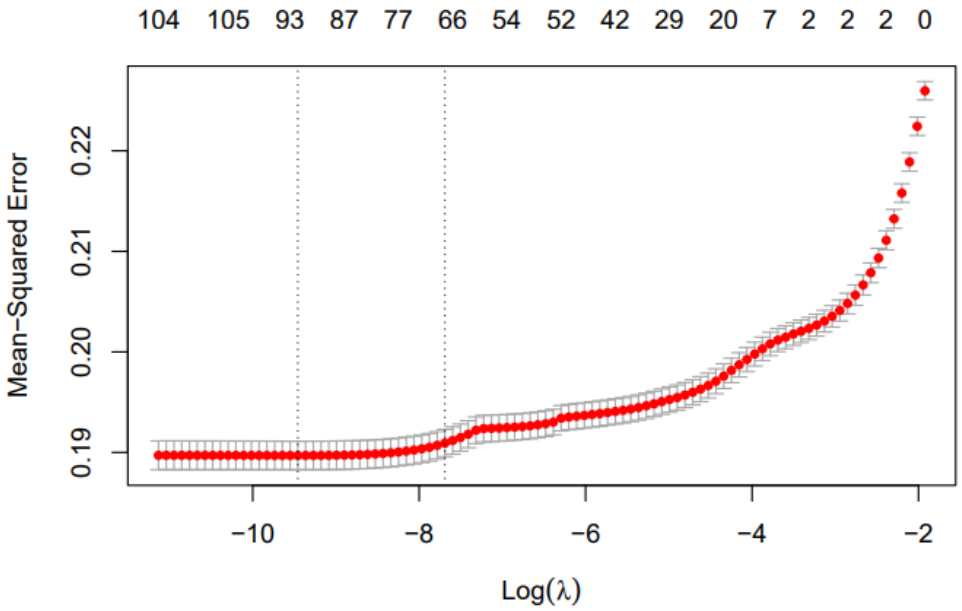
Fig. 26

```
##                                                    s1
## poly(duration, degree = 3, raw = TRUE)[, 2:3]3  0
## day5:depart                                        0
```

In contrast to the first model's lasso results, we only remove two predictors, leaving 60 non-zero terms. Once again, these two coefficients were not statistically significant in previous analysis of the logistic regression model. It may be surprising that after adding more predictors, the amount of removed predictors decreases. However, the RSS for model 1 in logistic regression was higher than that of model 2, and thus fewer coefficients need to be shrunk to zero. This can be attributed to the impact polynomials add to the model, causing the goodness of fit to improve.

The last model we will use cross-validation and lasso regression on has all the same predictors as model 2, but with the addition of three-term interactions for day, depart, duration, and month. After performing CV, the λ minimizing the MSE is λ = 7.81e-05. Shown below is the CV visualization:

Fig. 27



As with the prior plots, the left-most dotted line indicates the location of the best lambda, here with the smallest MSE at log(7.81e-05) = -9.46. Now using this λ for lasso regression, the partial results for model 3 are shown below:

Fig. 28

```
107 x 1 sparse Matrix of class "dgCMatrix"
                                        s1
(Intercept)                 -4.774244e-02
day2                         .
day3                        -2.605681e-01
day4                        -1.760838e-01
day5                         6.033842e-02
day6                        -3.538724e-02
day7                         1.417677e-01
depart                      -1.214941e-02
duration                     8.478726e-03
month7                       1.979802e-01
month8                      -1.130105e-01
month9                      -6.920379e-01
carrierAS                   -1.123361e+00
carrierB6                    4.944157e-01
carrierDL                    2.382336e-01
```

In Figure 28 it can be seen that there are a total of 107 predictors in this model with the addition of three-term interactions. Considering the large number of terms in the full model, for interpretation's sake we wanted there to be a considerable number of predictors that are removed

by lasso. Luckily that was the case, as can be seen by the coefficients that are shrunk to zero:

Fig. 29

```
##                           s1
## day2                      0
## day3:depart               0
## day4:depart               0
## day5:depart               0
## day3:duration             0
## day5:duration             0
## day7:month8               0
## day7:month9               0
## duration:month9           0
## day6:depart:month7        0
## day3:depart:month9        0
## day5:duration:month8      0
## depart:duration:month8    0
## depart:duration:month9    0
```

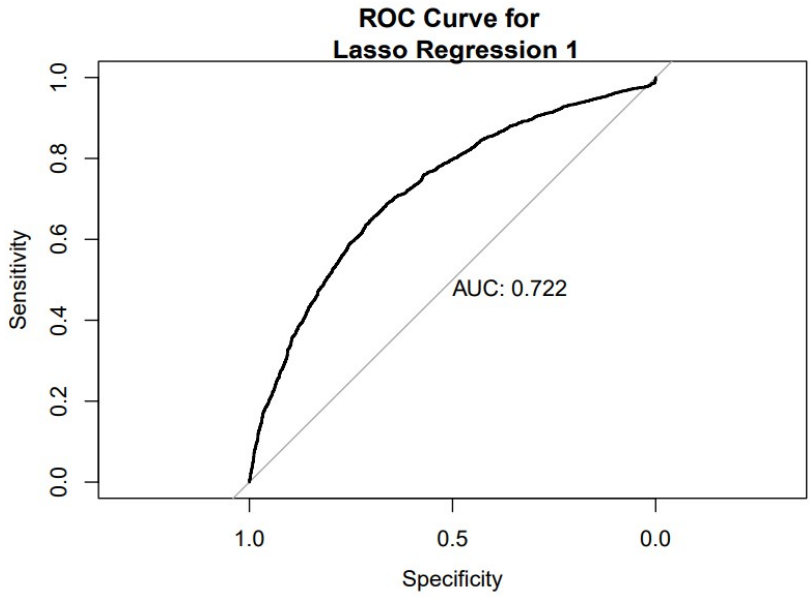
Shown in Figure 29, a total of 14 coefficients were shrunk to zero by lasso, thus leaving 93 non-zero terms left in the model. Following a similar trend to the results of the first model, “depart:duration:month” was previously removed by backward selection for model 3. Additionally, the rest of the predictors shown above were not statistically significant in logistic regression, with the exception of “day4:depart”. Therefore we see in these results the usual suspects of lasso removal. It came as no surprise that for an especially large model such as this one, there needs to be removals made in order to balance minimizing RSS and the number of terms included.

After comparing each lasso regression model on the interpretive level, we wanted to see how they compared when predicting flight delays.

Fig. 30

```
          Reference
Prediction    0    1
         0 3753 1280
         1  553  829

               Accuracy : 0.7143
                 95% CI : (0.703, 0.7253)
    No Information Rate : 0.6712
    P-Value [Acc > NIR] : 6.325e-14

                  Kappa : 0.2902

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8716
            Specificity : 0.3931
         Pos Pred Value : 0.7457
         Neg Pred Value : 0.5999
             Prevalence : 0.6712
         Detection Rate : 0.5850
   Detection Prevalence : 0.7846
      Balanced Accuracy : 0.6323

       'Positive' Class : 0
```

Fig. 31



As we see in figure 30, model 1 has a high sensitivity (0.8716) and relatively good positive predictive value (0.7457), making it effective at correctly identifying positive cases (class 1) and

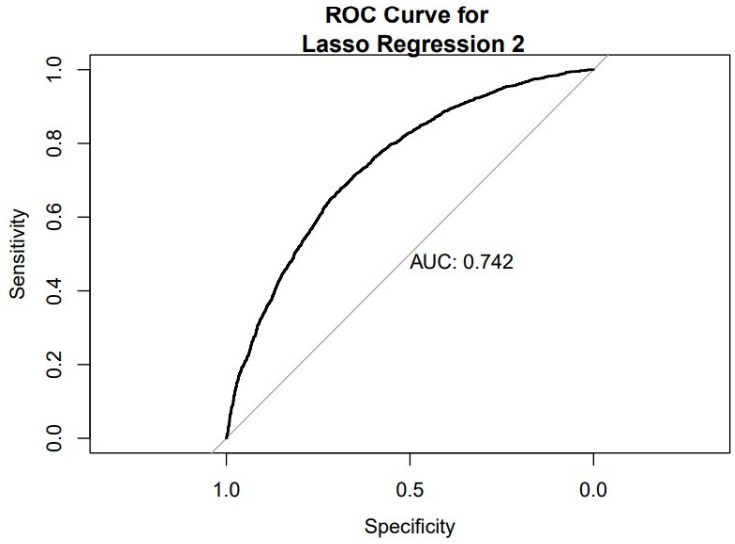
predicting positives with a reasonable success rate. However, its low specificity means that it generates a large number of false positives, which could be problematic. The accuracy (0.7143) is good, but considering the imbalance between sensitivity and specificity, it may be worth further tuning the model, especially if we want to reduce false positives. The ROC curve shows a pretty standard curve with an AUC of 0.722, which shows that this model is pretty good, but could be better.

Shown below are the prediction results for model 2:

Fig. 32

```
          0 3650 1169
          1  656  940

               Accuracy : 0.7155
                 95% CI : (0.7043, 0.7265)
    No Information Rate : 0.6712
    P-Value [Acc > NIR] : 1.193e-14

                  Kappa : 0.3128

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8477
            Specificity : 0.4457
         Pos Pred Value : 0.7574
         Neg Pred Value : 0.5890
             Prevalence : 0.6712
         Detection Rate : 0.5690
   Detection Prevalence : 0.7512
      Balanced Accuracy : 0.6467

       'Positive' Class : 0
```

Fig. 33



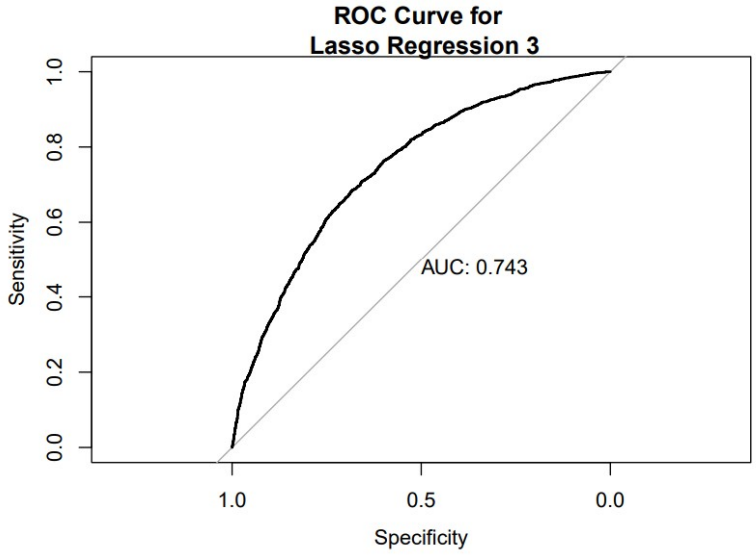
Model 2 has a high sensitivity (0.8477) and good positive predictive value (0.7574), making it reliable for detecting positives (class 1). However, its low specificity (0.4457) indicates that it misclassified many negative cases as positives. The accuracy of 0.7155 is reasonable but could be improved. The ROC curve shows a pretty standard curve with an AUC of 0.742, which shows that this model is pretty good, and slightly better than model 1.

Shown below are the prediction results for model 3:

Fig. 34

```
Prediction    0    1
         0 3679 1200
         1  627  909

               Accuracy : 0.7152
                 95% CI : (0.704, 0.7262)
    No Information Rate : 0.6712
    P-Value [Acc > NIR] : 1.818e-14

                  Kappa : 0.3067

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8544
            Specificity : 0.4310
         Pos Pred Value : 0.7540
         Neg Pred Value : 0.5918
             Prevalence : 0.6712
         Detection Rate : 0.5735
   Detection Prevalence : 0.7606
      Balanced Accuracy : 0.6427

       'Positive' Class : 0
```

Fig. 35



Just like models 1 and 2, model 3 shows good performance in terms of sensitivity (0.8544) and positive predictive value (0.7540). However, the low specificity (0.4310) means that it generates a high number of false positives. An accuracy of 0.7152 is good, similarly to models 1 and 2, but the difference between sensitivity and specificity means that the model favors predicting positives over negatives. The ROC curve shows a pretty standard curve with an AUC of 0.743, which shows that this model is just slightly better than models 1 and 2 for prediction. Ultimately, however, model 2 would be our chosen model for making predictions. It is very slightly lower in AUC, yet by having far fewer variables is much more interpretable.

Creating Simulated Data Sets

After analyzing and finding the results for the MIA data set, we then reverse engineered the logistic regression process. Rather than having a real data set and improving upon a model, we simulated the data such that our logistic regression model gets very close to achieving an AUC of 1 or 0.5. We began at a starting point, and knowing what data makes for strong or weak predictions in a logistic regression, we then made adjustments.

The model throughout this experiment contained 6 main effect variables and 6 higher-ordered or combination terms. Additionally, each variable had a size of 15,000. The first three of the main effects (x1, x2, and x3) were correlated, Normally distributed variables with a covariance of 0.2. The fourth and fifth predictors, x4 and x5, were independently Normally distributed with standard deviations of 1, and mean 2 and -2 respectively. The last main effect, x6, followed a Binomial distribution with probability of success being 0.3. Finally, formulas for the higher ordered and combination terms, as well simulated betas for 12 terms are shown in the table below:

Fig. 36

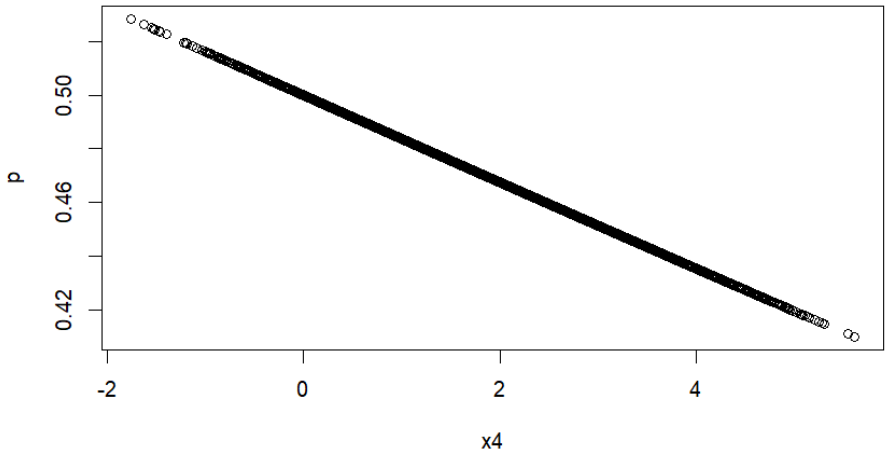
Original Model	
predictor	simulated beta
x1	0.082
x2	-1.21
x3	1.105
x4	-0.065
x5	0.073
x6	0.05
x7 = x1*x4	0.06
x8 = x3*x4*x6	-0.07
x9 = (x1)^2	0.09
x10 = x1*x2	1.1
x11 = (x3)^3	-1.4
x12 = x1*x2*x4	0.12

Using this starting data, the AUC produced for the logistic regression model including each variable is 0.864. To increase or decrease this, we looked at what can be changed in the aforementioned data. More specifically, we wanted to see how AUC changes when we change: the variability in x4 and x5, the covariance between x1, x2, and x3, the number of correlated terms, the probability of success for x6, and finally the simulated true beta values.

There are several factors we considered when changing our model to improve its AUC. In general the ability of a model to classify the response will be improved with a wide range of prediction data that can capture unique information for the model. This means that ideally there would be no correlation between predictors as to prevent instability and prevent redundant information. Additionally, each predictor would have lots of variation so that the model could capture the nuances in the data. This comes with the caveat however that we'd want enough data for our data as to not overfit the model. Finally, we'd want the estimated coefficients in our model to be as close to the true coefficient as possible in order to capture the true effect of our predictor.

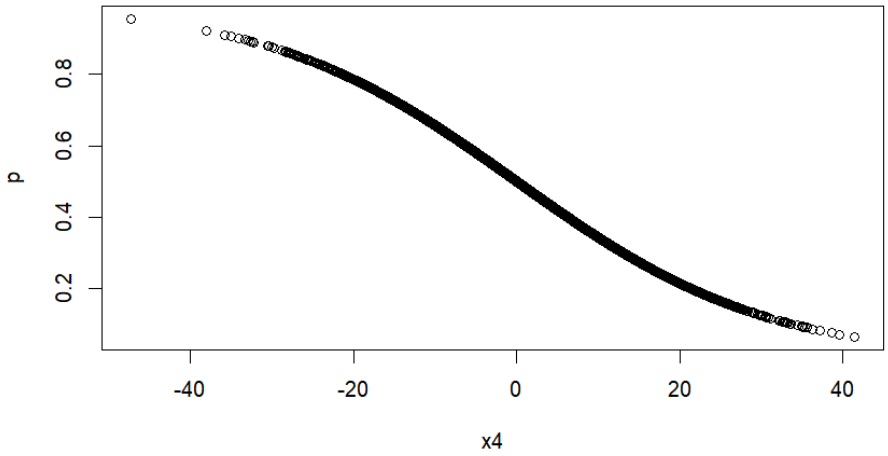
With all of this in consideration, the first alteration we looked at for increasing AUC was the variability in the Normally distributed x4 and x5 variables. To visualize the behavior of the probability of y against variability of one of these predictors, we simulated a model performing logistic regression using just x4 with standard deviation 1:

Fig. 37



As can be seen in Figure 2 the probability of y being 1 has a rather small range across x4. This means that making a correct prediction on account of this variable is not entirely clear, there are a lot of points hovering around p = 0.5. If we were to increase the spread of x4 however by changing its standard deviation to 10, then the same plot would instead look like this:

Fig. 38



Now that x4 is more variable, the values for p are no longer so restricted around 0.5 but instead have many points well above and well below the half-way mark. This makes the model more decisive, as it becomes more clear whether to classify y as 1 or 0. Taking this observation and applying it to the full model, the standard deviations of both x4 and x5 were changed to 10, and the AUC becomes 0.897, marking an improvement of 0.033.

The next glaring issue with the data for achieving high AUC was multicollinearity. One of the main assumptions for logistic regression is that no independent variables are correlated. Correlated predictors can lead to unstable and conflicting betas. This can cause issues in testing for statistical significance in addition to harming the model's ability to predict, and so we wanted to fix this. However, simply decreasing the covariance between x1, x2, and x3 did not help all that much. So to make major improvements, we started by removing one correlated predictor. In its replacement, x3 became a Uniformly distributed variable with a and b being -5 and 5 respectively. Choosing the Uniform distribution was mainly due to the fact it is symmetric. When predicting y = 0 or 1, we did not want there to be any skewness in the data to predict one more often than the other as false positives could occur. The decision for these parameters was to maximize the variability without spreading the data too thin. Implementing the newly generated x3 brought the AUC to an impressive 0.988, improving by 0.091 from the last step taken. Seeing that reducing the number of correlated terms worked so well, we brought it down to zero in hopes of further improvement. To replace x1 and x2, we gave both Logistic distributions with scale 2 and locations of 2 and -2 respectively. Once again, the distribution is chosen for symmetry. Implementing both into the model, and the AUC improves to 0.997, increasing by 0.009 from the last step.

The next parameter we wanted to change was the probability of success in the Binomially distributed x6. The thought process behind fixing this parameter to our advantage is quite similar to what we had done with x4 and x5. Ideally we want to maximize the variability of x6 for reasons previously mentioned. The formula for Binomial variance is np(1-p), which is maximized as p = 0.5. So once this is changed, we run the predictions for the full model and an AUC of 0.998 is achieved, increasing by 0.001 from the last step.

Finally for the high AUC model is changing the simulated true betas. The betas' role in the model is essentially the effect a variable has on the probability of y, which in turn tells us whether to classify as 1 or 0. As of now there is a mix of negative and positive betas, the negative one influencing a classification of 0 and the positive a classification of 1. If each of these values were to increase in magnitude, then the gap of classifying either option would widen, and thus the ability to accurately predict would be enhanced. Therefore, we gave each true beta a 50% increase, which seemed reasonable for simulation's sake while still being effective. The values of each simulated beta are shown below:

Fig. 39

High AUC Model	
Assignment 6	
	simulated
predictor	beta
x1	0.123
x2	-1.815
x3	1.6575
x4	-0.0975
x5	0.1095
x6	0.075
x7 = x1*x4	0.09
x8 = x3*x4*x6	-0.105
x9 = (x1)^2	0.135
x10 = x1*x2	1.65
x11 = (x3)^3	-2.1
x12 = x1*x2*x4	0.18

Making predictions with this final High AUC Model produced an exact AUC of 0.9988048, being incredibly close to one. Additionally, this final model met the goal of having at least 4 main effects and 10 total predictors with statistical significance at the alpha 0.10 level as seen in the summary table below:

Fig. 40

```
Call:
glm(formula = y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 +
    x10 + x11 + x12 - 1, family = binomial(logit), data = train.batch)

Coefficients:
     Estimate Std. Error z value Pr(>|z|)
x1   0.086891   0.041726   2.082   0.0373 *
x2  -1.967327   0.098874 -19.897   <2e-16 ***
x3   1.790763   0.138796  12.902   <2e-16 ***
x4  -0.097510   0.010638  -9.166   <2e-16 ***
x5   0.110131   0.009353  11.775   <2e-16 ***
x6   0.240793   0.124943   1.927   0.0540 .
x7   0.089195   0.005776  15.443   <2e-16 ***
x8  -0.117313   0.010933 -10.730   <2e-16 ***
x9   0.141858   0.009116  15.562   <2e-16 ***
x10  1.742670   0.083908  20.769   <2e-16 ***
x11 -2.236577   0.109366 -20.450   <2e-16 ***
x12  0.188693   0.009041  20.872   <2e-16 ***
```

Having achieved this as well as the high AUC, we then try to reduce the AUC to 0.5.

When trying to get the lowest AUC, we decided to implement each strategy in phases. After looking at how we could get the highest AUC

possible, we had more of an idea for how we could generate the opposite. Each data set had N = 15000 values. We had 12 predictors with 6 being main effects that are simulated from probability distributions and the other being interaction or polynomial terms based off of the 6 main effect predictors. Our x1, x2, and x3 were all correlated normal variables. X4 and x5 were also both normal variables (but not correlated) with means of 2 and −2 respectively. Our x6 was a binomial variable with a probability of 0.5. X9 and x11 are both polynomial terms while x7, x8, x10, and x12 are all interaction terms. The form of the predictors and their original beta coefficient values are seen in Figure 36. All of our variables except for x7 were statistically significant at our starting point. While making changes to our variables and their characteristics, we had to make sure that at least 2 predictors for main effects and 5 total simulated terms in the model were significant at the level of alpha ≤ 0.25. Our starting AUC value was 0.863.

We first looked at how changing the variability of our predictors impacted our AUC curve. Since our polynomial and interaction terms (x7 through x12) are based off of the main effects predictors, we can't change their variability and can only change the variability of the main effect terms. X1 through x3 are all correlated random normal variables and x6 is a binomial variable whose variance is based off of the probability, so we couldn't change the variability of these variables. This leaves us with x4 and x5 which are both normally distributed variables with a variance of 1. We decreased the variance of x4 and x5 from 1 to 0.01. Our reasoning behind this is that when we decrease the variance, the values we get across observations are closer together. If the values are more similar across observations, then the values from different classes are going to be more likely to overlap each other and give less of a distinction between classes. This can make it harder for our model to distinguish between the classes which lowers the AUC. After making this change, half of our predictors were statistically significant and the other half were not. The main effect variables x2, x3, and x6 along with high-ordered/combination variables x8, x9, and x11 were statistically significant. Our AUC reduced slightly from 0.863 to 0.860.

Our next step was changing the correlation of our correlated variables, x1 through x3. We can change the correlation by changing the covariance. This is possible because the correlation of two variables is equal to their covariance divided by each variable's standard deviation. The variance for x1, x2, and x3 are all 1 so the correlation is equal to the covariance. Our starting correlation was 0.2 and we increased it to 0.7. X1, x2, and x3 being highly correlated means that they predict similar information. If we are only trying to predict with x1 but also look at x2 and x3 to try to get more insight, we aren't really getting much new information or contributions.

Adding them to a model doesn't improve our model's predictions by a lot since the insights we get from them aren't unique or independent. These variables having a high correlation with each other will most likely lead to multicollinearity, which is when one or more variables are highly correlated with each other, and this can make it difficult for our model to know the effect of each predictor independently and makes results unreliable. After increasing the correlation, we had 5 terms that were statistically significant: x2, x3, x8, x9, and x11. Our AUC decreased from 0.860 to 0.838.

Fig. 41

Final Low AUC Model	
predictor	simulated beta
x1	0.082
x2	-0.12
x3	0.15
x4	-0.065
x5	0.073
x6	0.05
x7 = x1*x4	0.06
x8 = x3*x4*x6	-0.07
x9 = (x1)^2	0.09
x10 = x1*x2	0.11
x11 = (x3)^3	-0.14
x12 = x1*x2*x4	0.12

We also looked into how changing our true beta values would affect our AUC. We can see in Figure 36 our original true beta values. Our method in decreasing the beta values was to multiply large ones by 0.1 or 0.01 and keep the lower beta values. We ended up with the beta values seen in Figure 41. The beta coefficients overall tell us how much of an impact each predictor has on the outcome. By reducing these coefficients and making them relatively small, our predictors are going to have less of an effect on our outcome. Since they have a small effect, it is harder for our model to accurately make predictions on our outcome based on our predictors. After these changes, the variables that were statistically significant didn't change and are the same as previously mentioned. Our AUC decreased from 0.838 to 0.639. Changing the true beta values has been the most impactful on our AUC thus far.

The next step was to figure out how changing the probability of success for the binary predictor (x6) affected the significance of x6, the AUC, and potentially the significance of the other predictors. Keeping all other parameters constant as they were given in the original model, and only decreasing the binomial probability, the model AUC increased from 0.863 to 0.864, which makes sense because a smaller probability would cause a greater distinction between classes and a better performance, as described in the high AUC model. By increasing the

probability of success to 0.7, an imbalance is created in the predicted classes. The model started to predict the positive class more often and there is less variability in the predictions. The new AUC becomes 0.862. This is reflected in the fact that the t statistic of x6 now jumps to 0.659 and makes it no longer significant, while all other predictors stay significant at the alpha = 0.25 level.

The next step was to see how changing the number of correlated predictors affected the AUC. By increasing the number of correlated predictors from 3 to 6 and adjusting the covariance from 0.2 to 0.7, and leaving all the other parameters as they were given originally, the model had a little impact on the AUC. Simply changing the correlated predictors to 6 from 3 with a covariance of 0.2 did not affect the AUC at all, but when changing the covariance to 0.7, the AUC went down by just 0.01. In theory, the increased number of correlated predictors causes multicollinearity, which affects the model's ability to predict unique contributions from each predictor, resulting in a lower AUC due to poorer distinction of classes. But in this case, the main factor which affected the decreased and increased AUC values was the changing of the true betas.

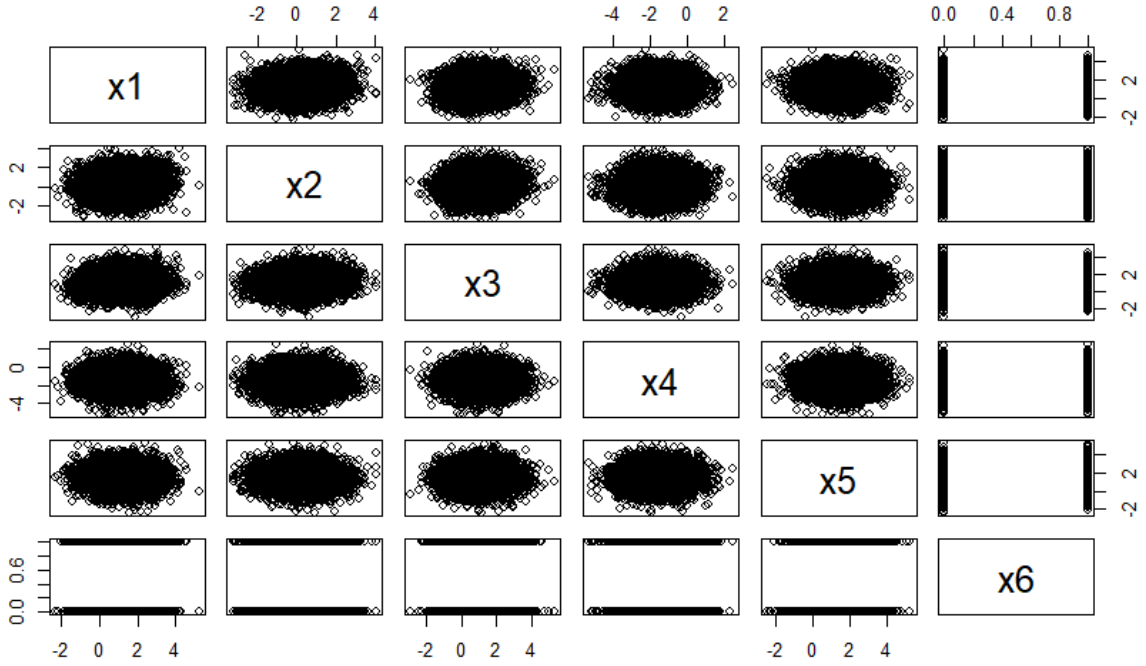
Analyzing Simulation Data Sets

We then continued the analysis of simulated data sets and prediction accuracy, except now the data sets were not to be altered. Given two simulated data sets, one for achieving High AUC and the other for Low AUC, we set out to use what we know about interaction terms, polynomial terms, and variable selection methods in order to reach goal AUC's.

For both High and Low AUC, the structure of the data sets were the same. The full simulated data sets are size n = 15,000 with 6 predictor variables, x1 through x6. Also given are the training and testing sets, which are size n = 10,000 and n = 5,000 respectively. Other than these shared qualities between both, the data sets differ in values and desired AUC's, and thus we can begin detailing the different approaches taken for each goal.

Prior to making any decisions on how the model should be constructed, we wanted to first visualize the High AUC training data. Using a scatterplot matrix, we were be able to see the relationships between predictors and the values of the variables themselves:

Fig. 42



As can be seen, there appeared to be no multicollinearity between the predictors as each of the plots between x1 through x5 are relatively null. This was a good starting point for achieving high AUC, as we did not have to worry too much about multiple predictors explaining the same variance. Another thing we can note from Figure 42 is that x6 is binary, and so when we eventually created polynomial terms it would be excluded from such transformations.

Once we became more familiar with the High AUC data, we wanted to begin model creation. A large inspiration for this was the results of our analysis on the Miami International Airport data. Through trying different models and variable selection methods on said data set, we found what methods increased AUC the most. The first of these was the inclusion of interaction and polynomial terms. For the MIA models, we worked with 3 different models, two of which are larger and include polynomial terms and two-or-three-way interactions. Generally the two larger models had significantly higher AUC's than the simpler model. This was likely due to the help of the second effective component, and that was variable selection methods. If we were to simply increase the number of variables in the model without variable selection, we would risk overfitting, and thus prediction performance weakens on a test or validation set. However, through the use of Lasso Regression and Backward Selection we are able to remove any

variables that potentially hurt AUC while still expanding upon a less predictive small model. Therefore, we proceed by finding the AUC for two larger models with Backward Selection, and two with Lasso.

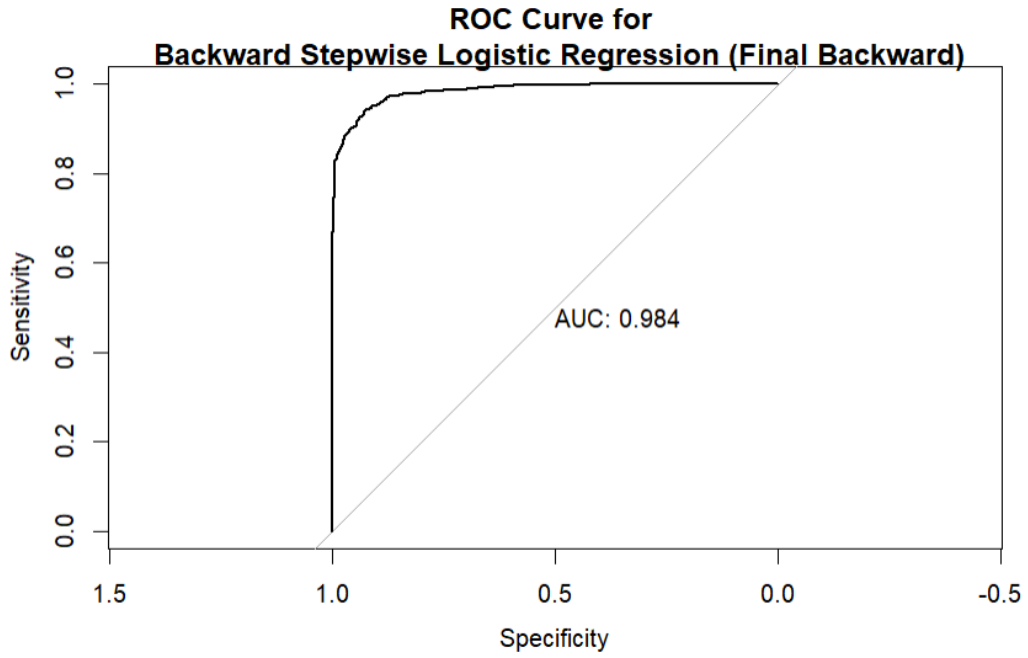
Both models we analyze include all six main effects, as well as second and third order polynomials for x1 through x5. Where the two models differ is that the first includes pairwise interactions, whereas the second includes three-way interactions. If interpretability was a consideration, these models would become increasingly difficult to describe. However, this is simulated data and we are simply trying to maximize correct predictions rates, so it is of no concern.

We started our model by using backwards stepwise regression for pairwise interactions and third degree polynomials for all main effects (except x6 as it is a binary variable). This led us to an AUC of 0.979. A few of the third degree polynomial regressors proved to be statistically significant at the 0.05 level so we decided to keep the third degree polynomial in the regression, but this time try again with three way interactions between the regressors. This led us to an AUC of 0.984. The coefficients and ROC graphs for this final backwards stepwise regression model are shown below in Figure 43 and Figure 44.

Fig. 43

```
Coefficients:
                                     Estimate Std. Error z value Pr(>|z|)
(Intercept)                           0.60422    0.31974   1.890  0.05880 .
x1                                    1.32855    0.25741   5.161 2.45e-07 ***
x2                                   -2.04139    0.28894  -7.065 1.60e-12 ***
x3                                   -0.89820    0.32338  -2.778  0.00548 **
x4                                    1.72038    0.20297   8.476  < 2e-16 ***
x5                                    1.71015    0.21534   7.942 1.99e-15 ***
x6                                    0.85783    0.11641   7.369 1.72e-13 ***
poly(x1, degree = 3, raw = TRUE)[, 2:3]2 -1.39580    0.12659 -11.026  < 2e-16 ***
poly(x1, degree = 3, raw = TRUE)[, 2:3]3 -0.01129    0.04054  -0.279  0.78057
poly(x2, degree = 3, raw = TRUE)[, 2:3]2 -0.12558    0.09496  -1.322  0.18604
poly(x2, degree = 3, raw = TRUE)[, 2:3]3  0.10485    0.04576   2.291  0.02195 *
poly(x3, degree = 3, raw = TRUE)[, 2:3]2 -0.01361    0.17929  -0.076  0.93949
poly(x3, degree = 3, raw = TRUE)[, 2:3]3  2.08546    0.11704  17.818  < 2e-16 ***
x1:x2                                -0.37761    0.18011  -2.096  0.03604 *
x1:x3                                -0.84593    0.20934  -4.041 5.32e-05 ***
x1:x4                                -0.27655    0.15154  -1.825  0.06801 .
x1:x5                                 0.21585    0.13940   1.548  0.12152
x2:x3                                 0.40177    0.22683   1.771  0.07652 .
x2:x4                                 0.25303    0.16869   1.500  0.13362
x2:x5                                -1.36072    0.12864 -10.577  < 2e-16 ***
x3:x4                                -0.28234    0.16726  -1.688  0.09142 .
x3:x5                                 1.34035    0.15917   8.421  < 2e-16 ***
x4:x5                                -0.26998    0.11474  -2.353  0.01863 *
x1:x2:x4                             -0.19439    0.10367  -1.875  0.06078 .
x1:x3:x4                              0.14991    0.10031   1.494  0.13505
x1:x3:x5                              0.19004    0.09265   2.051  0.04025 *
x1:x4:x5                              0.16100    0.07396   2.177  0.02949 *
x2:x3:x4                              1.85523    0.12819  14.472  < 2e-16 ***
x2:x3:x5                             -0.16391    0.10077  -1.627  0.10384
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Fig. 44

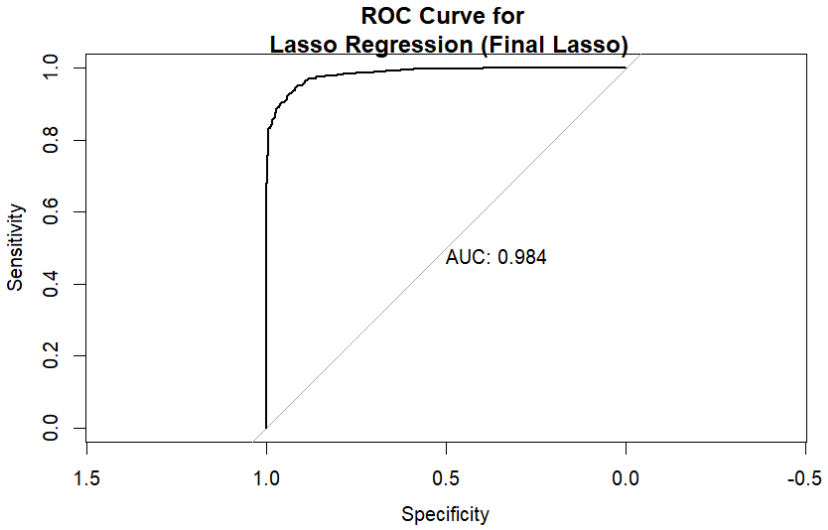


Next we decided to see if we could use Lasso regression to obtain an even higher AUC. Since our best model for Backwards Stepwise regression included third order polynomials and three way interaction effects, we decided to run the same model through Lasso regression. This Lasso regression included the main effects, three way interactions, and the second and third degree polynomials. With this we got an AUC of 0.9843419. Shown below are the coefficients and the ROC curve.

Fig. 45

```
(Intercept)                                 0.28098953
x1                                          1.35295843
x2                                         -2.12740840
x3                                         -0.50788921
x4                                          1.43413535
x5                                          1.91964165
x6                                          0.82462397
poly(x1, degree = 3, raw = TRUE)[, 2:3]2 -1.34147340
poly(x1, degree = 3, raw = TRUE)[, 2:3]3 -0.01230513
poly(x2, degree = 3, raw = TRUE)[, 2:3]2 -0.10178779
poly(x2, degree = 3, raw = TRUE)[, 2:3]3  0.10112207
poly(x3, degree = 3, raw = TRUE)[, 2:3]2  0.01135699
poly(x3, degree = 3, raw = TRUE)[, 2:3]3  2.00458107
poly(x4, degree = 3, raw = TRUE)[, 2:3]2 -0.06751346
poly(x4, degree = 3, raw = TRUE)[, 2:3]3 -0.01141079
poly(x5, degree = 3, raw = TRUE)[, 2:3]2  .
poly(x5, degree = 3, raw = TRUE)[, 2:3]3 -0.01649335
x1:x2                                      -0.15872024
x1:x3                                      -0.91993572
x1:x4                                      -0.16573428
x1:x5                                       0.19262477
x2:x3                                       0.37618019
x2:x4                                       0.23685052
x2:x5                                      -1.38835515
x3:x4                                      -0.08078273
x3:x5                                       1.18432706
x4:x5                                      -0.17569851
x1:x2:x3                                   -0.08798278
x1:x2:x4                                   -0.08427391
x1:x2:x5                                    .
x1:x3:x4                                    0.05150276
x1:x3:x5                                    0.17897129
x1:x4:x5                                    0.12233737
x2:x3:x4                                    1.74476323
x2:x3:x5                                   -0.13892527
x2:x4:x5                                   -0.08328266
x3:x4:x5                                   -0.03475761
```

Fig. 46



This AUC being very close to the goal of 0.988, we stopped here and adopted the Lasso Regression model for predictions made on the testing set.

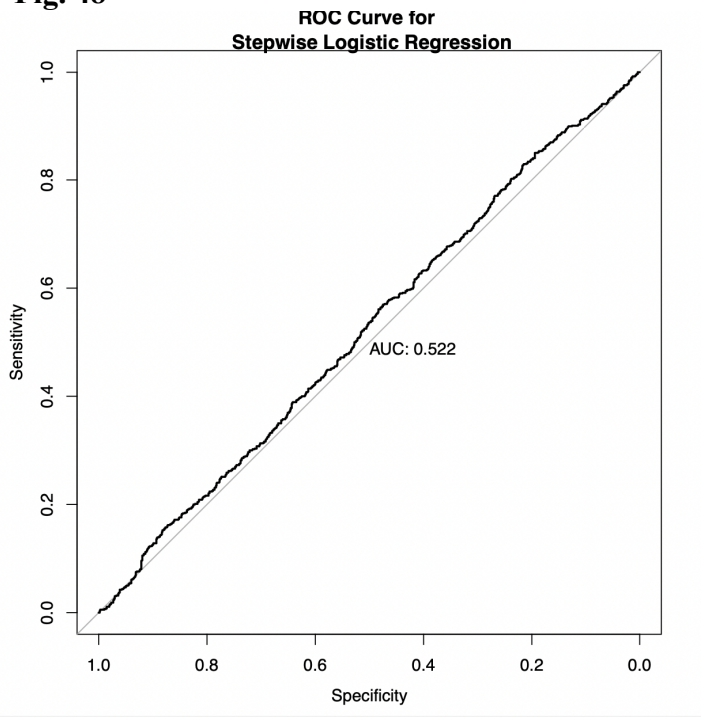
The AUC of the Low AUC data we were given was around 0.517 and our goal was to get a model with an AUC that matches it. We started off with three different types of regression in mind: backward stepwise regression, forward stepwise regression, and lasso regression. Backward stepwise regression was the first regression that we tested out and we tried models with different complexities. After seeing the range of AUCs we could get with backward stepwise, we looked into forward stepwise regression. For the same model, forward stepwise regression performed worse than backward stepwise. This could be because forward stepwise was more restrictive with its inclusion of predictors than backward. Removing predictors that aren't that statistically significant by themselves could hurt our model's performance and give it less to rely on. The last type of regression we looked at was lasso regression. We did both lasso and forward stepwise regression for the same models and concluded that forward stepwise regression performed worse than lasso. Lasso regression shrinks less important predictors which in our case seemed to strengthen the model. An interesting observation is that lasso regression was computationally faster by a few seconds than forward stepwise regression for really complex models. After all of our comparisons, we decided to move forward with forward stepwise regression.

Our next step in building our model was to determine what predictors we were going to include. We started off with the most complex model possible which had all main effect predictors x1 through x6, two-way and three-way interaction terms with all the main effects, and polynomial terms with all the main effects (both squared and cubed). This gave us an AUC that was in the 0.53 - 0.54 range. Removing various squared and cubed polynomial terms seemed to worsen our model and after this we looked at the interaction terms. There were a lot of different interaction terms we started off with so we removed them one by one and saw that the AUC decreased. We removed all of the polynomial terms as well as the three-way interaction terms. This left us with the predictors seen in Figure 47 with an AUC of 0.522 which can be seen in Figure 48.

Fig. 47

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.002494   0.048384  -0.052  0.95888
x1           0.065059   0.060156   1.082  0.27947
x2          -0.012464   0.057740  -0.216  0.82910
x3          -0.012215   0.059568  -0.205  0.83752
x4          -0.133665   0.056488  -2.366  0.01797 *
x5           0.107224   0.032972   3.252  0.00115 **
x6           0.025010   0.094194   0.266  0.79061
x1:x2       -0.002796   0.042268  -0.066  0.94726
x1:x3       -0.004563   0.042863  -0.106  0.91522
x1:x4        0.015848   0.042457   0.373  0.70895
x1:x5        0.033276   0.034007   0.979  0.32782
x1:x6       -0.087612   0.083287  -1.052  0.29283
x2:x3        0.016019   0.042542   0.377  0.70651
x2:x4        0.004330   0.043103   0.100  0.91998
x2:x5        0.016611   0.032479   0.511  0.60905
x2:x6        0.079343   0.082577   0.961  0.33664
x3:x4       -0.008869   0.043967  -0.202  0.84014
x3:x5       -0.007378   0.034027  -0.217  0.82834
x3:x6       -0.056061   0.083287  -0.673  0.50088
x4:x5       -0.047300   0.033651  -1.406  0.15984
x4:x6        0.028651   0.084438   0.339  0.73437
x5:x6       -0.038501   0.056268  -0.684  0.49382
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Fig. 48



For both High and Low AUC models, we then wished to assess how close our models and predictions were to the true ones. Our high AUC was 0.988 and our low AUC was 0.522. First, we compared our simulated betas and predictors from our chosen High and Low AUC models to the true betas and predictors used to simulate the original datasets. Then, we used a

confusion matrix to compare our predicted “y” values for the testing sets to the true “y” values that were originally simulated.

Fig. 49

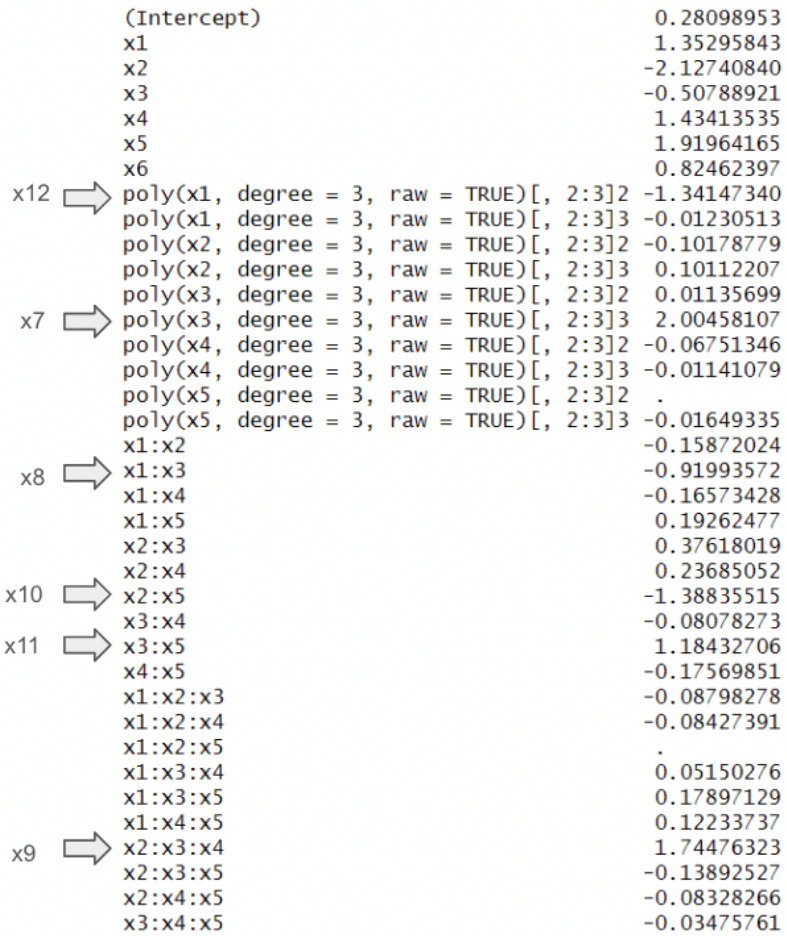


Fig. 50

predictor	simulated true beta
x1	1.7
x2	-1.8
x3	-0.6
x4	1.3
x5	2.1
x6	0.8
x7 = (x3)^3	2.0
x8 = x1*x3	-0.9
x9 = x2*x3*x4	1.8
x10 = x2*x5	-1.6
x11 = x3*x5	1.5
x12 = (x1)^2	-1.4

We started off comparing the beta values of our main effect predictors (x1 through x6) for our high AUC model. The beta values and predictors for our high AUC model are in Figure 49 while the true beta values and predictors are in Figure 50. For both models, the main effect predictors were included. The beta values for our model were relatively close to the true beta values. X6 had the closest beta value (0.82) to its true value (0.8) while x2 had the farthest beta value (-2.13) to its true value (-1.8). For our other estimated predictors, we included many polynomial and interaction terms. The true predictors, other than the main effects, are listed in Figure 49 as x7 through x12. By looking at Figure 49, we can see that our model includes these

predictors, as well as a lot of others. Comparing the beta values of the predictors seen in both models, the beta values are almost exactly the same except for x9 through x11. Even still, the estimated beta values for x9 through x11 are not too far off from the true beta values. Reflecting on our model and previous work, the estimated beta values for the predictors not included in the true model are very low so we definitely could have excluded them.

Fig. 51

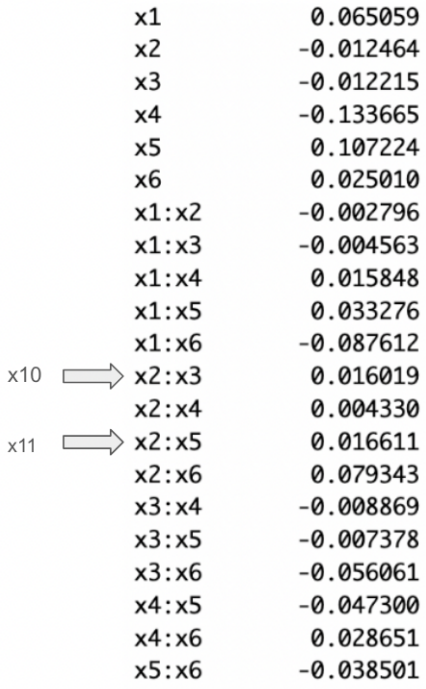


Fig. 52

predictor	simulated true beta
x1	0.002
x2	0.003
x3	-0.006
x4	-0.050
x5	0.070
x6	0.003
x7 = x2*x3*x4	0.001
x8 = (x3)^2	-0.005
x9 = x1*x4*x5	0.007
x10 = x2*x3	-0.002
x11 = x2*x5	0.001
x12 = (x1)^3	-0.002

Now looking at our low AUC model, we started off by comparing the beta values of our main effect predictors (x1 through x6). The beta values and predictors for our low AUC model are in Figure 51 while the true beta values and predictors are in Figure 52. For both models, the main effect predictors were included but we can see that the beta values for both models are very different and even the signs differ slightly. The x2 predictor is negative in our model while it is positive in the true low AUC model. The AUC of our model is larger than the AUC of the true model by only 0.005, so it is interesting to see how this seemingly small difference translates into a pretty large difference between our beta values and predictors. We can look now at the other predictors and their beta values. We included a lot more interaction term predictors in our model compared to the true low AUC model, but there still isn't much overlap between the predictors included in both models; only x10 and x11 are in both models. We lack more complex interaction terms and polynomial terms in our simulated low AUC model. The estimated beta values for x10 and x11 are very different from their true values, but this makes sense after seeing how different the beta values for our main effects predictors are from their true values, as x10 and x11 are interaction terms of some of our main effects.

We then compared the true "y" values of the test set for the high AUC model with our predicted values. The confusion matrix and other statistics are shown below.

Fig. 53

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1599  121
         1  162 3118

               Accuracy : 0.9434
                 95% CI : (0.9366, 0.9496)
    No Information Rate : 0.6478
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8753

 Mcnemar's Test P-Value : 0.01742

            Sensitivity : 0.9080
            Specificity : 0.9626
         Pos Pred Value : 0.9297
         Neg Pred Value : 0.9506
             Prevalence : 0.3522
         Detection Rate : 0.3198
   Detection Prevalence : 0.3440
      Balanced Accuracy : 0.9353

       'Positive' Class : 0
```

The model performed at a high accuracy of 94.34%, with a 95% confidence interval between 93.66% and 94.96%. The confusion matrix indicates strong performance, with 1599 true negatives, 3118 true positives, 121 false negatives, and 162 false positives. Sensitivity and specificity are both also high at 90.80% and 96.26%, which shows the model's ability to correctly identify both classes.

We then did the same process with the Low AUC dataset. Shown below are the confusion matrix along with other prediction related statistics:

Fig. 54

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2383 1986
         1  334  297

               Accuracy : 0.536
                 95% CI : (0.5221, 0.5499)
    No Information Rate : 0.5434
    P-Value [Acc > NIR] : 0.8565

                  Kappa : 0.0076

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8771
            Specificity : 0.1301
         Pos Pred Value : 0.5454
         Neg Pred Value : 0.4707
             Prevalence : 0.5434
         Detection Rate : 0.4766
   Detection Prevalence : 0.8738
      Balanced Accuracy : 0.5036

       'Positive' Class : 0
```

As we can see from the results in Figure 54, the overall accuracy for our Low AUC test predictions is 53.6%. This is very reminiscent of the AUC of 0.522 achieved on the validation set, showing similar prediction behavior for the test set. Additionally, we once again had achieved near randomness, with almost 50% overall accuracy. One major contributing factor to this is the Specificity of 0.1301, meaning the Low AUC model is very poor at predicting 0.

Function & R Package: EvalModels

Our evaluate_models function within our EvalModels R package is designed to compare the performance of regression models, including Logistic Regression, LASSO Regression, and Ridge Regression for binary classification tasks. The function takes as input a dataset, a formula string specifying the response variable and predictors, and an optional parameter to specify the proportion of data to include in the training set (train_test_split, which defaults to 70%), with the remaining data being used for the test set. The function preprocesses the data, fits the models, and evaluates their performance, providing metrics such as accuracy, precision, recall, F1-score, and AUC. It also has error handling for invalid inputs, including formula validation. We can apply the function to multiple flight datasets, which contain detailed flight information such as departure and arrival times, flight carriers, distances, departure times, duration, day of the week, and month. The response variable represents a binary outcome, such as delay, which is the target for classification. To account for datasets that may have different response variable names or to

analyze other binary outcomes, the response variable and predictors are specified through a formula parameter. Predictors include both numerical and categorical variables. The evaluate_models function preprocesses the dataset by handling rare levels in categorical predictors like carrier, splitting the data into training and testing sets, and then fitting the three models. The performance of each model is evaluated using multiple metrics.

Our function uses logistic regression and its regularized variants, LASSO and Ridge regression, to make predictions. Logistic regression is a statistical method used for binary classification, modeling the probability of an event occurring based on many predictors. Regularized methods like LASSO and Ridge regression improve logistic regression by preventing overfitting and generalizing the model to new test data. LASSO regression achieves this by shrinking some coefficients to zero, effectively performing variable selection, which makes subset selection unnecessary in this function. Ridge regression prevents overfitting by shrinking all coefficients closer to zero without eliminating any predictors. To further enhance LASSO and Ridge regression, we used the cv.glmnet function from the glmnet library, which performs k-fold cross-validation. This method splits the data into k folds, training the model on k-1 folds and validating it on the remaining fold. The process is repeated for a range of lambda values, allowing the cv.glmnet function to compare models with different degrees of regularization. The function identifies the best lambda value that minimizes the cross-validation error.

The evaluate_models function begins by validating the input parameters. It ensures the input data is a data frame and that train_test_split is a value between 0 and 1. The response variable and predictors are extracted from the formula string. If the formula is invalid, the function uses a tryCatch block to return an error message, preventing more data processing incorrect with inputs. Additionally, the function checks that the response variable is binary, returning an error if this condition is not met. The function preprocesses the data by removing irrelevant columns related to flight destinations and origins. It processes the categorical predictor carrier by grouping rare levels (those with fewer occurrences than 5% of the training data) into a single category labeled "Other." The response variable is converted into a factor to indicate that it is binary. The data is then split into training and testing sets using stratified sampling to preserve the distribution of the response variable, with 70% of the data allocated to training by default. After preprocessing, the formula is converted into a formula object with the as.formula function, and matrices for the training and testing datasets are generated using this formula. The function fits three models: LASSO Regression, Ridge Regression, and Logistic Regression. The LASSO and Ridge models are trained using the cv.glmnet function with cross-validation to identify the optimal lambda values, while the Logistic Regression model is fitted using the glm function. Predictions are made on the test set for each model, and performance metrics are calculated. A custom calculate_metrics helper function computes metrics such as precision, recall, F1-score, accuracy, AUC, and the confusion matrix. The code ensures the confusion matrix is always 2x2, even in cases where predictions result in only one class. The function then

compiles the performance metrics into a summary data frame, which includes AUC, accuracy, precision, recall, F1-score, and confusion matrices for all three models.

R Shiny App

Our Shiny app (Function reference - 1.9.0, Shiny) is called the Regression Model App, which is designed to enable the evaluation of various regression models, including logistic regression, lasso regression, and ridge regression. Our app allows users to upload datasets, specify a regression formula, and select the desired regression model. It helps users analyze relationships between variables and assess model performance, and make predictions. This app is useful for working with binary classification problems, where the goal is to predict a categorical outcome.

This app allows users to upload a dataset in CSV format and then give the formula they wish to use for the regression model. The user can pick between three different types of models: lasso, ridge, and logistic regression. After making these selections and inputting the information, the app splits the dataset into a training set and a testing set according to a specified ratio, which the user can adjust using a slider; the default split is 0.7 in the training set. The user can also select what variable they want to be visualized in a barplot or histogram. The app first outputs the precision, accuracy, f1 score, recall, and AUC of the specified model. It then outputs the standard regression summary output table that shows the coefficients of our variables and their statistical significance. The data visualization of the user's choosing is at the very bottom of the output.

The application begins by allowing users to upload a dataset in CSV format. The uploaded data is read using the read.csv function, with column types and missing value placeholders specified. Certain columns, such as categorical identifiers (e.g., "dest" and "origin"), are excluded from the dataset to focus on relevant predictors. The dataset is also cleaned by removing rows with missing values, ensuring that the models are trained on complete data. An optional sampling mechanism can be applied to limit the dataset size for performance optimization, but it is not incorporated into the main interface. Model Training and Evaluation The user-specified formula is validated and created into a formula object using as.formula, which is a compatible formula with the regression models. The response variable is converted to a factor for binary classification. The dataset is then split into training and testing sets based on the user defined ratio using createDataPartition from the caret package, which performs stratified sampling. Categorical predictors are converted into dummy variables using the model.matrix function to ensure compatibility with the models. For logistic regression, a generalized linear model (glm) with a binomial family is trained. Predictions on the test set are obtained using the predict function, and performance metrics are calculated. For lasso and ridge regression, the cv.glmnet function from the glmnet package is used, with cross-validation determining the best λ. Lasso regression uses an alpha value of 1, while ridge regression uses an alpha value of 0. Test set predictions are generated using the best λ value for each model. This method means that a

validation set is not necessary, since the training set itself is used as a validation set within the cross validation technique.

The R Shiny app returns metrics to assess the efficacy of the selected regression models. Metrics include precision, recall, F1-score, accuracy, and AUC, which are displayed in the output panel alongside a confusion matrix summarizing true positives, true negatives, false positives, and false negatives. In addition to the metrics, the app includes an intuitive user interface. Users can upload datasets using a simple file input field, specify regression formulas by directly typing them into a text box, and adjust the train-test split ratios using an interactive slider. The app also features a dropdown menu for selecting the regression model type (logistic, lasso, or ridge regression), making it easy for users to compare different approaches. Additionally, there is a dropdown menu for selecting variables, which allows users to visualize data distributions. Histograms are generated for numerical variables, while bar plots are created for categorical variables. These visualization tools enable users to assess the distribution of each variable, identify potential outliers.

Discussion and Conclusions

Our research began with the hopes of predicting flight delays, and finding what contributes most to them. After comparing several logistic regression models of different sizes and variable selection methods, the model that best balanced prediction accuracy and interpretability was the Lasso Regression Model 2. Additionally, a recurring interpretation of the data from the visualizations through the model analyses was that departure time was the largest contributing factor to delays. Once our real world question was answered, we successfully achieved high and low AUC models using simulation studies, and in doing so fully explored how logistic regression works. This being done, we had the confidence to create a function and shiny app that could consolidate the processes used in our analysis, and we hope it will serve anyone who hopes to make predictions of their own using logistic regression.

References

Bevans, R. (2020) Akaike Information Criterion | When & How to Use It (Example). https://www.scribbr.com/statistics/akaike-information-criterion/

Casella, G. and Berger, R.L. (2002) Statistical Inference. 2nd Edition, Duxbury Press, Pacific Grove.

Fawcett, T. (2006) An Introduction to ROC Analysis. Pattern Recognition Letters. 27 (8): 861–874.

Function reference - 1.9.0. Shiny. https://shiny.posit.co/r/reference/shiny/latest/

Hastie, T., Qian, J., Tay, K. (2023). glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models. R package version 4.1-8. https://cran.r- project.org/web/packages/glmnet/

James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R (sixth printing). New York: Springer.

Miami-Dade County, (2024) About Us; [accessed 2024 Sep 13]. https://www.miami-airport.com/about_us.asp

Robin X, Turck N, Hainard A, Tiberti N, Lisacek F, Sanchez J, Müller M (2011) pROC: an open-source p

Sperandei, S. (2014) Understanding logistic regression analysis. *Biochemia Medica* 24: 12-18.

Contributions

All group members did approximately the same amount of work throughout the semester.