Westminster College: A Retention Prediction Model

Ryan P. Smith

Northwest Missouri State University, Maryville MO 64468, USA
S547012@nwmissouri.edu

Abstract. The abstract should briefly summarize the contents of the paper in

Keywords: higher education · retention · data analytics · data science

1 Introduction

At Westminster College, the Office of Institutional Effectiveness is responsible for compiling and analyzing data at our institution. This project will focus on the domain of higher education, and more specifically, we will be examining retention. The domain of higher education is being examined in this project as it can be utilized within our department in the future, and hopefully help provide more insight into retention factors at our college.

The data for this project will be sourced from Westminster College. We will be utilizing a couple of our databases from both our Enrollment Customer Relationship Management tool (CRM) Slate by Tehcnolutions, and our main SIS (Student Information System) which is provided by Jenzabar. All the student data used will be anonymized to ensure that there is no personal identifiable information used. The data will be compiled from the many tables into a useable dataset for this project. Data features that will be included in the dataset will be some biographical such as gender, commuter/resident, athlete. Some other data features will be items such as ACT score and high school GPA. This project will look to take those factors, create a multi-variable regression model to predict the likelihood that a student at Westminster College will be retained or not.

1.1 Defining the Problem

For this project, we want to see if retention can be predicted at Westminster College based on the selected data points and a machine learning model. There are multiple reasons why this is an important issue to look at and examine. First there appears to be a shrinking pool of high school graduates as we approach 2030. [5] This means that if there are fewer students in the graduating high school classes to recruit, there will be more competition between colleges and universities to enroll these students. If this leads to lower enrollment at an institution, that means that retention will become even more important to retain the students that you are able to enroll.

We can also see that enrollment has currently been dropping overall for the past several years. [4] The National Student Clearinghouse also illustrates this. We can see the enrollment declines among the different higher educations types, with a slight uptick actually showing in community colleges. [2] Since Westminster College has two prominent sources of revenue, donations and enrollment, it will be important to retain students at a higher percentage than before, if the enrollment of students continues to decline.

1.2 Goals of this Research and Methods to explore

This project will be looking to successfully predict if a student is at risk of being retained. We can use this data with our Student Success Center so that they can set up an intervention, or take the appropriate actions to ensure they can help the student persist. After gathering all the data to analyze, a model will be built and trained on train data, and then tested. After the model is completed, we can discuss the results and see how the model performed.

This project will be following the traditional Data Science Life Cycle. The steps are outlined below as:

1. Business Understanding - as seen above, this will be Predicting Retention at Westminster College.
2. Data Collecting - Data set to be produced from our CRM and SIS.
3. Data Cleaning - Missing data will be handled as well as mismatched data types.
4. Exploratory Data Analysis - Here we will begin to find trends and groups in the data.
5. Model Building - Here the model method will be selected, and then trained and tested.
6. Results - Visualizations will be created, conclusions will be formed and discussed.

The key components that will be focused on for this project will be data from previously enrolled students. We will be focusing only on students that attended Westminster College, and utilize the data that we have access to for each student. Some items may be missing such as ACT from recent years. Enrollment recently went to ACT optional, so there are more missing ACT scores than from the past, as more students are choosing not to submit those for consideration. We will look at including and not including ACT scores as another trend has been the decline in overall ACT scores across the country [3], and the state of Missouri [1] (19.5 and 19.8 respectively). This could be an interesting limitation, so we will try looking at retention with and without this feature.

2 About the Data

Data was collected, cleaned, and analyzed for this project. It came directly from two databases from Westminster College. These two databases come from the

Student Information System (SIS) which is provided by Jenzabar. The other is our Enrollment Customer Relationship Management (CRM) called Slate, by Technolutions. The data was pulled directly from a combination of these two databases, compiled into a usable dataset within a csv file, cleaned using Python, all before being analyzed.

2.1 Data Sourcing

The data was sourced from two databases on campus, the SIS and our Enrollment CRM. This data is structured data stored within these two databases. All tables being used to source our data are able to be joined together based on the student ID. The data was sourced directly from the databases, and the results were written to a .csv file to be imported into a Python script for cleaning and initial data exploration. There was no scraping needed to gather this data, it was collected with a SQL script using SQL Server. All tables and the two databases were able to be joined together with the primary key of student ID.

When looking back at students entering from the year 2010, there are 2605 records in the data set, when expanded to the year 2000, the amount of records increased to 5265. This project utilized the latter to have more records for analysis. For this project, the data fields were limited to 7 total. The fields in the data set along with their respective data type will be as follows:

1. Unique Identifier: Numeric Integer - Used to identify a unique student within the data set.
2. ACT Score: Numeric Integer - The highest ACT score that a student submitted with their enrollment.
3. Resident/Commuter: Character - denotes is a student resides on campus or commutes to campus
4. Gender: Character
5. Athlete: String - A sport will denote whether the student is an athlete or not (converted to a Y or N)
6. High School GPA: Numeric Float - a float numeric value to show the GPA from the students graduating High School.
7. Graduation Data/Exit Date: date - Currently a data field for both columns, these were combined along with exit reason to establish if a student graduated and was retained or withdrew and was not retained.

Overall, the data is very clean. There are some missing values in ACT scores, which was expected since we became an ACT optional reporting school in terms of enrollment. Looking through the data there are several other missing values, but not enough to have to cut it out. During the cleaning phase those issues will be addressed. There are two date fields that are formatted as YYYY,mm,dd, however, those fields were combined into a single field to determine if the student was retained (graduated) or not (withdrew). There were no bad characters found in the data set.

All of the data compiled from the databases was executed and written to a .csv document. With this format, the data was taken and using various Python libraries, cleaned and explored (both examined more in the upcoming sections). Also, as mentioned there are a couple items where were combined from the elements from the database to make a single feature within the dataset. After collecting and compiling all the data, the next section will cover how the data was cleaned.

2.2 Cleaning the Data

After collecting all the data, it was exported from the SQL Server environment into a .csv file. A Python script had been written to pull the .csv file into a pandas data frame for further analysis. After the data was imported into a dataframe, the dataframe was printed and also looked at the columns, and described the data. That can be seen in the image figure 1.

When looking at the data itself the cleaning looked minimal. However, using Python there were a few things discovered that needed to be cleaned. The first cleaning that was discovered was the duplicate values. The though going in was there would be no duplicates, however, there were some as the ACT test switched formats and some students had taken both tests, and had two highest scores recoreded. The script that was used to remove those were:

$$data.drop_duplicates(inplace = True) \tag{1}$$

Once the duplicates were dropped, there were 5063 records remaining in the data set still with 7 the features.After getting the duplicates removed from the data set, the next thing to check was the null values. There were two columns that contained null values, and they were ACT Scores and High School GPA. A couple of scripts were ran against the datafame to see what columns contained the null values, as well as what percent of our data had a null in each of those columns. They can be seen in figure 2.

Below is a table showing the results of the null values in the data. Again we can see the only two columns were ACT Score and High School GPA. They had 17.28 percent and 22.71 percent respectively in null values.

Table 1. Null Data Percentage by Feature

Feature	Percent Null
ID NUM	0.00%
Gender	0.00%
Res Communter Sts	0.00%
ACT SCORE	17.27%
ACT Score Received	0.00%
HS GPA	22.71%
Athlete	0.00%
Retained	0.00%

After identifying the null data, it was decided to use the mean value to fill in the missing values for both the ACT Scores and the High School GPA. The mean was selected as there was not a wide range of outliers being that ACT Scores and GPAs are relatively narrow and restricted in range. The script to replace the null values can be seen below.

```
data.act_scores.fillna(data.act_scores.mean()).isnull().any()
```

Once the two columns had their null values replaced with the mean, all the columns data was complete. The duplicates were removed, the nulls were replaced with the mean of their respective columns, and there were no bad characters in any column. Below we can see the features that will be analyzed after the cleaning of the data.

Table 2. Features Used After Data Cleaning

Feature	Type
ID NUM	Int
Gender	Character
Res Communter Sts	Character
ACT SCORE	float
ACT Score Received	Character
HS GPA	float
Athlete	character
Retained	character

The features themselves are defined as below.

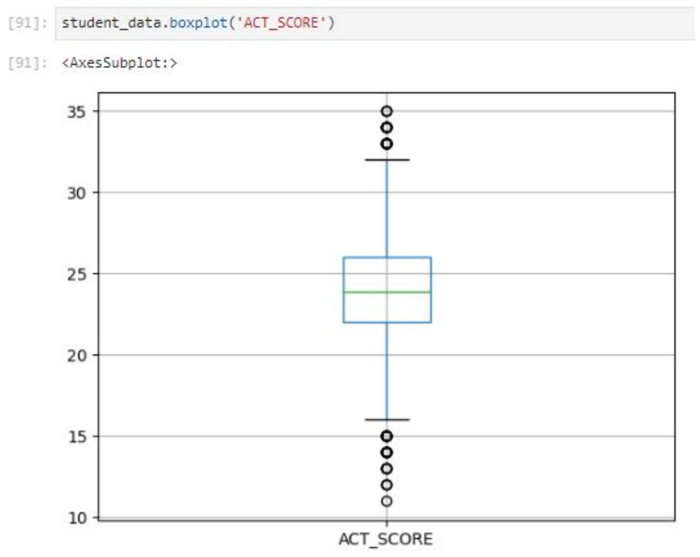
1. Unique Identifier: Used to identify a unique student within the data set.
2. ACT Score: The highest ACT score that a student submitted with their enrollment.
3. Resident/Commuter: denotes is a student resides on campus or commutes to campus
4. Gender: A Students self reported gender
5. Athlete: Whether or not a student is an athlete
6. High School GPA: gpa from high school
7. Retained: Whether or not a student graduated and was retained at the college.

After cleaning the data we will be using 6 of the features above (ID number is just an identifier) along with the 5063 entries to analyze the data. The data will be used to see if it can predict whether a student will be retained or not at Westminster College. The dependent variable is going to be whether or not a student was retained through gradation at the college. We will use the other features of ACT Score/ACT Provided, Gender, Athlete status, High school GPA, and Resident Status to make that prediction and they will be the independent variables. ACT Scores and ACT Provided will be used in separate models to see if scores affect retention, or if having a score at all is effective. The next section will show some basic discover and exploration of the data.

2.3 Exploratory Data Analysis

After the initial cleaning of the data, we can now start looking at what the data is showing us. To start, several Python libraries will be utilized to begin the data exploration. The libraries being utilized are the following, pandas, numpy, scipy - stats, seaborn, and matplotlib.pyplot. As stated in the previous section, the data was pulled into a pandas dataframe to be cleaned. After the data was cleaned, it became a new dataframe with the clean data. The first thing that was looked at was the distribution of the score data for both ACT and High School GPAs. Box-plots and Histograms were created for both categories.After the initial creation of the box plots, it was revealed that there were a handful of 0's that had been entered for High School GPA. That data was cleaned again as in the prior section, however instead of using the mean to fill in the null values, the mean was used to replace the values that had 0 as a value. After that data was fixed, the box plots and histograms were created again. Another change to the data was making every binary category a 1 or a 0. For Gender: 1 = Female, 0 = Male. For Resident: 1 = Resident, 0 = Commuter. For Retained: 1 = Retained, 0 = Not Retained. For Athlete: 1 = Athlete, 0 = Non Athlete. For ACT Received: 1 = Received, 0 = Not Received. These conversions were made to better see the correlations, and help with the upcoming modeling.

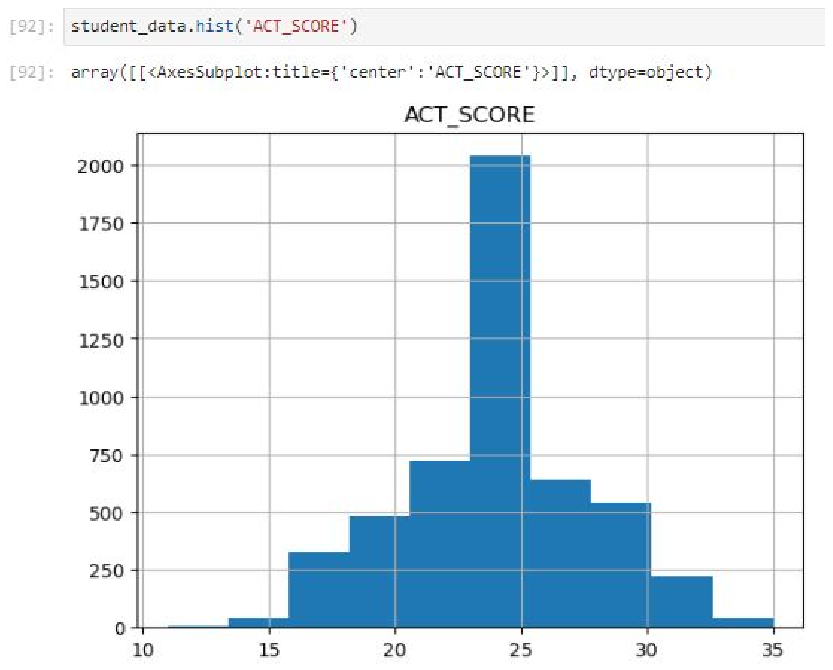
The first item that was checked was the ACT Scores. Below we can see the box plot shows a nice center falling between 22 and 26, we can see several lower scores as well as some higher scores, but nothing that is alarming in the ACT Score ranges.



```
[91]: student_data.boxplot('ACT_SCORE')
[91]: <AxesSubplot:>
```

Fig. 3. ACT Scores Box Plot

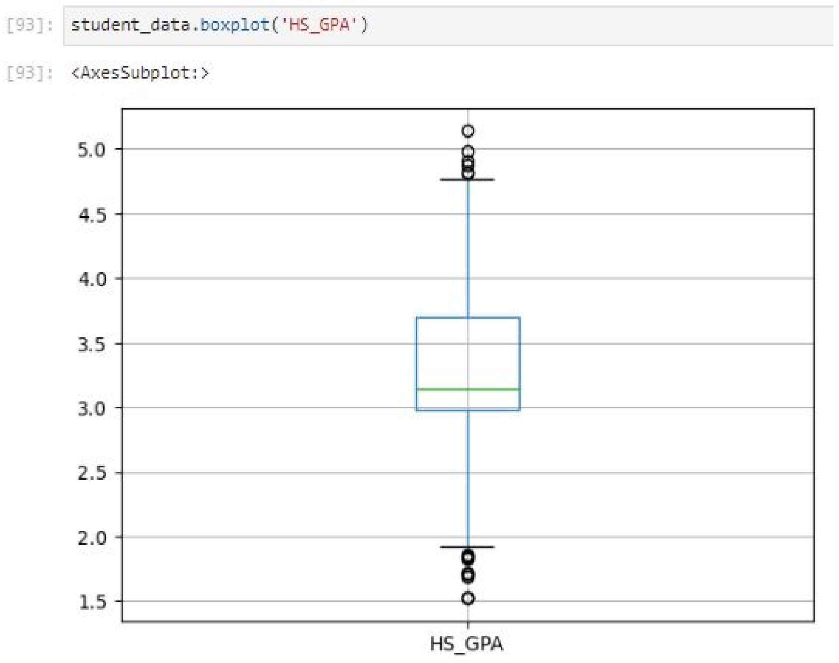
Similarly to the box plot, the ACT scores can be looked at to see their score distribution. This was shown in a histogram that can be seen below. As we can see from the histogram, there is a nice spread of data. It is a pretty normalized distribution, however, there is a large number at the center, as that appears to be where the null values that were replaced with the average will fall. This gives us a little larger group there, that may have been distributed a little better had the student reported their score.



```
[92]: student_data.hist('ACT_SCORE')

[92]: array([[<AxesSubplot:title={'center':'ACT_SCORE'}>]], dtype=object)
```

Fig. 4. ACT Scores Histogram

After looking at both the charts for ACT scores, there is a fairly decent distribution of data. The null values may be something that needs to be revisited, but since it was under 20 percent of the data, the mean will be used for the initial model. After looking at the ACT scores, the same graphs were completed to look at High School GPAs. As mentioned previously, the first box-plot showed there were some 0 values entered for GPA, so those were cleaned up before the new chart was produced. Below is the box plot for the High School GPAs.

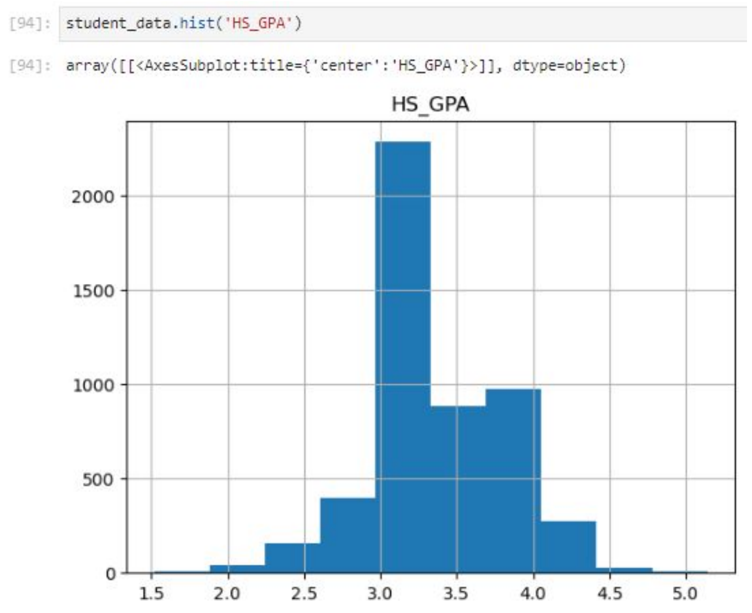


```
[93]: student_data.boxplot('HS_GPA')

[93]: <AxesSubplot:>
```

Fig. 5. High School GPA Box Plot

Again the box plot for GPA is pretty normal. There are some lower GPAs that were recorded, but nothing that falls outside of the normal GPA range. There were some low GPAs admitted that went through a committee, and admitted on other , so all of these are expected. On the high end there are some scores that fall outside of the typical 4.0 scale. These are also to be expected sometimes, as some High Schools have a weighted GPA scale, there was nothing else that seemed to jump out from this though.

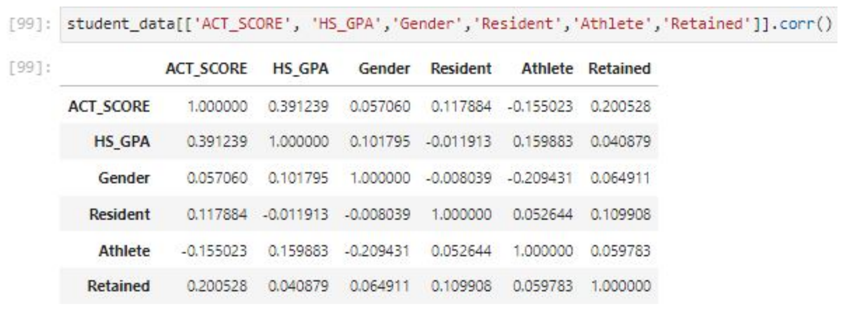
Below we can also see the distribution of High School GPAs by looking at its histogram. From this chart, we can see that the chart is a little left skewed, with a bit group falling where the mean was used to fill in for the nulls and the 0 values. There were a lot fewer lower GPAs as there is a minimum usually required to be admitted (this can be disregarded if the committee can see the student has other factors that may overcome a low GPA). The large amount of nulls may skew the model results, but for the first attempt, they will be left in as it accounted for less than 22 percent of the rows.



```
[94]: student_data.hist('HS_GPA')

[94]: array([[<AxesSubplot:title={'center':'HS_GPA'}>]], dtype=object)
```

Fig. 6. High School GPA Histogram

After looking at some of the basic distribution of the data from the score categories, there was a check on the correlation of the data. Each of these was completed twice. One was with ACT scores itself and the other was if an ACT was received at all regardless of the score a student received. The first chart is based on the ACT scores themselves. The ACT scores had a fairly positive correlation with the retention column. Each category interestingly enough has at least a small positive correlation with retention. Another high correlation is with ACT score and High School GPA itself.



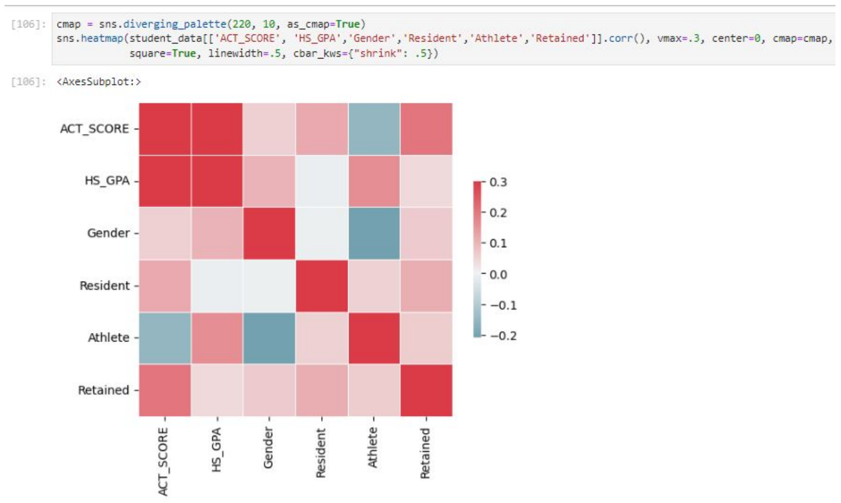
```
[99]: student_data[['ACT_SCORE', 'HS_GPA','Gender','Resident','Athlete','Retained']].corr()
```

[99]:

	ACT_SCORE	HS_GPA	Gender	Resident	Athlete	Retained
ACT_SCORE	1.000000	0.391239	0.057060	0.117884	-0.155023	0.200528
HS_GPA	0.391239	1.000000	0.101795	-0.011913	0.159883	0.040879
Gender	0.057060	0.101795	1.000000	-0.008039	-0.209431	0.064911
Resident	0.117884	-0.011913	-0.008039	1.000000	0.052644	0.109908
Athlete	-0.155023	0.159883	-0.209431	0.052644	1.000000	0.059783
Retained	0.200528	0.040879	0.064911	0.109908	0.059783	1.000000

Fig. 7. ACT Score Correlations with Categories

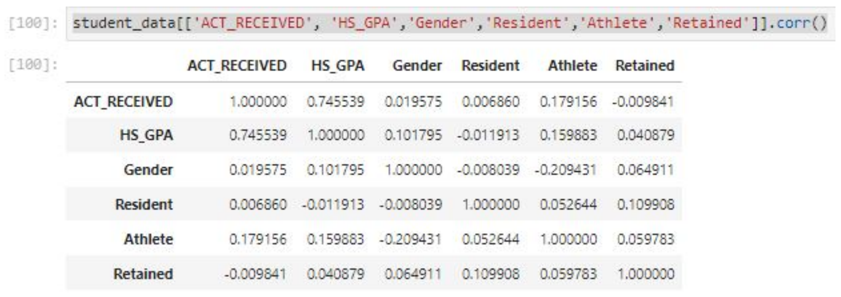
We can look at a heat chart of the data to give us a better visual of the data being correlated. The deeper the red in the following charts shows more of a positive correlation between the features. Again if we look at the last column which is if a student was retained or not, we can see at least a slight correlation with every category, with the darkest being the ACT Score.



```
[106]: cmap = sns.diverging_palette(220, 10, as_cmap=True)
       sns.heatmap(student_data[['ACT_SCORE', 'HS_GPA','Gender','Resident','Athlete','Retained']].corr(), vmax=.3, center=0, cmap=cmap,
                   square=True, linewidth=.5, cbar_kws={"shrink": .5})

[106]: <AxesSubplot:>
```

Fig. 8. ACT Score Correlations Heat Map

Again, the same practice was done with each category and ACT received was substituted for ACT Scores. This proved to provide some interesting initial insight. As we can see from both the chart and the heat map, there was a slightly negative correlation between ACT score being provided and retention, where all the other columns remained the same in regards to retention. However, ACT scores being received does have a high correlation with High School GPA itself. Below we can see the matrix for the correlations when an ACT score is received.



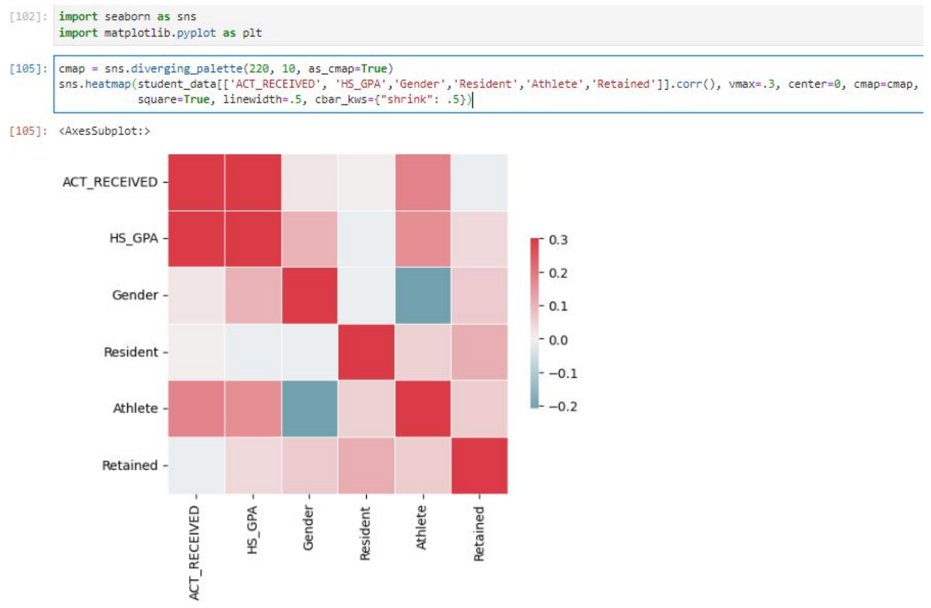
```
[100]: student_data[['ACT_RECEIVED', 'HS_GPA','Gender','Resident','Athlete','Retained']].corr()
```

[100]:

	ACT_RECEIVED	HS_GPA	Gender	Resident	Athlete	Retained
ACT_RECEIVED	1.000000	0.745539	0.019575	0.006860	0.179156	-0.009841
HS_GPA	0.745539	1.000000	0.101795	-0.011913	0.159883	0.040879
Gender	0.019575	0.101795	1.000000	-0.008039	-0.209431	0.064911
Resident	0.006860	-0.011913	-0.008039	1.000000	0.052644	0.109908
Athlete	0.179156	0.159883	-0.209431	0.052644	1.000000	0.059783
Retained	-0.009841	0.040879	0.064911	0.109908	0.059783	1.000000

Fig. 9. ACT Received Correlations with categories

As noted above, the heat chart can provide a better visual of the correlations. We can notice the slight blue on the ACT Received and the retention feature showing the slight negative correlation.



```
[102]: import seaborn as sns
       import matplotlib.pyplot as plt

[105]: cmap = sns.diverging_palette(220, 10, as_cmap=True)
       sns.heatmap(student_data[['ACT_RECEIVED', 'HS_GPA','Gender','Resident','Athlete','Retained']].corr(), vmax=.3, center=0, cmap=cmap,
                   square=True, linewidth=.5, cbar_kws={"shrink": .5})

[105]: <AxesSubplot:>
```

Fig. 10. ACT Received Correlations Heat Map

After are initial exploration, it has shown that it is more beneficial to start with ACT scores generally overall then if one was provided or not. We will still try and look at that, but for the initial model, we will focus on the scores themselves. All the code that is being used for the clean up and exploration can be found at the capstone github repository.After looking at his data, the next section will cover how the model will be selected, and the building, training, and testing of the model.

3 Predictive Modeling

Up to this point, the following pipeline has been utilized to get to and build the predictive model.

1. Gathering and sourcing the data - This was gathered from Westminster College local databases.
2. Data Cleaning - Data was cleaned to take care of null values, variables changed to a binary for analysis.

3. Exploratory Data Analysis - Multiple graphs and charts were created to start looking at trends that could be seen from the data, such as finding outliers, and how the data was grouped together.
4. Predictive Analysis (Current Step): Decide on the appropriate model to utilized, train and test the model, and start to interpret the results. This will be covered more in this section.
5. Results - Visualizations will be created, conclusions will be formed and discussed.

Multiple models were looked at to determine what would be the best model to ultimately use for the predictive analysis. The following will be some of the models that will be examined: Linear Regression Model, Decision Tree Model, Neural Net Model. In order to test out the various models, the data was split into a train set, and a test set. The library being utilized for this is the sklearn library. Below you can see the data being split into their respective sets. The test size of the data will be set to 20 percent.

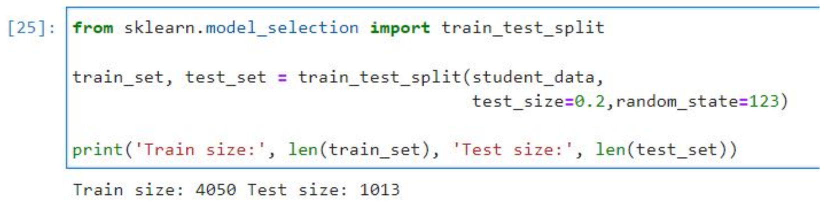

```
[25]: from sklearn.model_selection import train_test_split

      train_set, test_set = train_test_split(student_data,
                                             test_size=0.2,random_state=123)

      print('Train size:', len(train_set), 'Test size:', len(test_set))

      Train size: 4050 Test size: 1013
```

Fig. 11. This script takes the data set and breaks it into a training set and testing set. The test set will be 20 percent of the data.

Each model that will be shown in the upcoming section, will utilize the train split that was shown above.

3.1 Training and Testing the Models

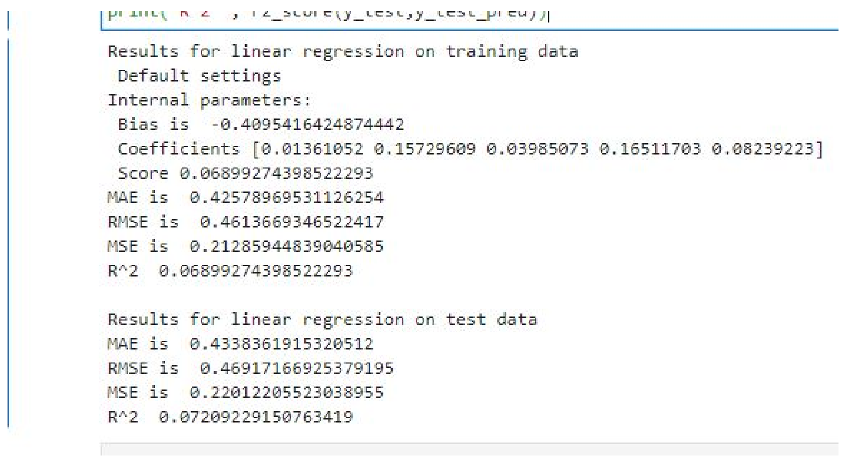
This section will examine the various models, and determine which will work best for predictive analysis on our data. The model should predict whether a student will be retained to the institution with the given variables of ACT score, high school GPA, resident, gender, and if they are an athlete.

The first model that was examined was a Linear Regression Model.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
X = train_set[['ACT_SCORE', 'HS_GPA', 'Gender', 'Resident','Athlete']]
y = train_set['Retained']
X_test = test_set[['ACT_SCORE', 'HS_GPA', 'Gender','Resident','Athlete']]
y_test = test_set['Retained']

lr_model = LinearRegression()
lr_model.fit(X,y)

###Predictoin of the model with training data

y_pred = lr_model.predict(X)
print('Results for linear regression on training data')
print(' Default settings')
print('Internal parameters:')
print(' Bias is ', lr_model.intercept_)
print(' Coefficients', lr_model.coef_)
print(' Score', lr_model.score(X,y))
print('MAE is ', mean_absolute_error(y, y_pred))
print('RMSE is ', np.sqrt(mean_squared_error(y, y_pred)))
print('MSE is ', mean_squared_error(y, y_pred))
print('R^2 ', r2_score(y,y_pred))

### Getting the performance of model on the test data

y_test_pred = lr_model.predict(X_test)
print()
print('Results for linear regression on test data')
print('MAE is ', mean_absolute_error(y_test, y_test_pred))
print('RMSE is ', np.sqrt(mean_squared_error(y_test,
y_test_pred)))
print('MSE is ', mean_squared_error(y_test, y_test_pred))
print('R^2 ', r2_score(y_test,y_test_pred))
```

The results from this model were not very appealing. Shown in the image below, we can see from the Score that the model only predicts correctly (in this case retention) about 7 percent of the time. The error scores are all very low, but again with the R squared score it is around 7 percent. Ideally this number would be much higher and approaching 1.



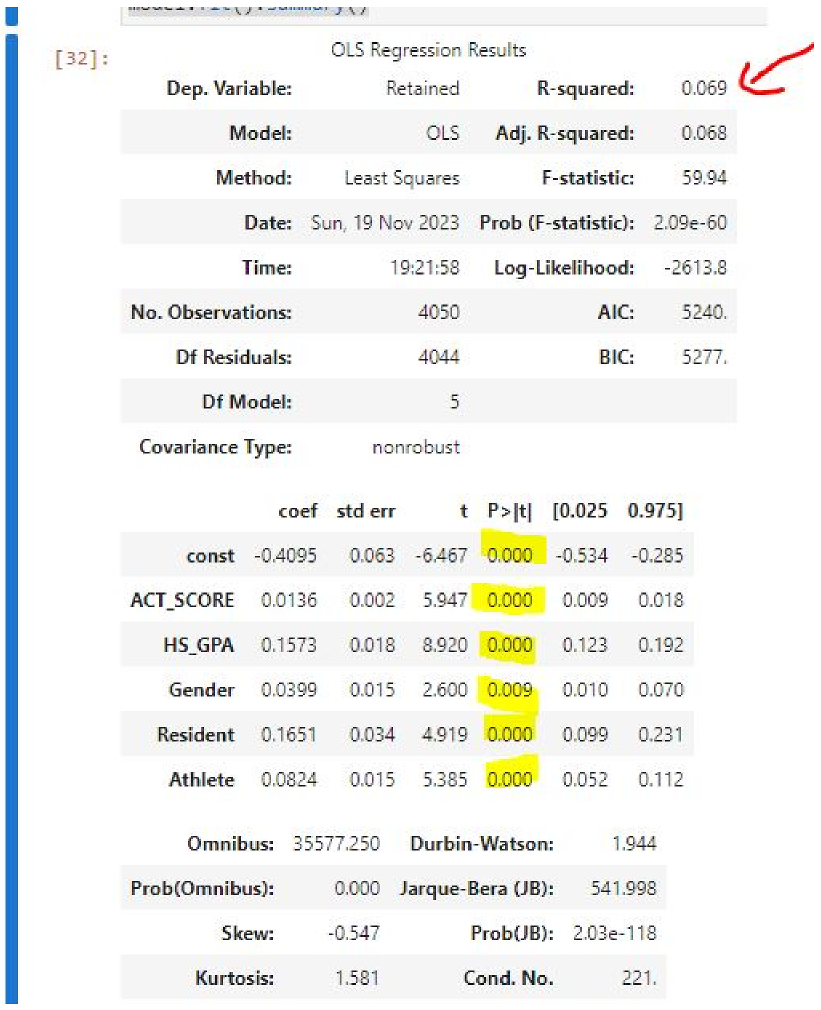
```
Results for linear regression on training data
 Default settings
Internal parameters:
 Bias is  -0.4095416424874442
 Coefficients [0.01361052 0.15729609 0.03985073 0.16511703 0.08239223]
 Score 0.06899274398522293
MAE is  0.42578969531126254
RMSE is  0.4613669346522417
MSE is  0.21285944839040585
R^2  0.06899274398522293

Results for linear regression on test data
MAE is  0.4338361915320512
RMSE is  0.46917166925379195
MSE is  0.22012205523038955
R^2  0.07209229150763419
```

Fig. 12. These results from the Linear Regression Model, shows the model only successfully predicts retention about 7 percent of the time.

Another linear regression model was tested from the statsmodel library. The following Python code was executed to get the results for this linear regression model.

```
import statsmodels.api as sm

X_sm = X = sm.add_constant(X)
model = sm.OLS(y, X_sm)
model.fit().summary()
```

Similarly to the model from the sklearn library this model provided nearly identical results that can be seen below. As shown in the image, the P values are all very close to 0, however, again the R-squared is right around 7 percent, which is not a very good result for the model.



```
[32]:
                        OLS Regression Results
     Dep. Variable:          Retained          R-squared:     0.069
             Model:               OLS     Adj. R-squared:     0.068
            Method:     Least Squares         F-statistic:     59.94
              Date:  Sun, 19 Nov 2023  Prob (F-statistic):  2.09e-60
              Time:          19:21:58     Log-Likelihood:   -2613.8
 No. Observations:              4050                 AIC:     5240.
      Df Residuals:              4044                 BIC:     5277.
          Df Model:                 5
   Covariance Type:         nonrobust

              coef  std err       t  P>|t|  [0.025  0.975]
      const -0.4095    0.063  -6.467  0.000  -0.534  -0.285
  ACT_SCORE  0.0136    0.002   5.947  0.000   0.009   0.018
     HS_GPA  0.1573    0.018   8.920  0.000   0.123   0.192
     Gender  0.0399    0.015   2.600  0.009   0.010   0.070
   Resident  0.1651    0.034   4.919  0.000   0.099   0.231
    Athlete  0.0824    0.015   5.385  0.000   0.052   0.112

       Omnibus:  35577.250    Durbin-Watson:     1.944
 Prob(Omnibus):      0.000  Jarque-Bera (JB):   541.998
          Skew:     -0.547          Prob(JB):  2.03e-118
      Kurtosis:      1.581          Cond. No.      221.
```

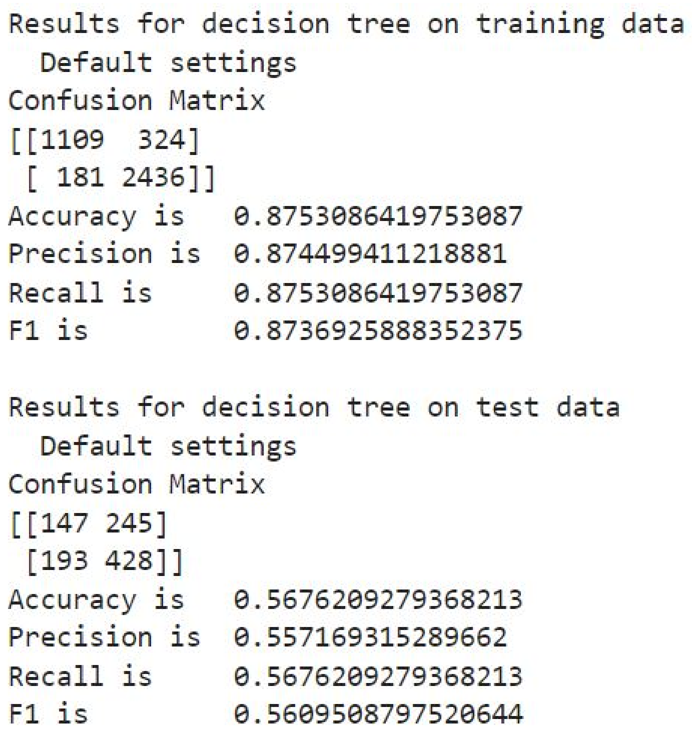
Fig. 13. These results from the linear regression model in the statsmodel library, shows the model only successfully predicts retention about 7 percent of the time.

After looking at two linear regression models, the next model that was examined was a Decision Tree model. This model will show us a confusion matrix, along with precision, accuracy, and an F1 score, which will be different from the prior linear regression model. Below is the Python code that was used to create this model.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import precision_score, recall_score
```

```
X = train_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y = train_set['Retained']

X_test = test_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y_test = test_set['Retained']

tree_model = DecisionTreeClassifier()
tree_model.fit(X,y)

y_pred = tree_model.predict(X)
print('Results for decision tree on training data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y, y_pred))
print('Accuracy is  ', accuracy_score(y, y_pred))
print('Precision is ', precision_score(y, y_pred, average='weighted'))
print('Recall is    ', recall_score(y,y_pred, average='weighted'))
print('F1 is        ', f1_score(y, y_pred, average='weighted'))
print()

y_test_pred = tree_model.predict(X_test)
print('Results for decision tree on test data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y_test, y_test_pred))
print('Accuracy is  ', accuracy_score(y_test, y_test_pred))
print('Precision is ', precision_score(y_test, y_test_pred, average='weighted'))
print('Recall is    ', recall_score(y_test,y_test_pred, average='weighted'))
print('F1 is        ', f1_score(y_test, y_test_pred, average='weighted'))
```

After this model was executed, the results looked a lot better, but still now the best. As we can see from the image below, the model performed very well on the training data set, however when it came to the test data set, the performance dropped quite a bit. The results were around 87 percent for accuracy, precision, recall, and F1 for the train set, but around 57 percent for the same items in the test set. This shows that the model is most likely overfitting the data.



```
Results for decision tree on training data
  Default settings
Confusion Matrix
[[1109  324]
 [ 181 2436]]
Accuracy is   0.8753086419753087
Precision is  0.874499411218881
Recall is     0.8753086419753087
F1 is         0.8736925888352375

Results for decision tree on test data
  Default settings
Confusion Matrix
[[147 245]
 [193 428]]
Accuracy is   0.5676209279368213
Precision is  0.557169315289662
Recall is     0.5676209279368213
F1 is         0.5609508797520644
```

Fig. 14. These results from the decision tree model, show an 87 percent success rate on the train data set, but just a 57 percent success rate on the test data set

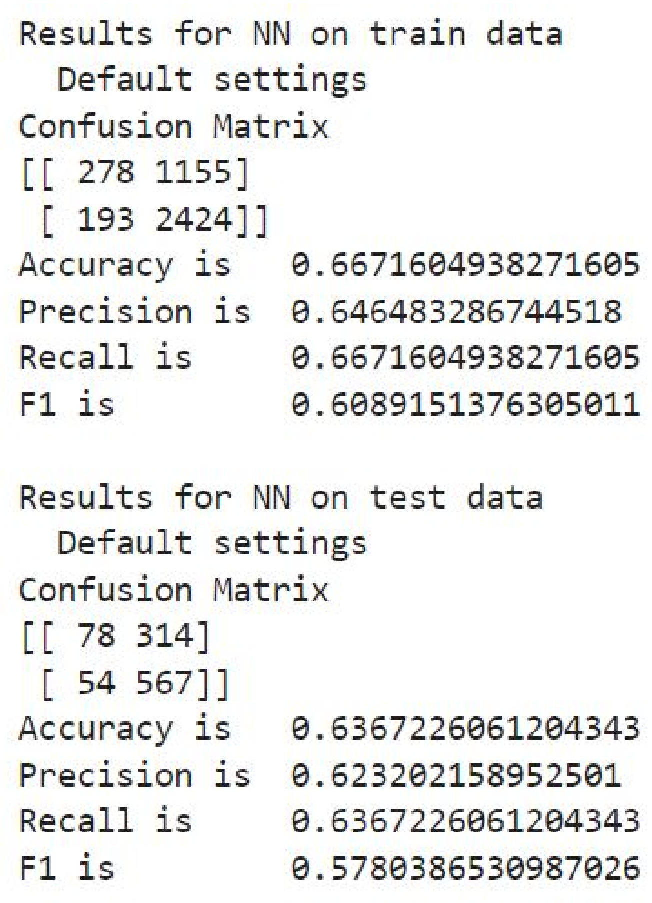
After examining the results from the decision tree, it can be seen that it also won't be the best model for predicting retention. Even though there was a promising success rate in the train data set, the test set revealed that it was most likely due to overfitting in the model.

The third model that was examined was a neural net model. It was also used from the sklearn library. The neural net model will display its results similar to the decision tree with a confusion matrix, and then displaying the accuracy, precision, recall, and F1 score. Below is the Python code that was used for this model.

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import precision_score, recall_score


X = train_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y = train_set['Retained']
```

```
X_test = test_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y_test = test_set['Retained']

nn_model = MLPClassifier(hidden_layer_sizes=(50, 25, 10),
                         solver='lbfgs')
nn_model.fit(X,y)

y_pred = nn_model.predict(X)


print('Results for NN on train data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y, y_pred))
print('Accuracy is  ', accuracy_score(y, y_pred))
print('Precision is ', precision_score(y, y_pred, average='weighted'))
print('Recall is    ', recall_score(y,y_pred, average='weighted'))
print('F1 is        ', f1_score(y, y_pred, average='weighted'))
print()

y_test_pred = nn_model.predict(X_test)
print('Results for NN on test data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y_test, y_test_pred))
print('Accuracy is  ', accuracy_score(y_test, y_test_pred))
print('Precision is ', precision_score(y_test, y_test_pred, average='weighted'))
print('Recall is    ', recall_score(y_test,y_test_pred, average='weighted'))
print('F1 is        ', f1_score(y_test, y_test_pred, average='weighted'))
```

Looking at these results in the below image, it can be shown that this model has performed the best to this point. For the neural net model, the accuracy, precision, recall were all around 67 percent with an F1 score at 61 percent. The test data set was slightly lower at 63 percent for the same items and an F1 score near 58 percent. This model, although still not performing the best, has provided the best results up to this point.



```
Results for NN on train data
  Default settings
Confusion Matrix
[[ 278 1155]
 [ 193 2424]]
Accuracy is   0.6671604938271605
Precision is  0.646483286744518
Recall is     0.6671604938271605
F1 is         0.6089151376305011

Results for NN on test data
  Default settings
Confusion Matrix
[[ 78 314]
 [ 54 567]]
Accuracy is   0.6367226061204343
Precision is  0.623202158952501
Recall is     0.6367226061204343
F1 is         0.5780386530987026
```

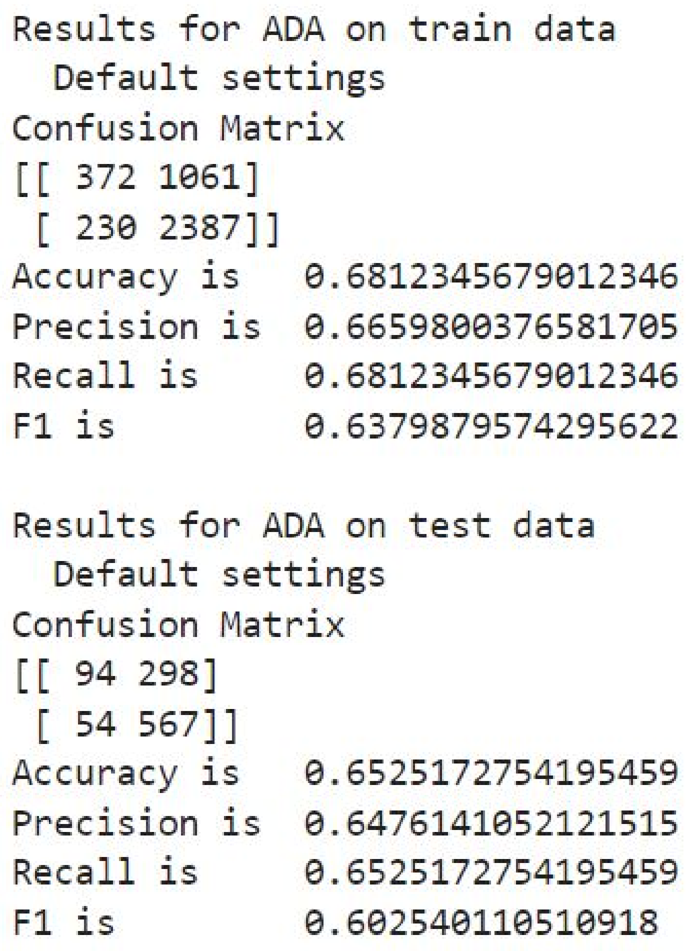
Fig. 15. The results above from the neural net model. The results are similar for both the train and test data sets, and is the best performing model so far. However, the model still only accurately predicts about 66 percent of the time.

After comparing the three models, it was decided to try one more model. This model was also comes from the sklearn library, and is the ADA boosted model. The results will also be displayed similarly to the two prior models. Below is the Python code that was used for the ADA boosted model.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import precision_score, recall_score

X = train_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y = train_set['Retained']
```

```
X_test = test_set[['ACT_SCORE','HS_GPA','Gender', 'Resident','Athlete']]
y_test = test_set['Retained']

ada_model = AdaBoostClassifier(n_estimators=150)
ada_model.fit(X,y)

y_pred = ada_model.predict(X)

print('Results for ADA on train data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y, y_pred))
print('Accuracy is  ', accuracy_score(y, y_pred))
print('Precision is ', precision_score(y, y_pred, average='weighted'))
print('Recall is    ', recall_score(y,y_pred, average='weighted'))
print('F1 is        ', f1_score(y, y_pred, average='weighted'))
print()

y_test_pred = ada_model.predict(X_test)
print('Results for ADA on test data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y_test, y_test_pred))
print('Accuracy is  ', accuracy_score(y_test, y_test_pred))
print('Precision is ', precision_score(y_test, y_test_pred, average='weighted'))
print('Recall is    ', recall_score(y_test,y_test_pred, average='weighted'))
print('F1 is        ', f1_score(y_test, y_test_pred, average='weighted'))
```

Again looking at the below results, it can be seen that this is also one of the best performing models. Again, it only correctly predicts retention about 67 percent of the time, like the neural net model, but it is a few percentages better in each category.



```
Results for ADA on train data
  Default settings
Confusion Matrix
[[ 372 1061]
 [ 230 2387]]
Accuracy is   0.6812345679012346
Precision is  0.6659800376581705
Recall is     0.6812345679012346
F1 is         0.6379879574295622

Results for ADA on test data
  Default settings
Confusion Matrix
[[ 94 298]
 [ 54 567]]
Accuracy is   0.6525172754195459
Precision is  0.6476141052121515
Recall is     0.6525172754195459
F1 is         0.602540110510918
```

Fig. 16. The results from the above ADA boosted model. The results are similar for both the train and test data sets, and very similar, though slightly better than the neural net model

After running through these models and seeing their results on the training data set and the test data set, we can start to see what model is going to work best for our analysis. Further discussion of the model results will be displayed in the next section.

3.2 Determining the Best Model

As shown in the prior section, we looked at four different models. A linear regression model, a decision tree model, a neural net model, and an ADA boosted model. The linear regression model was quickly ruled out as it had a very low r-squared showing that it only could predict correctly on the data around 7 percent of the time.Below we can start looking at the results and compare the other three models to make the determination for the model. Since the remaining three models used the same output, a table was made for easier comparison of their results. Below each model will be shown with its accuracy and F1 score on both the train and test data sets.

Table 3. Results and comparisons of the models

Model	Training Features	Acc Train	F1 Train	Acc Test	F1 Test
Decision Tree	ACT, GPA, Gender, Resident, Athlete	.875	.874	.568	.561
Neural Net	ACT, GPA, Gender, Resident, Athlete	.667	.601	.637	.578
ADA Boosted	ACT, GPA, Gender, Resident, Athlete	.681	.638	.653	.603

From the chart above, it can be seen that the ADA boosted model performed the best overall, so that was the model that was selected for this retention analysis. Overall though it can be determined that none of the models performed great, but it could be beneficial to the school even if the model can predict 67 percent accurately if a student will be retained of not. After exploring several model options for our predictive model building, the results will be further examined in the next section. Along with the further discussion of results, additional discussion will be had to maybe determine what could help enhance our model going forward.

4 Results

4.1 Clear Summary of Results

What did we find out from our model?

4.2 Visualizations

Share some visuals from the results and show what the data says.

5 Discussion

5.1 Conclusion of Results

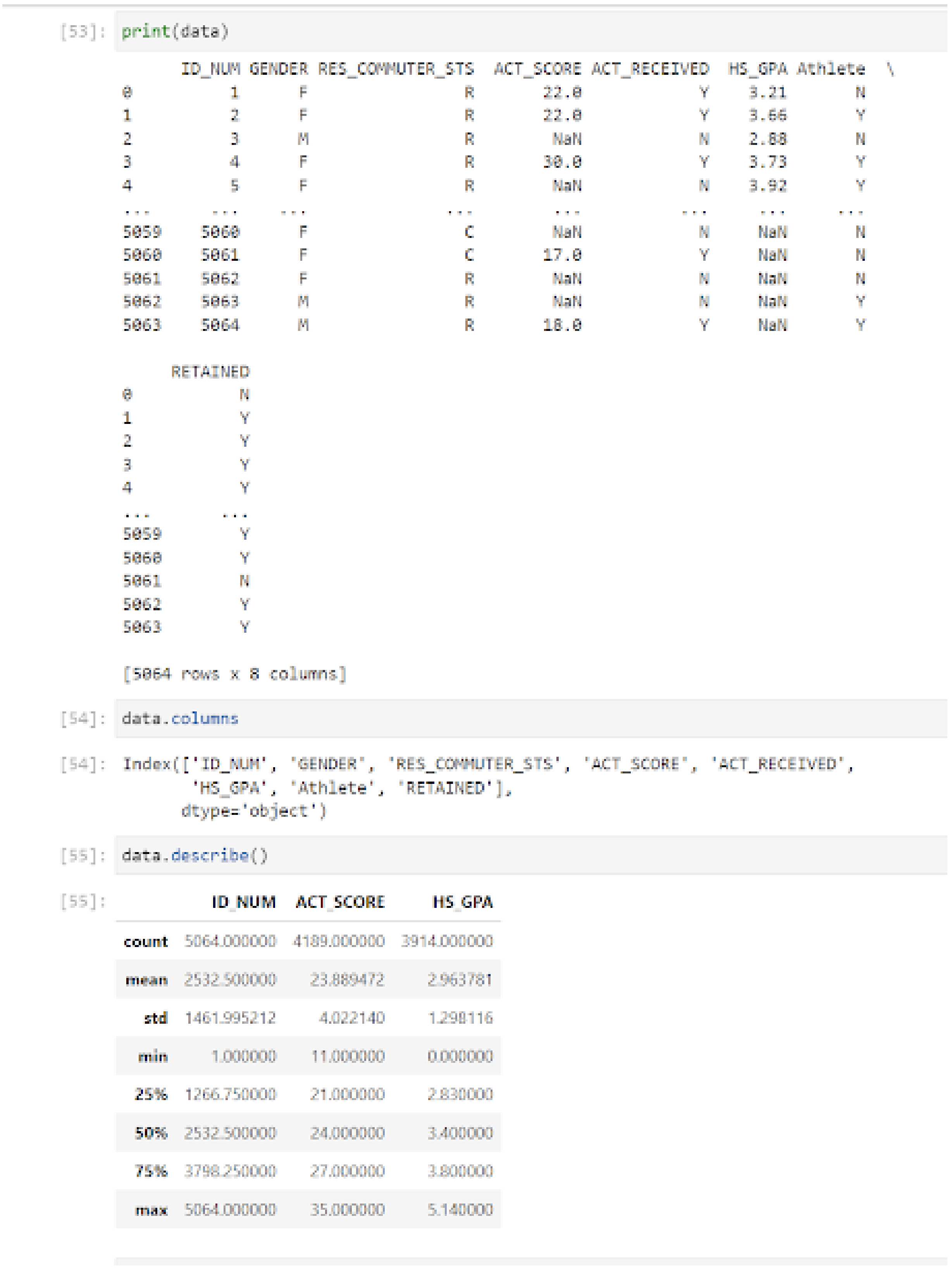
5.2 Limitations

5.3 Future Recommendations and Work

References

1. Act scores drop for 6th straight year, `https://www.northwestmoinfo.com/local-news/act-scores-drop-for-6th-straight-year/`

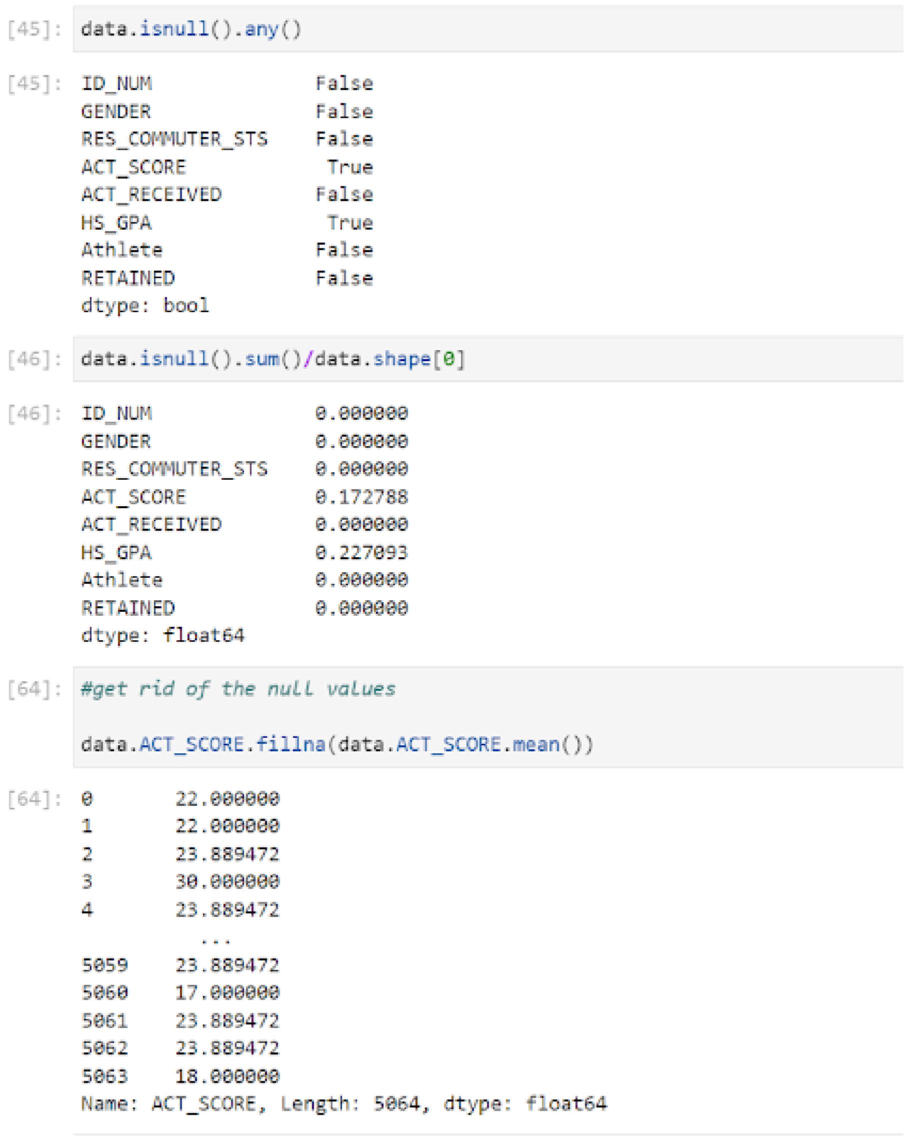
2. Current term enrollment estimates, https://nscresearchcenter.org/current-term-enrollment-estimates/
3. Castillo, E.: Act scores hit 30-year low, https://www.bestcolleges.com/news/act-test-scores-hit-30-year-low
4. Knox, L.: Leveling off on the bottom, https://www.insidehighered.com/news/admissions/traditional-age/2023/05/24/leveling-bottom
5. Seltzer, R.: Birth dearth approaches, https://www.insidehighered.com/news/2020/12/15/more-high-school-graduates-through-2025-pool-still-shrinks-afterward



```
[53]: print(data)

       ID_NUM GENDER RES_COMMUTER_STS  ACT_SCORE ACT_RECEIVED  HS_GPA Athlete  \
0           1      F                R       22.0            Y    3.21       N
1           2      F                R       22.0            Y    3.66       Y
2           3      M                R        NaN            N    2.88       N
3           4      F                R       30.0            Y    3.73       Y
4           5      F                R        NaN            N    3.92       Y
...       ...    ...              ...        ...          ...     ...     ...
5059     5060      F                C        NaN            N     NaN       N
5060     5061      F                C       17.0            Y     NaN       N
5061     5062      F                R        NaN            N     NaN       N
5062     5063      M                R        NaN            N     NaN       Y
5063     5064      M                R       18.0            Y     NaN       Y

     RETAINED
0           N
1           Y
2           Y
3           Y
4           Y
...       ...
5059        Y
5060        Y
5061        N
5062        Y
5063        Y

[5064 rows x 8 columns]

[54]: data.columns

[54]: Index(['ID_NUM', 'GENDER', 'RES_COMMUTER_STS', 'ACT_SCORE', 'ACT_RECEIVED',
             'HS_GPA', 'Athlete', 'RETAINED'],
            dtype='object')

[55]: data.describe()

[55]:
```

	ID_NUM	ACT_SCORE	HS_GPA
count	5064.000000	4189.000000	3914.000000
mean	2532.500000	23.889472	2.963781
std	1461.995212	4.022140	1.298116
min	1.000000	11.000000	0.000000
25%	1266.750000	21.000000	2.830000
50%	2532.500000	24.000000	3.400000
75%	3798.250000	27.000000	3.800000
max	5064.000000	35.000000	5.140000

Fig. 1. Looking at data in the dataframe, displaying the columns, and a basic describe



```
[45]: data.isnull().any()

[45]: ID_NUM               False
      GENDER               False
      RES_COMMUTER_STS     False
      ACT_SCORE             True
      ACT_RECEIVED         False
      HS_GPA                True
      Athlete              False
      RETAINED             False
      dtype: bool

[46]: data.isnull().sum()/data.shape[0]

[46]: ID_NUM               0.000000
      GENDER               0.000000
      RES_COMMUTER_STS     0.000000
      ACT_SCORE            0.172788
      ACT_RECEIVED         0.000000
      HS_GPA               0.227093
      Athlete              0.000000
      RETAINED             0.000000
      dtype: float64

[64]: #get rid of the null values

      data.ACT_SCORE.fillna(data.ACT_SCORE.mean())

[64]: 0       22.000000
      1       22.000000
      2       23.889472
      3       30.000000
      4       23.889472
                ...
      5059    23.889472
      5060    17.000000
      5061    23.889472
      5062    23.889472
      5063    18.000000
      Name: ACT_SCORE, Length: 5064, dtype: float64
```

Fig. 2. Columns that show null data, and the percent of each that are null within each one.