Analyzing Academics: Factors Affecting Student Performance

Landis Humphrey, Eric Burdick, Grant Chandler

[1] Champlain College, Burlington VT 05401, USA

Abstract. This report explores the relationship between a student and their academic performance, based on personal factors. The dataset was virtually unusable in the beginning and had to go through extensive cleaning. The cleaning and preprocessing were done using NumPy, and Pandas libraries, while the data visualizations were done with the matplotlib.pyplot library. Once the data cleaning and preprocessing were complete, we discovered, through data visualizations, that students with good academic behaviors had higher scores and that students who did not utilize their free time for school work effectively yielded lower scores. The analysis revealed positive correlations between academic-related variables and student scores, contrasting with negative correlations for non-academic traits. However, predictive modeling with Decision Tree and Random Forest classification algorithms yielded modest results, suggesting the complexity of predicting academic performance solely based on observable habits. While there is a clear connection between better study habits and better grades, it is a challenging process to accurately predict that through a supervised learning model.

Keywords: Academics, Data Analytics, Predictive Modeling

1 Introduction

This data analysis project was based on a dataset that was comprised of student data. This data ranged from academic features, such as the study time of a student, but also non-academic features, like the occupation of the student's parents. We wanted to see if student behaviors had an impact on their academic performance and explore the implications of this, as well as see if an accurate prediction model can be made from the features within the dataset.

1.1 Literature Review

One dataset we looked at was an Olympic performance dataset [1]. This data included all the stats of Olympic athletes back from 1896. The math showing the sports and seasons divided could have made the charts in two different sets each. This could be delved into further by finding another dataset including any Olympic medals won.

Eric found a dataset labeled Video Game Dataset [2]. It contained video games and their scores from Metacritic.com and it was a very interesting dataset to look at. The reason he chose this was because of the amount of details and data that could be looked at. Games were recorded by score, developer, platform, genre and so much more. This data goes all the way back to 1995 to 2024 as well bringing in both old and new entries. Some calculations he wanted to experiment with were average scores by genre and consoles, average score by year, also because games are released on multiple platforms, it allowed for the analysis of a single game and the difference in scores based on the console it was played on.

The dataset we ultimately chose was one Landis found, entitled Student Performance [3]. This dataset contained numerous features, explaining a student's scenario and lifestyle through different labels, such as 'studytime,' 'freetime,' and 'Dalc,' which contained ordinal data on weekday alcohol consumption. The data was also extremely compressed, with all data existing within one column. The group went forward with this set because we felt that we could display the data accurately, as well as clean and preprocess the data in a way that could yield a lot of correlations between features, ultimately to the target feature we chose, 'G3,' which represents the final score.

1.2 Contributions

Each group members researched possible datasets to use for our projects. We then met to talk about which one we would move forward with, and it ended up being one of Landis' sets. There were 30 features to evaluate for data

cleaning, so each group member was assigned 10 features for data cleaning. Landis ended up doing Grant's features in addition to his own. After initial data cleaning from member, Landis performed more cleaning, preprocessed all of the data and performed predictive modeling. Eric created the presentation, and Landis created the report. Both group members helped the other on their respective tasks when asked to do so. Each group member will have a part in presenting our presentation to the class.

2 Methods/Algorithms

To perform our data analysis, we had to use several features, methods, and algorithms that we learned during this course. The NumPy and Pandas libraries were essential to our projects, providing us the means to view, process, filter, and extract our data and identify trends. For our visualizations, we utilized matplotlib.pyplot and their plot() and bar() methods. These methods allowed us to display our results in an interpretable manner. There were certain methods and algorithms that were crucial to the project's success and we will explore them further below.

2.1 Split

In the beginning, our data was condensed into one row and one column. In other words, all features and values were combined into one value per row. However, each feature and value was divided by a semicolon. This allowed pandas.Series.str.split() to separate our data names and values using the semicolon that divided them. This method divided all feature names and feature values into their own respective pandas Series at which they could be used to populate a new pandas DataFrame, creating a properly distributed dataset.

In [3]:
```
data_cols = data_init.columns[0].split(';')

data_feat_split = pd.DataFrame(columns = data_cols)
data_feat_split
```

Out[3]:

school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3

0 rows × 33 columns

In [4]:
```
for i in range(len(data_init)):
    current_row = data_init.iloc[i][0]

    val_split = current_row.split(';')

    data_feat_split.loc[len(data_feat_split)] = val_split

data_feat_split
```

Out[4]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	"F"	18	"U"	"GT3"	"A"	4	4	"at_home"	"teacher"	...	4	3	4	1	1	3	4	"0"	"11"	11
1	GP	"F"	17	"U"	"GT3"	"T"	1	1	"at_home"	"other"	...	5	3	3	1	1	3	2	"9"	"11"	11
2	GP	"F"	15	"U"	"LE3"	"T"	1	1	"at_home"	"other"	...	4	3	2	2	3	3	6	"12"	"13"	12
3	GP	"F"	15	"U"	"GT3"	"T"	4	2	"health"	"services"	...	3	2	2	1	1	5	0	"14"	"14"	14
4	GP	"F"	16	"U"	"GT3"	"T"	3	3	"other"	"other"	...	4	3	2	1	2	5	0	"11"	"13"	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	"F"	19	"R"	"GT3"	"T"	2	3	"services"	"other"	...	5	4	2	1	2	5	4	"10"	"11"	10
645	MS	"F"	18	"U"	"LE3"	"T"	3	1	"teacher"	"services"	...	4	3	4	1	1	1	4	"15"	"15"	16
646	MS	"F"	18	"U"	"GT3"	"T"	1	1	"other"	"other"	...	1	1	1	1	1	5	6	"11"	"12"	9
647	MS	"M"	17	"U"	"LE3"	"T"	3	1	"services"	"services"	...	2	4	5	3	4	2	6	"10"	"10"	10
648	MS	"M"	18	"R"	"LE3"	"T"	3	2	"services"	"other"	...	4	4	1	3	4	5	4	"10"	"11"	11

649 rows × 33 columns

Fig. 1. Pandas split application

2.2 Drop

Once our data was properly distributed by feature, the group wanted to remove features that we deemed unhelpful for our analysis on our target variable. Once group members decided which features should not be kept for data analysis, pandas.DataFrame.drop() was utilized in order to remove those features from our dataset. Since features

are stored as columns in a pandas DataFrame, we would pass in a list containing the names of the column names we wanted to remove as well as the 'axis = 1' parameter so the method knows we want to drop columns instead of rows.

```
In [5]: def data_cutter(columns_names):
            return data_feat_split.loc[:,columns_names]

        def column_dropper(data_cut, column_names):
            data_cut.drop(columns = column_names, inplace = True, axis = 1)

In [6]: data_cols = data_feat_split.columns

In [7]: # Grant is in charge of processing features 1-10 ('school' - 'Fjob') (Landis ended up doing his features, Grant was unresponsive)
        # Eric is in charge of processing features 11-20 ('reason' - 'nursery')
        # Landis is in charge of processing features 21-30 ('higher' - 'absences')

        data_gc = data_cutter(data_cols[0:10])
        data_eb = data_cutter(data_cols[10:20])
        data_lh = data_cutter(data_cols[20:30])

In [8]: # Removing features we deemed unnecessary/not helpful
        column_dropper(data_gc, ['school', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob'])

        column_dropper(data_eb, ['reason', 'guardian', 'nursery'])

        column_dropper(data_lh, ['internet', 'romantic', 'famrel', 'Walc', 'health'])
```

Fig. 2. Pandas drop method

2.3 Strip

Some of our data values had extra characters that made them hard to process. More specifically, features 'sex,' 'activites,' and 'higher' contained binary strings values, but were stored with an extra pair of quotations marks. Those extra quotation marks proved to be troublesome for data preprocessing so pandas.Series.str.strip() was used to remove those extra characters so that they could be implemented in pandas.DataFrame.loc and NumPy masks, which we will discuss later.

```
In [17]: def strip_quotes(col_names):
             for col in col_names:
                 data_clean[col] = data_clean[col].str.strip('"')

In [18]: cols_to_strip = ['sex', 'activities', 'higher']

         strip_quotes(cols_to_strip)
```

Fig. 3. Pandas strip method

2.4 Masks

Several times, the data needed to be filtered through a certain value within a feature, either for replacement or visualization. A NumPy mask takes in a Boolean argument and applies it to a pandas Series or DataFrame. This way, NumPy masks create sub-datasets with rows that share the criterion, the nature of the applied mask. This feature was essential to this project, as it allowed the data to be focused through a lens based on a single feature, and allowed for analysis on a different feature than the one used for the mask.

```
In [ ]: def object_to_binary(col_names):
            for col in col_names:
                no_mask = data_clean[col] == 'no'
                yes_mask = np.invert(no_mask)

                data_clean.loc[no_mask, col] = 0
                data_clean.loc[yes_mask, col] = 1

In [19]: cols_to_binary = ['activities', 'ext_sup', 'higher']

         object_to_binary(cols_to_binary)
```

Fig. 4. NumPy mask & loc[] application

2.5 Loc

Masks were utilized for filtering the data, but sometimes the data that fulfilled the mask needed to be changed in some way. As shown in Figure 4, one way to do this is to create a mask and it would then be used in pandas.DataFrame.loc[] to alter the data. The first parameter takes in a Boolean value, like a NumPy mask, and the second parameter is the column that will be altered, followed by the new value that will replace the existing values in that column. This is a very useful method in data cleaning, preprocessing, and visualization.

2.6 Astype

While the split and strip methods were useful for breaking up our data by features, it also stored it as a string, ergo, an object. For modeling and visualizations, the values existing as type 'object' was problematic. This is where pandas.DataFrame.astype() was used to specify which datatype the data should be. For our analysis, with one exception, as seen in the figure below, our data was converted to type 'int' as it was all numerical by nature. This way, it could be easily accessed for our future work.

```
In [20]: for i in range(1, len(data_cols)):
             current_col = data_cols[i]

             data_clean[current_col] = data_clean[current_col].astype(int)
```

```
In [21]: data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 649 entries, 0 to 648
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sex         649 non-null    object
 1   age         649 non-null    int32
 2   traveltime  649 non-null    int32
 3   studytime   649 non-null    int32
 4   failures    649 non-null    int32
 5   activities  649 non-null    int32
 6   ext_sup     649 non-null    int32
 7   higher      649 non-null    int32
 8   freetime    649 non-null    int32
 9   goout       649 non-null    int32
 10  Dalc        649 non-null    int32
 11  absences    649 non-null    int32
 12  G3          649 non-null    int32
```

Fig. 5. Pandas astype() application

2.7 Outlier Treatment

Most values within any given feature were confined to a certain range, however, some features had values that were not representative of the overall trend. These data points are called outliers. Outliers can impair a model's accuracy and affect the interpretability and cleanness of visualization elements. To counteract this, outlier treatment is done. Using NumPy masks, a new dataset can be created using the criteria within the mask and this new dataset can be reapplied to itself, effectively dropping rows that contained the outlier data. Figure 5 shows the distribution of data within the 'absences' feature before the outlier treatment. Notice the significant number of outlined dots. Those are all the outliers that are not representative of the overall feature data. Figure 6 shows the distribution after outlier treatment has been performed. Notice again how there are significantly less outliers. It is important to keep a few outliers, however, as the boxplot only represents up to 75% of the data so keeping a few outliers can illustrate a wider range of data.

```
In [23]: plt.boxplot(data_clean['absences'])
         plt.show()
```

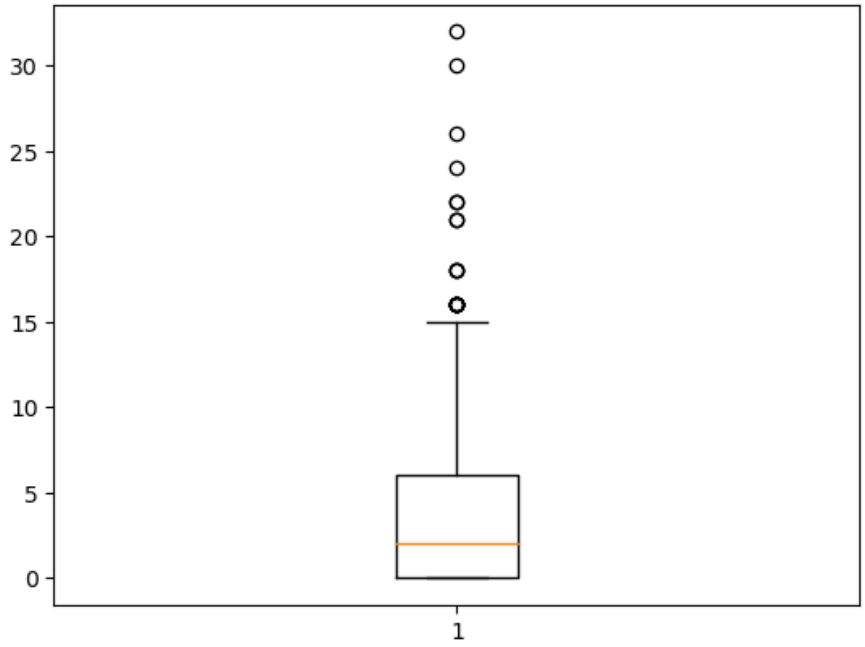

Fig. 5. Boxplot Visualization before Outlier Treatment on ‘absences’ Feature

```
In [29]: absence_outliers = data_clean['absences'] <= 20
         data_clean = data_clean[absence_outliers]

         plt.boxplot(data_clean['absences'])
         plt.show()
```

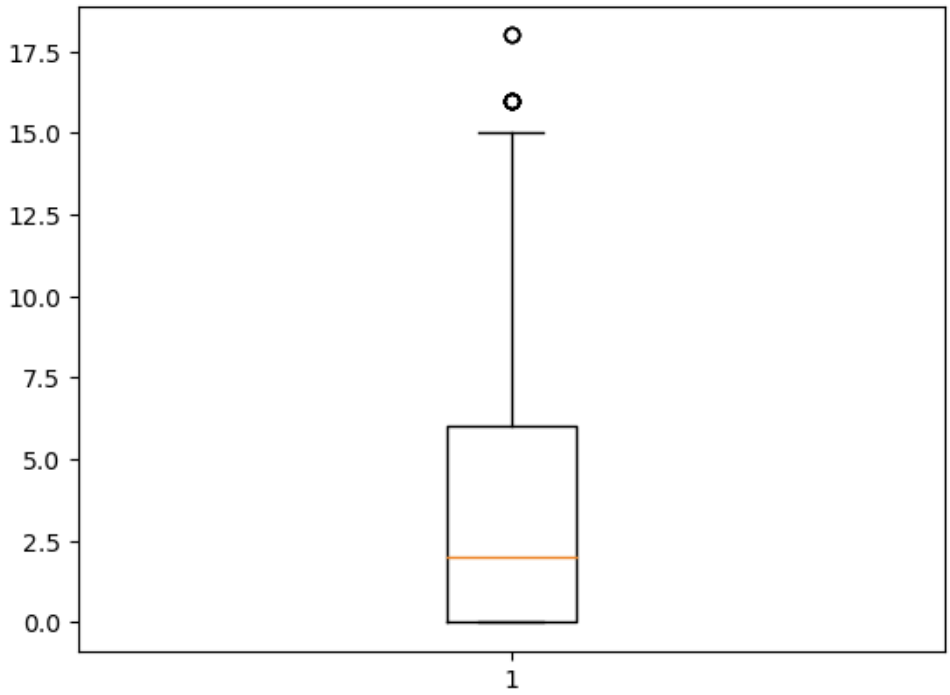

Fig. 6. Outlier Treatment with Boxplot Visualization on 'absences' Feature

2.8 Unique

For visual elements, it was important/insightful to see the relationships between two features within the dataset, especially if they pertained to our target variable. pandas.unique was a useful function that returned an array of all unique values within a feature. This was either used verbatim or, if the array returned by the method was not already sorted, an array was created of that length with sorted values. Figure 7 shows the implementation of this method. A significant number of our features were ordinal data, with values existing on a scale, starting at either 0 or 1. The unique method was used to identify how many unique values there were in the feature set, and depending on the starting point(0 or 1), a value was passed into the start_point parameter that dictated whether the sorted values started at 0 or 1. pandas.unique made it easy to determine how many subsets needed to be made and, in turn, made creating visualizations more convenient.

```
In [27]: def feat_y_avg_by_feat_x(start_point, feat_x, feat_y):
             vals = data_clean[feat_x].unique()
             if start_point == 0:
                 vals_sorted = np.arange(0,len(vals))
             else:
                 vals_sorted = np.arange(1,len(vals)+1)

             avgs = []

             for val in vals_sorted:
                 mask = data_clean[feat_x] == val
                 data_mask = data_clean[mask]

                 avg = round(data_mask[feat_y].mean(),2)
                 avgs.append(avg)

             plt.plot(vals_sorted, avgs)
```

Fig. 7. Line Plot Function Definition

2.9 Matplotlib.pyplot

As seen in Figures 5 & 6 and mentioned in the last line of Figure 7, we created visualizations in our project to help our analysis and understanding of our data. We used the library matplotlib, specifically the pyplot import for our data visualizations. This library can create a variety of different plots, from histograms, pie charts, line graphs, etc. The two graphs primarily used for our project were the bar and line graphs, called by bar() and plot(), respectively. Both plots were helpful in determining which features had a strong connection to our target variable, 'G3.'

2.9.1 Plot

matplotlib.pyplot.plot was the most basic graph used in our project. It takes in an independent variable, x, and a dependent variable, y, and shows the linear relationship of those two values. The values in x and y are often arrays and it is important for the x and y values to be of the same length. This plot shows if a correlation exists between the two variables. Figure 8 shows an example of an outputted plot, where there exists no distinct correlation.

```
In [34]: feat_y_avg_by_feat_x(0, 'absences', 'G3')
```

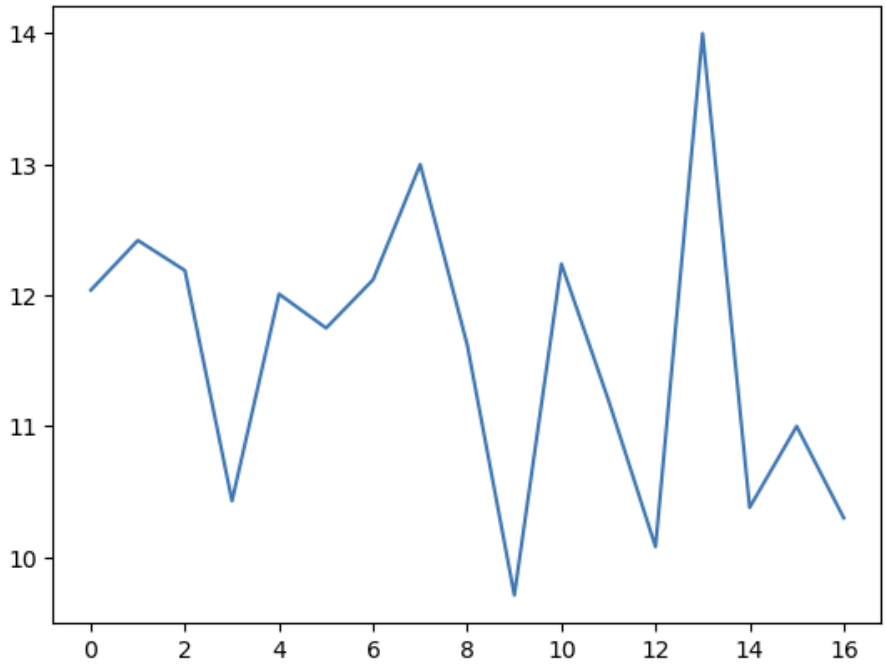

Fig. 8. 'absences' Feature vs 'G3' feature

2.9.2 Bar

matplotlib.pyplot.bar was another useful method we utilized. This graph worked well with our ordinal data, where the values existed on a scale or in a steadily increasing manner. The bar graph showed the relationship between that ordinal data and the frequency of another applied feature. This could either be the frequency of a value or the corresponding value on that x value. Figure 8 shows the average G3 score by each study time group and for this instance, the values displayed on the top of the bars are the corresponding values at each x value. These graphs are properly labeled as they were used a lot in our presentation elements. The line plots, on the other hand, were used for general understanding based on preexisting knowledge of the data.

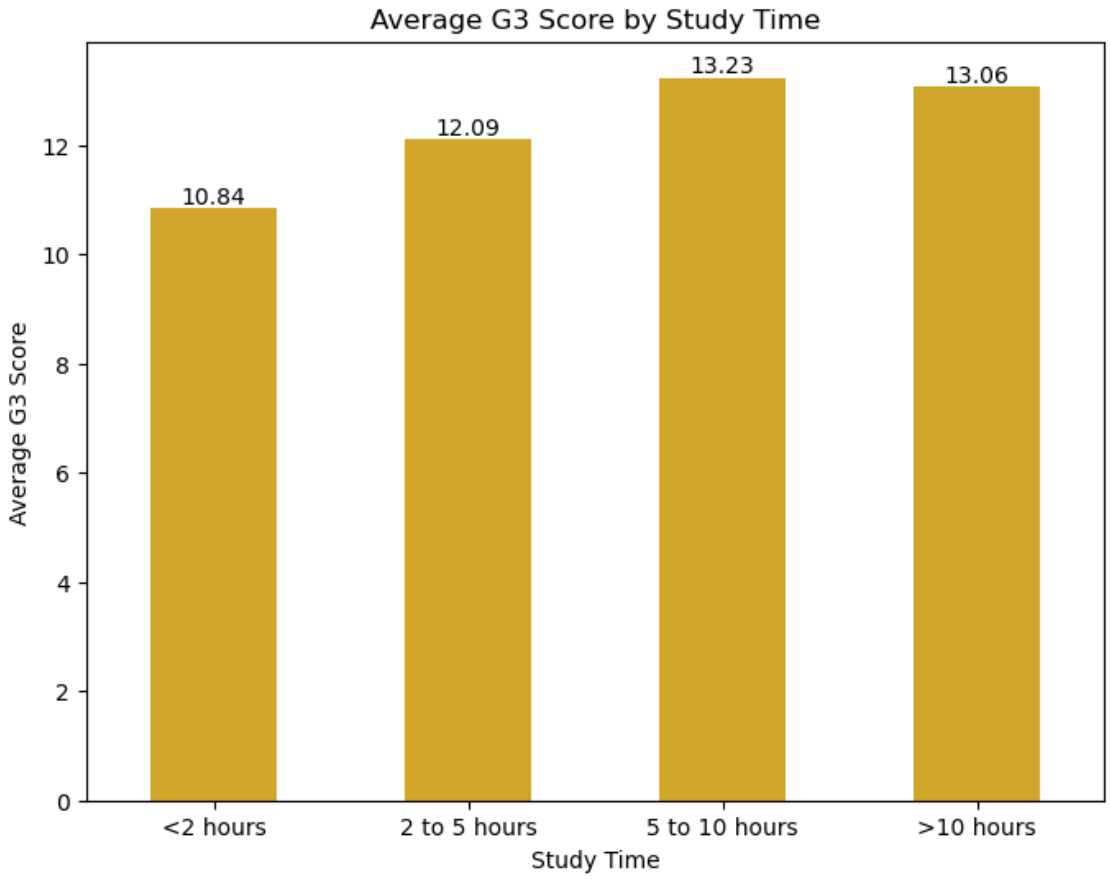

Fig. 8. Average G3 Score by Study Time Bar Graph

2.10 Classification Algorithms

An important part of this project was seeing if this data could be used in a model to predict a 'G3' score based on the given features. This involves the work of classification algorithms. To preface, classification is a machine learning technique used in supervised learning that predicts categorical output. For this algorithm to work with our data, the 'G3' feature got replaced by a 'Grade Group' categorical feature set. The difference between these two features are that 'G3' can have numerous unique values within it, while 'Grade Group' condenses that and groups rows based on an upper and lower bound. This is displayed in Figure 9. This makes 'Grade Group' a categorical feature that can then be used in a classification algorithm. Two classification algorithms were used in this project: Decision Trees and Random Forest.

In [61]:
```
lower_bound = 0
upper_bound = 4

group_num = 1

for i in range(4):
    range_mask = (data_pred['G3'] >= lower_bound) & (data_pred['G3'] <= upper_bound)
    data_pred.loc[range_mask, 'Grade Group'] = group_num

    lower_bound += 5
    upper_bound += 5

    group_num += 1

data_pred.drop(columns = 'G3', inplace = True)
```

Fig. 9. 'G3' to 'Grade Group'

2.10.1 Decision Tree

Decision Trees are a useful and versatile classification method. It utilizes a hierarchical tree-like structure, built upon nodes. Each node represents a point where the data splits on a value from a single feature. It looks at the data based on the criteria specified at that node and divides the data accordingly. Each feature can only be used as decision criteria once to prevent an overdependence on a select number of features. Because of this, however, the more features a dataset has, the more complex the decision tree is.

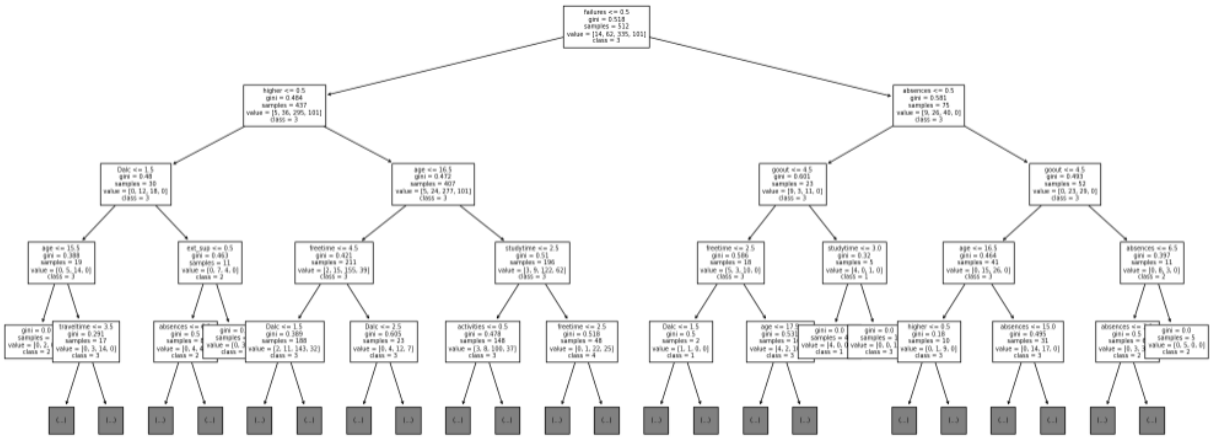

Fig. 10. Decision Tree based on All Features

2.10.2 Random Forest

Decision trees are a powerful classification technique. However, they have a big problem: decision trees are prone to overfitting the data. To counter this, a random forest can be used. Random Forest is a series of different decision trees running on different features in the dataset. Random forest performs better than single decision trees on datasets with larger feature sets as the algorithm spreads the features out amongst the many working decision trees. This solves single decision trees' problem of overfitting and provides better accuracy overall.

3 Results

Through the data analysis, we found that features that had to do with academics had positive correlations with our target variable, 'G3.' For example, as study time increased, students' G3 scores increased. On the other hand, features that were representative of non-academic traits and activities had negative correlations with 'G3.' What this data shows us is that students who had or spent more time to work on their school work, on average, had better scores. This supports the long-standing notion that focusing on school will land a student better grades than one who doesn't.

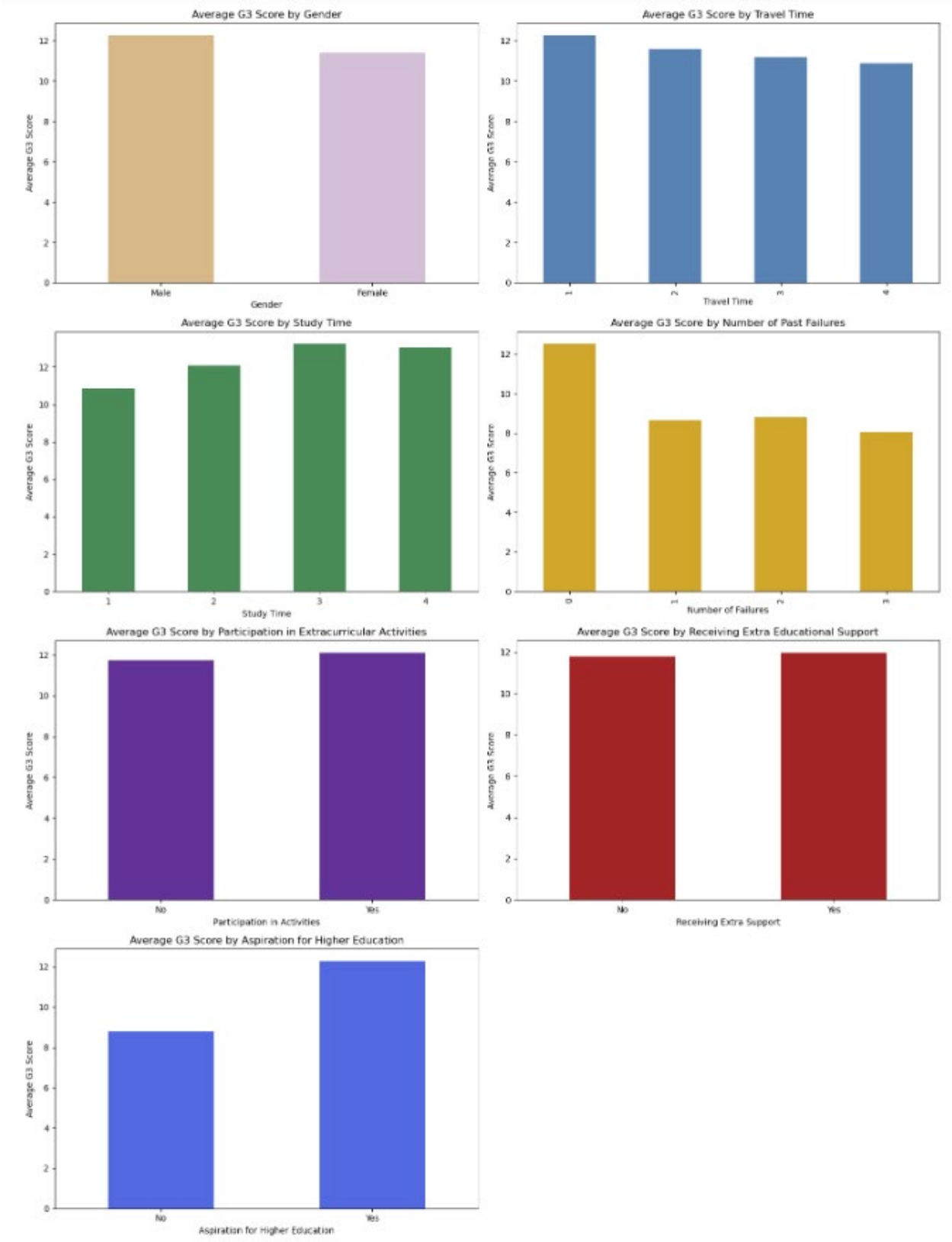

Fig. 11. Results of Data Analysis

As for the predictive modeling, our decision tree and random forest algorithms did not perform as well as expected. Modeling was done twice: once for all features and once for only features that showed a strong correlation to our target 'G3' feature. The expected outcome was the classifiers trained on the strongly correlated features to perform better than the one trained on all the features. However, as Figure 12 shows, the results were similar, but also with the algorithms in the all features classifier outperforming their respective counterparts in the strongly correlated classifier. There are a couple of reasons as to why this may be. For one, the data might not have as many strongly correlated features as we may have thought or the data is not easily splitable on each feature. Overall, the Decision Tree and Random Forest classification algorithms yielded an average performance in predicting G3 based on the dataset.

Evaluation Metrics	All		Strongly Correlated	
	Decision Tree	Random Forest	Decision Tree	Random Forest
Confusion Matrix	[[0 0 2 0] [1 4 14 0] [2 10 51 16] [1 1 15 12]]	[[0 1 1 0] [0 4 15 0] [0 6 68 5] [0 1 23 5]]	[[0 1 1 1] [1 5 6 0] [1 12 59 14] [0 0 22 6]]	[[0 1 2 0] [0 4 8 0] [1 7 69 9] [0 0 26 2]]
Accuracy	0.52	0.6	0.54	0.58
Precision	0.52	0.55	0.53	0.51
Recall	0.52	0.6	0.54	0.58

Fig. 12. Evaluation Metrics for Classification Algorithms

4 Conclusion & Future Work

Our dataset showed the importance and impact that good academic habits can have on a student's overall performance. However, our predictive modeling also shows that it is hard to predict a person's academic performance based on their habits and mannerisms unless a more complex algorithm is used. Everyone works a little differently and there is no clear cut guide to academic success. However, our data analysis did show that, more times than not, spending more time on school will yield better scores and results.

For future work, there are many things that we could potentially work on. A lot of features were removed from the original dataset for our analysis. A lot of them had to do with familial status and relations, so it would be interesting to see the correlation between those features and academic performance. Additionally, we focused primarily on using 'G3' as the main dependent variable in our graphs and analysis, but it would also be interesting to see the relationships between non-target features. Lastly, decision tree and random forest algorithms did not display significant predictive power on this dataset so exploring other classification algorithms, such as Support Vector Machines(SVM), could yield better results.

References

1. https://www.kaggle.com/datasets/beridzeg45/video-games?resource=download
2. https://www.kaggle.com/datasets/krishd123/olympics-legacy-1896-2020
3. https://archive.ics.uci.edu/dataset/320/student+performance