An Investigation of a Student Performance Database using Foundational AI Techniques

Michael Simpson

14th October, 2024

1 Introduction

Many advanced applications of AI such as its use in healthcare, agriculture, or finance are all based on foundational principles, of which have their roots in mathematics. This project was a study of some of these principles and how they can be applied to a database; particularly, a student performance database that has recorded the scores of students in Maths from two Portuguese schools, as well as a variety of other features such as study time and the parent's occupation. One aim of this project was to identify some of the key features that were a good indicator of student performance (final grade), but a more general aim was to develop an enriched understanding of some foundational AI techniques by applying them to this real-world scenario.

Before applying these techniques, the data had to be understood by performing an Exploratory Data Analysis (EDA), which is a way of obtaining meaningful insights into a database by comparing descriptive values and constructing univariate and bivariate graphs, as well as other methods. This was done with the database provided which gave a general insight into which features were more associated with a higher or lower final grade. Next, a Singular Value Decomposition (SVD) was applied to the data as a form of feature reduction. The Principal Components were obtained through this method and it was tested whether the number of features were actually reduced while still retaining most of the information. A gradient descent model was then created to identify the optimal parameters that allow for accurate prediction of the target variable (final grade). Another model was created to optimise the hyperparameter values (learning rate and number of iterations). Finally, a Markov Decision Process was implemented by utilizing features known to increase a student's final grade. The results for each section of this project are laid out, as well as a discussion of the effectiveness of the methods used and their ethical implications as applied to the data.

Figure 1: Overview of the database columns

Attribute	Description	Data Type	Data
School	Student's School	Binary	"GP" – Gabriel Pereira "MS" – Mousinho da Silveira
Sex	Student's Sex	Binary	"F" – Female "M" – Male
Age	Student's Age	Numeric	15 – 22
Address	Student's home address	Binary	"U" – Urban "R" – Rural
Famsize	Family size	Binary	"LE3" – Less or equal to 3 "GT3" – Greater than 3
Pstatus	Parent's cohabitation status	Binary	"T" – Living together "A" – Apart
Medu	Mother's education	Numeric	0 – none 1 – Primary education 2 – 5th to 9th grade 3 – Secondary education 4 – Higher education
Fedu	Father's education	Numeric	
Mjob	Mother's job	Nominal	Evident in database
Fjob	Father's job	Nominal	
Reason	Reason to choose this school	Nominal	
Guardian	Student's Guardian	Nominal	
Traveltime	Home to school travel time	Numeric (mins)	1: <15, 2: 15 – 30, 3: 30 – 60, 4: >60
Studytime	Weekly study time	Numeric	1: <2, 2: 2 – 5, 3: 5 – 10, 4:>10
Failures	Number of past class failures	Numeric	n if 1 – 3, else 4
Schoolsup	Extra educational support	Binary	Yes or no
Famsup	Family educational support	Binary	Yes or no
Paid	Extra paid classes within course subject	Binary	Yes or no
Activities	Extra-curricular activities	Binary	Yes or no
Nursery	Attended nursery school	Binary	Yes or no
Higher	Wants to take higher education	Binary	Yes or no
Internet	Internet access at home	Binary	Yes or no
Romantic	In romantic relationship	Binary	Yes or no
Famrel	Quality of family relationship	Numeric	1 – very bad to 5 – excellent
Freetime	Free time after school	Numeric	1 – very low to 5 – very high
Goout	Going out with friends	Numeric	
Dalc	Workday alcohol consumption	Numeric	
Walc	Weekend alcohol consumption	Numeric	
Health	Current health status	Numeric	1 – very bad to 5 – very good
Absences	Number of school absences	Numeric	0 to 93
G1	First period grade	Numeric	0 to 20
G2	Second period grade	Numeric	0 to 20
G3	Final period grade	Numeric	0 to 20

1.1 Data description and preparation

The database used was a record of student performance in Mathematics from two different schools in Portugal. It contains 31 feature variables and 1 target variable (being the final grade G3), and a description of each can be found above in Figure 1. The attributes includes features that are demographic, social or school-related, and are given as either binary, numerical or nominal. Binary features only have 2 data points, for example, the school either being "GP" or "MS", which are shorthand for the school names. Nominal features are for attributes that can be categorised more than 2 times, such as the Mother's job which is either "teacher", "health", "services", "at home" or "other". The last type is numerical, which is used for features that have a certain order to the categorisation, such as "health" which ranges from 1 to 5 (very bad to very good).

Before any form of data analysation could be done, a few preliminary steps had to be completed. The first of which was to check whether the database contained any missing values, as the presence of these could have lead to inaccurate results. After importing the database as a Pandas dataframe, the line of code shown in Figure 2 was used to return whether any missing values were present in any columns. A sample of the return is shown in Figure 3 as a demonstration, but the rest of the columns returned False as well, indicating the presence of no missing values in the dataframe.

```
print(df_math.isna().any())
```

Figure 2: Code for checking values

```
school      False
sex         False
age         False
address     False
famsize     False
Pstatus     False
```

Figure 3: Columns denoted "False", indicating no missing values

The second part of data preparation involved encoding many of the features that allowed for the SVD to be performed. This was done using One Hot Encoding for the nominal features, and binary encoding for the binary features. One Hot Encoding meant that a new column was created for each categorical value within a nominal feature, and a True or False value was assigned for each row For binary encoding, each value in a binary feature was simply assigned True or False.

Mjob_at_home	Mjob_health	Mjob_other	Mjob_services	Mjob_teacher
True	False	False	False	False
True	False	False	False	False
True	False	False	False	False
False	True	False	False	False
False	False	True	False	False

Figure 4: Example of One Hot Encoded data

2 Methodology

2.1 Exploratory Data Analysis (EDA)

2.1.1 Descriptive Analysis

The following analysis included a graphical representation of the data as well as calculations of some descriptive values such an the mean and standard deviation. An overview of these descriptive values was found using the *describe* method on the dataframe. This was done to obtain a good initial understanding of the data.

Several different graphs were then plotted to gain an understanding of how different variables relate to the target variable (G3). To plot these, the libraries *matplotlib.pyplot* and *seaborn* were imported, which provided a way to create subplots, as well as density and box plots. These plots were used to observe the distribution of G3 over varying categories, such as between schools or different studying amounts per week. These graphs were also supplemented with specific mean and standard deviation calculations to allow for a more exact comparison between categories.

2.1.2 Hypothesis testing and Chi-Squared comparison

Although it is interesting to observe the differences in the means across categories (such as comparing schools), it is often difficult to conclude whether this is due to some natural sampling variability or because of a deeper reason. This is why a hypothesis test was implemented. Specifically, a z-test was used because the sample numbers were over 30 for each group within a category. A z-test can also only be done on normally distributed data, so the outliers (where G3 was equal to 0) were removed from the dataset. The data for both groups was then visualised using a density plot to observe whether they were normally distributed or not. The following null hypothesis was

then formed:

H_0 =There is no significant difference between the groups in terms of G3

Using the *ztest* function that is part of the *statsmodel* package, the p-value was then calculated for selected feature columns and compared to a selected significance level *alpha* of 0.05. If the p-value was less than 0.05, then the null hypothesis had to be rejected.

To further the exploration of this dataframe, a Chi-Squared analysis was implemented in order to compare categorical features. This was done using the *chi2 contingency* function from the *scipy* package. This function takes in a cross-tabulation table formed using *pd.crosstab* for two selected feature variables. From this, a p-value is retrieved and compared to *alpha*, being 0.05. If the p-value was lower than *alpha*, then it meant there was a significant association between the chosen feature variables.

2.2 Singular Value Decomposition

Often times a dataset can end up containing a large number of feature variables making it difficult to analyse, which is why Principal Component Analysis (PCA) can be incredibly useful in reducing this number while retaining most of the information. PCA can be done a number of different ways such as eigen-decomposition, but the method of choice for this project was Singular Value Decomposition. This method takes a matrix A of real values and decomposes it into 3 components shown below:

$$A = U * S * V_T$$

where U and V_T are both orthogonal matrices and S is a diagonal matrix. To obtain this, the pre-processed data was used (binary and One Hot encoded) and the feature variables were separated from the target variable (G3). The feature data was then standardised which set the mean to 0 and standard deviation to 1, allowing for better analysis. The code shown in Figure 5 was then used to decompose the data according to the above equation, where V_T revealed the Principal Components. Each Principal Component contains a percentage of the information in the database, so a 95 percent threshold was chosen to test how much it took to reach this.

```
U, S, Vt = np.linalg.svd(X_stan)
```

Figure 5: Python code used to implement SVD

2.3 Gradient Descent

Neural Networks are at the foundation of some very important branches of AI, but they have their roots in another foundational concept called Gradient Descent. It can be used to form an optimisation model that takes in feature data and returns target data that is close to the desired output. A function was created to implement this model. Firstly, the feature data was standardised the same way it was for SVD, then both the feature and target data were split into training and test data, where the training data covered 80 percent of the database. Initial weights were chosen along with hyperparameter values being the learning rate and number of iterations. Predicted values for the target variable were calculated in the function and compared to the actual values. This was used to calculate the cost function:

$$J(\theta) = (1 \div m) \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

where θ is the weight, m is the number of data points and $(\hat{y}_i - y_i)^2$ is the error. The gradient of this function with respect to the weights then gives the "direction" to lower the cost value. This is repeated multiple times until the cost function is minimised, resulting in optimised values for the weights.

The choice of hyperparameter values is also important as these can lead to very different results. Therefore, another function was implemented that iterated through an array of hyperparameter values to find the combination that would result in the most optimal value for the weights.

2.4 Markov Decision Process

The final foundation AI principle applied was the Markov Decision Process, which is a stochastic model for decision-making. It relies on a principle called the Markov Property which states that predictions of the behaviour of the system in the future depend solely on the current state of the system. Elements of the process include the States, Actions, Rewards and Transition Probability. To create a Markov Chain based on the dataset provided, the following elements were chosen, taking inspiration from some findings from the EDA:

States S:

- S(D): A final grade from 0 - 5
- S(C): A final grade from 6 - 10
- S(B): A final grade from 11 - 15
- S(A): A final grade from 16 - 20

Action A:

- A(SM): Study more than 5 hours per week
- S(SL): Study less than 5 hours per week

Reward R:

- R(up): +1 going up a grade
- R(same): 0 for staying the same grade
- R(down): -1 for going down a grade

A Transition Probability matrix was then constructed which contains information about the probability of transitioning from one state to another. At each step there is also a policy π which tells the system what the best action to take is that maximises the reward.

3 Results

3.1 Exploratory Data Analysis (EDA)

Figure 6 below shows a density plot of the distribution of each period score. Each one appears to be normally distributed except for "G2" and "G3" which have outliers at 0. These were removed later when doing the hypothesis testing. Figure 7 then shows a similar plot to the previous but in the

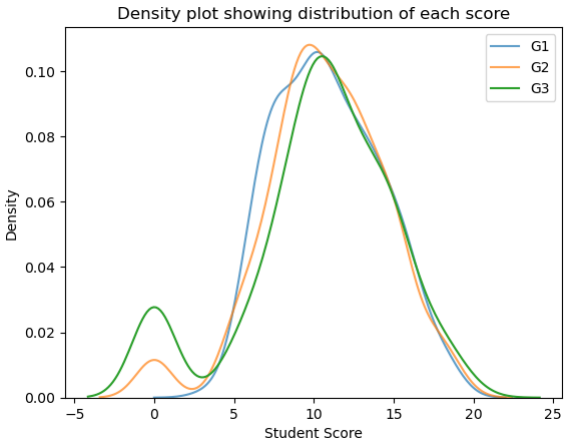

Figure 6: Density plot showing the distribution of each period score

form of a box plot and has been sorted by school. The mean scores were also calculated for each school, shown in Table 1, so from this and the plots it is clear that students from Gabriel Pereira performed better than students from Mousinho da Silveira.

	G1	G2	G3
GP	10.94	10.78	10.49
MS	10.67	10.20	9.85

Table 1: Means of each period score by school

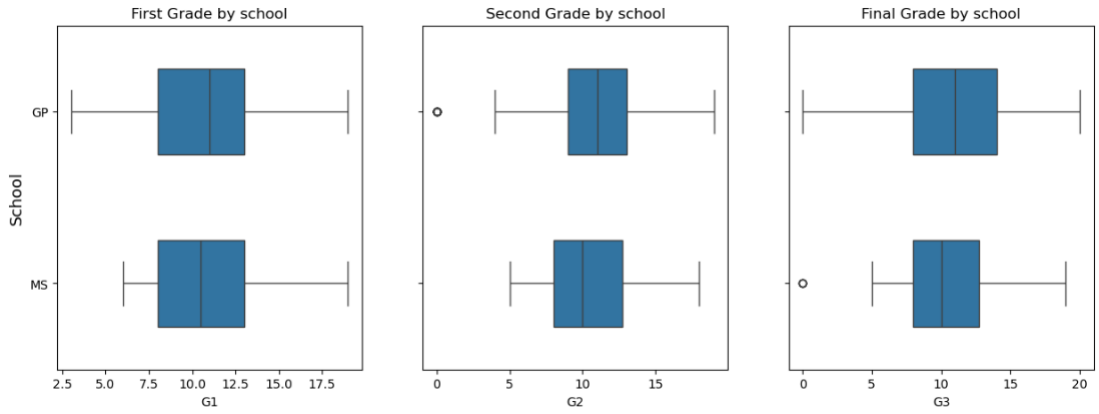

Figure 7: Density plot showing the distribution of each period score

It was then looked at how the amount of time students study per week relates to their final grade. Below is a pie chart showing the proportions of different study times. Figures 9 and 10 then show a density plot and violin

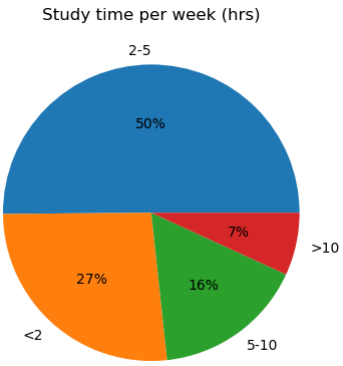

Figure 8: Pie chart showing the proportions of study times

plot, respectively, for each study time. Along with the table of means, there

below 2	2-5	5-10	above 10
Mean 10.05	10.17	11.4	11.26

Table 2: Means of each study time

is a clear trend that the longer the study time, the higher the final grade, although with excessive study time (¿ 10) it doesn't get much higher and may in fact get lower.

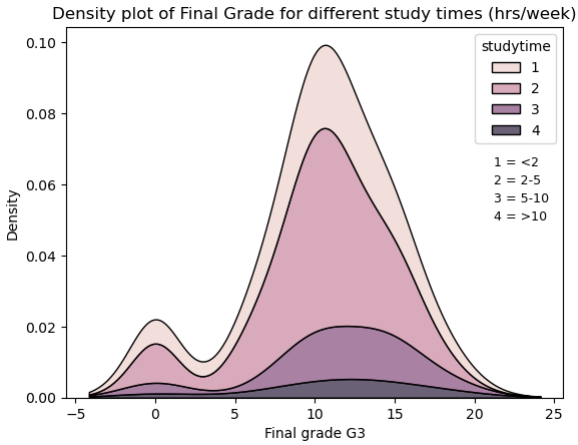

Figure 9: Density plot showing the distribution of final grade for different study times

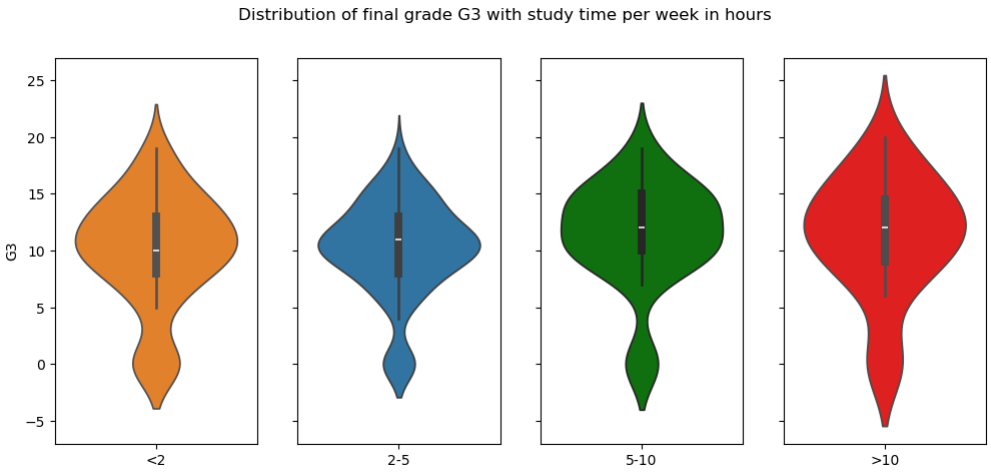

Figure 10: Violin plots of final grade for different study times

Travel time and it's relation to the final grade was also studied. Below are a density and box plots showing a clear trend towards a lower final grade as the travel time to school increases.

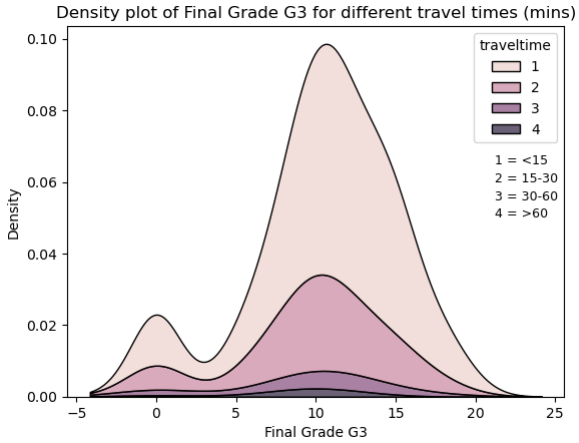

Figure 11: Density plot of final grade for different travel times

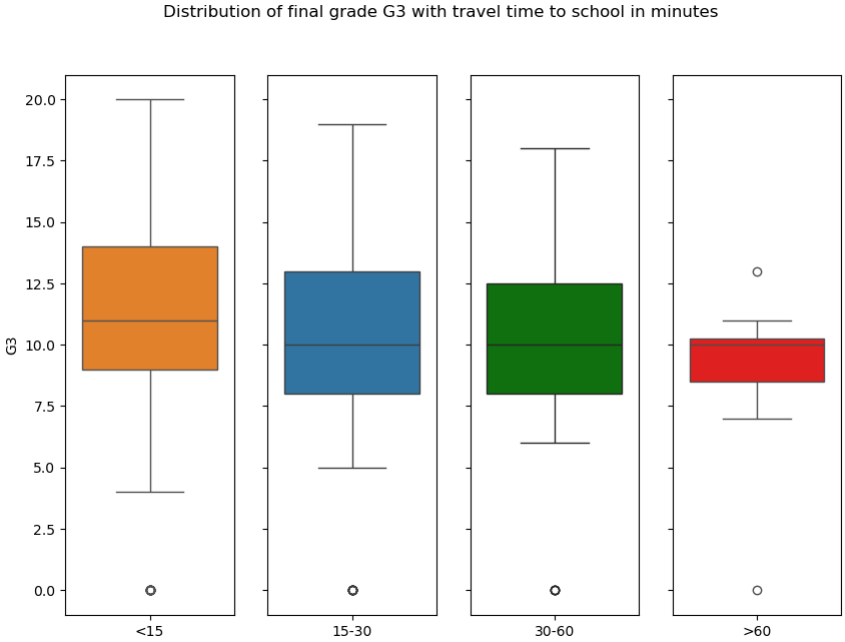

Figure 12: Box plots of final grade for different travel times

For many features the null hypothesis formulated in the methodology was

not rejected but there was a few of note where it was. One was the "internet" feature which produced a p-value of 0.034, thus highlighting the importance of having internet access for a student's final grade. The density plots for each group (being yes or no) are also shown below which appear to show a normal distribution for each, which ensures the validity of the test. The null hypothesis was also rejected for the features "Higher" and "Address" with p-values of 0.031 and 0.013, respectively.

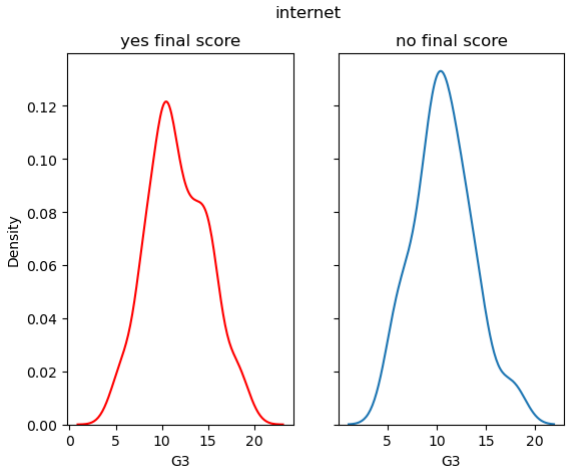

Figure 13: Density plots showing the distribution of final grade when the student either had internet or not

The chi squared contingency also yielded some interesting results, such as there being a strong association between the Mother and Father's education, as shown in the table below.

	0	1	2	3	4
0	0	1	2	0	0
1	1	37	15	5	1
2	0	28	51	17	7
3	0	15	28	38	18
4	1	1	19	40	70

Table 3: Cross-tabulation of Mothers education (top row) and Father's eduction (left column)

3.2 Singular Value Decomposition

While the Principal Components were obtained through use of SVD, the amount of information retained within each component was very little. This

can clearly be seen in Figure 14, where the 1st Principal Component contains below 10 percent of the information. It also takes around 33 components before the standard 95 percent threshold is crossed, which is approximately the same number of original features.

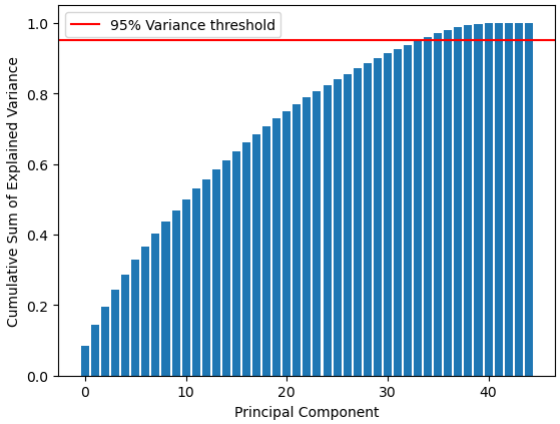

Figure 14: Cumulative sum of the explained variance ratio for each Principal Component

3.3 Gradient Descent

The model implemented did work on the data that was provided to it with a cost function that decreased exponentially as shown in Figure 15. Convergence was achieved at a value of around 154000 for a learning rate of 0.01 and 1000 iterations. The model was then improved with hyperparameter optimisation, and for the hyperparameter matrix chosen, the most optimal combination was a learning rate of 0.0195 and 1250 iterations. This produced a Mean squared error of 4.24.

3.4 Markov Decision Process

Two transition probability matrices were constructed, one for each action that could be taken at each state. This was all that was completed for the

	D	C	B	A
D	0.6	0.3	0.1	0
C	0.2	0.5	0.2	0.1
B	0.1	0.2	0.4	0.3
A	0	0.1	0.2	0.7

Table 4: Transition probabilities from one state to another for A(SM)

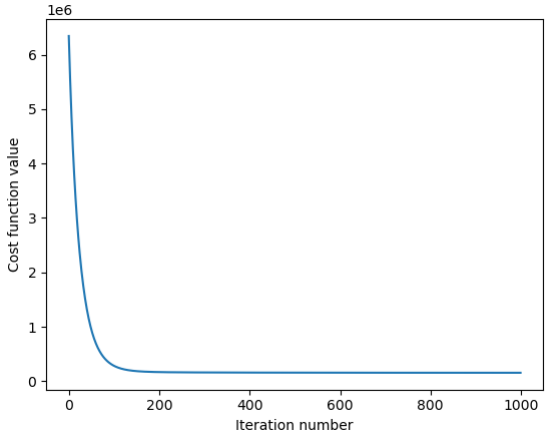

Figure 15: Cost function decreasing over iterations during Gradient Descent

	D	C	B	A
D	0.9	0.1	0	0
C	0.4	0.5	0.1	0
B	0.2	0.4	0.3	0.1
A	0	0.2	0.3	0.5

Table 5: Transition probabilities from one state to another for A(SL)

Markov Decision Process

4 Discussion

4.1 The Project

The EDA revealed some insights into the relationship between some of the feature variables and the target variables, especially travel time, showing that students who live closer to their school would perform better on average. This was not a comprehensive data analysis by any means, as I did not deem it necessary to plot all the feature variables due to a risk of over cluttering the report. But this is something to consider in the future, whereby a program could automatically plot all the features and highlight the ones that are most revealing.

The Hypothesis testing was interesting as it revealed that certain features such as "internet", "higher" and "Address" were significant contributors to the variation in the final grades. This suggests that special attention ought to be given to those students without internet access, who have no desire for higher education or those living in Urban areas, as these may be factors leading to a lower grade. The Hypothesis testing also couldn't be applied

to features that weren't binary, so perhaps methods could be implemented to include those.

It was disappointing that the Principal Components obtained from SVD contained such little information from the database each, but this may have been expected as PCA is often applied to datasets with hundreds of feature variables. The components were not compared to the original variables in this analysis due to them not retaining much of the original information.

Even though the Gradient Descent did produce a converging cost function with optimised weights, I do think a better model can be implemented to lower the Mean Squared Error more as it was quite large relative to the final grade scores. A more advanced model such as a neural network would be used, utilising the concepts of hidden nodes and backpropagation.

4.2 Ethical considerations

There are also ethical implications to be considered in relation to the AI techniques that have been implemented. Since the project was dealing with precious information about school children and their families, a certain level of transparency is necessary such that the children and parents know exactly how the data is being utilised at all times. This ensures fairness, as any ethical imbalance that may occur can be contested by the parents.

If the AI solution were to backfire and result in real-world outcomes that are less than desirable, then it is important to lay out who is to be held accountable for this, whether it be the parents for allowing their data to be collected, or the AI engineer for creating a faulty program.

5 Conclusion

This project was an investigation into a student performance database by performing an EDA as well as implementing some foundational AI techniques including SVD, Gradient Descent and Markov Decision Process. The EDA provided insight into the database, and the hypothesis testing and chi-squared contingency revealed some interesting comparisons between features and groups within the features.

The lack of information in each Principal Component showed that it was not necessary to perform feature reduction on the dataset. The Gradient Descent yielded a Mean Squared Error of approximately 4.24, which could be lower if a neural network was implemented instead.

Word Count: 3146 (according to https://www.montereylanguages.com/pdf-word-count-online-free-tool.html)

6 Appendix

```
import pandas as pd
import numpy as np


#Read in CSV file and seperate into columns
df_math = pd.read_csv('student-mat.csv', sep=';')

#Allows all columns to be displayed
pd.set_option('display.max_columns', None)

#Checks if there are any missing values
print(df_math.isna().any())

#Perform One Hot Encoding on nominal variables
categorical_variables = ['Mjob','Fjob','reason','
    guardian']
df_math_ohe = pd.get_dummies(df_math, columns =
    categorical_variables)

#Encoding the binary columns
binary_cols = ['school','sex','address','famsize','
    Pstatus','schoolsup','famsup','paid','activities','
    nursery','higher','internet','romantic']

df_math_ones = pd.get_dummies(df_math_ohe[binary_cols
    ], drop_first = True)

df_math_pre = df_math_ohe.drop(binary_cols, axis = 1).
    join(df_math_ones)

df_math_pre.head()

df_gp = df_math[df_math['school'] == 'GP']
df_ms = df_math[df_math['school'] == 'MS']

df_math.describe()

import matplotlib.pyplot as plt
import seaborn as sns
```

```
#Density plot of all student grades to observe
    distribution
plt.figure()
sns.kdeplot(df_math, x = 'G1', alpha = 0.7, label = '
    G1')
sns.kdeplot(df_math, x = 'G2', alpha = 0.7, label = '
    G2')
sns.kdeplot(df_math, x = 'G3', label = 'G3')
plt.xlabel("Student Score")
plt.title("Density plot showing distribution of each
    score")
plt.legend()

gp_grade_mean = np.round(df_gp[['G1','G2','G3']].mean
    (), 2)
ms_grade_mean = np.round(df_ms[['G1','G2','G3']].mean
    (),2)

gp_grade_std = df_gp[['G1','G2','G3']].std()
ms_grade_std = df_ms[['G1','G2','G3']].std()

print("Mean of G1 for GP and MS = {} and {}
    respectivily".format(gp_grade_mean.iloc[0],
    ms_grade_mean.iloc[0]))
print("Mean of G2 for GP and MS = {} and {}
    respectivily".format(gp_grade_mean.iloc[1],
    ms_grade_mean.iloc[1]))
print("Mean of G3 for GP and MS = {} and {}
    respectivily".format(gp_grade_mean.iloc[2],
    ms_grade_mean.iloc[2]))
#Creation of 3 subplots running horizontally
fig, ax = plt.subplots(1,3, figsize = (15,5))
#Three density plots comparing student scores between
    schools for each grade
sns.kdeplot(df_gp, ax = ax[0], x = 'G1', label = "GP",
     color = 'r')
sns.kdeplot(df_ms, ax = ax[0], x = 'G1', label = "MS",
     color = 'g')
ax[0].set_xlabel("First Grade G1")
ax[0].legend()

sns.kdeplot(df_gp, ax = ax[1], x = 'G2', label = "GP",
     color = 'r')
```

```
sns.kdeplot(df_ms, ax = ax[1], x = 'G2', label = "MS",
    color = 'g')
ax[1].set_xlabel("Second Grade G2")
ax[1].legend()

sns.kdeplot(df_gp, ax = ax[2], x = 'G3', label = "GP",
    color = 'r')
sns.kdeplot(df_ms, ax = ax[2], x = 'G3', label = "MS",
    color = 'g')
ax[2].set_xlabel("Final Grade G3")
ax[2].legend()

#Create figure with 3 subplots running horizontally
fig, ax = plt.subplots(1,3, figsize = (15,5), sharey =
    True)
#Three boxplots displaying distribution of grades
   grouped by school
sns.boxplot(df_math_ohe, ax = ax[0], x = 'G1', y = '
   school', width = 0.5)
ax[0].set_xlabel("G1")
ax[0].set_ylabel("School", size = 13)
ax[0].set_title("First Grade by school")

sns.boxplot(df_math_ohe, ax = ax[1], x = 'G2', y = '
   school',  width = 0.5)
ax[1].set_xlabel("G2")
ax[1].set_title("Second Grade by school")

sns.boxplot(df_math_ohe, ax = ax[2], x = 'G3', y = '
   school',  width = 0.5)
ax[2].set_xlabel("G3")
ax[2].set_title("Final Grade by school")

study_counts = df_math['studytime'].value_counts()
study_label = ['2-5','<2','5-10','>10']

plt.pie(study_counts, labels = study_label, autopct
   ='%.0f%%')
plt.title("Study time per week (hrs)")

df_study = df_math[['studytime','G3']].sort_values(['
   studytime'], ascending = True)
df_study1 = df_study[df_study['studytime'] == 1]
df_study2 = df_study[df_study['studytime'] == 2]
```

```
df_study3 = df_study[df_study['studytime'] == 3]
df_study4 = df_study[df_study['studytime'] == 4]

#4 violin plots showing the distribution of G3 for
    different study times
fig, ax = plt.subplots(1,4, figsize = (12,5), sharey =
    True)
fig.suptitle("Distribution of final grade G3 with
    study time per week in hours")
sns.violinplot(data = df_study1, ax = ax[0], x = '
    studytime', y = 'G3', color = 'tab:orange')
ax[0].set_xlabel("<2")
ax[0].tick_params(labelbottom = False)
sns.violinplot(data = df_study2, ax = ax[1], x = '
    studytime', y = 'G3')
ax[1].set_xlabel("2-5")
ax[1].tick_params(labelbottom = False)
sns.violinplot(data = df_study3, ax = ax[2], x = '
    studytime', y = 'G3', color = 'g')
ax[2].set_xlabel("5-10")
ax[2].tick_params(labelbottom = False)
sns.violinplot(data = df_study4, ax = ax[3], x = '
    studytime', y = 'G3', color = 'r')
ax[3].set_xlabel(">10")
ax[3].tick_params(labelbottom = False)

sns.kdeplot(df_study, x= "G3", hue = "studytime",
    multiple = "stack", alpha = 0.7)
plt.xlabel("Final grade G3")
plt.title("Density plot of Final Grade for different
    study times (hrs/week)")
plt.text(21.25, 0.065, "1 = <2", fontsize = 9)
plt.text(21.25, 0.06, "2 = 2-5", fontsize = 9)
plt.text(21.25, 0.055, "3 = 5-10", fontsize = 9)
plt.text(21.25, 0.05, "4 = >10", fontsize = 9)

study1_mean = np.round(df_study1["G3"].mean(),2)
study2_mean = np.round(df_study2["G3"].mean(),2)
study3_mean = np.round(df_study3["G3"].mean(),2)
study4_mean = np.round(df_study4["G3"].mean(),2)

print("The mean final score for each study time group
    (<2, 2-5, 5-10, >10) is {}, {}, {}, {},
    respectively".format(study1_mean,study2_mean,
```

```
    study3_mean,study4_mean))

sns.jointplot(df_math, x = "G2", y = "G3")

df_travel = df_math[['traveltime','G3']].sort_values
    (['traveltime'], ascending = True)
df_travel1 = df_travel[df_travel['traveltime'] == 1]
df_travel2 = df_travel[df_travel['traveltime'] == 2]
df_travel3 = df_travel[df_travel['traveltime'] == 3]
df_travel4 = df_travel[df_travel['traveltime'] == 4]

fig, ax = plt.subplots(1,4, figsize = (10,7), sharey =
     True)
fig.suptitle("Distribution of final grade G3 with
    travel time to school in minutes")
sns.boxplot(data = df_travel1, ax = ax[0], x = '
    traveltime', y = 'G3', color = 'tab:orange')
ax[0].set_xlabel("<15")
ax[0].tick_params(labelbottom = False)
sns.boxplot(data = df_travel2, ax = ax[1], x = '
    traveltime', y = 'G3')
ax[1].set_xlabel("15-30")
ax[1].tick_params(labelbottom = False)
sns.boxplot(data = df_travel3, ax = ax[2], x = '
    traveltime', y = 'G3', color = 'g')
ax[2].set_xlabel("30-60")
ax[2].tick_params(labelbottom = False)
sns.boxplot(data = df_travel4, ax = ax[3], x = '
    traveltime', y = 'G3', color = 'r')
ax[3].set_xlabel(">60")
ax[3].tick_params(labelbottom = False)

sns.kdeplot(df_travel, x= "G3", hue = "traveltime",
    multiple = "stack", alpha = 0.7)
plt.xlabel("Final Grade G3")
plt.title("Density plot of Final Grade G3 for
    different travel times (mins)")
plt.text(21.25, 0.065, "1 = <15", fontsize = 9)
plt.text(21.25, 0.06, "2 = 15-30", fontsize = 9)
plt.text(21.25, 0.055, "3 = 30-60", fontsize = 9)
plt.text(21.25, 0.05, "4 = >60", fontsize = 9)

travel1_mean = np.round(df_travel1["G3"].mean(),2)
travel2_mean = np.round(df_travel2["G3"].mean(),2)
```

```
travel3_mean = np.round(df_travel3["G3"].mean(),2)
travel4_mean = np.round(df_travel4["G3"].mean(),2)

print("The mean final score for each study time group
    (<15, 15-30, 30-60, >60) is {}, {}, {}, {},
    respectively".format(travel1_mean,travel2_mean,
    travel3_mean,travel4_mean))

import scipy.stats as st
from statsmodels.stats.weightstats import ztest as
    ztest
from scipy import stats

#Remove outliers (where final grade is zero) to ensure
     normality
df_math_nonzero = df_math.drop(df_math[df_math["G3"]
    == 0].index)
#Specify which feature to be tested and the two groups
     it is divided into
feature_name = "address"
group1_name = "U"
group2_name = "R"

group_1 = df_math_nonzero[df_math_nonzero[feature_name
    ] == group1_name]
group_2 = df_math_nonzero[df_math_nonzero[feature_name
    ] == group2_name]

fig, ax = plt.subplots(1,2, sharey = True)
sns.kdeplot(group_1, ax = ax[0], x = "G3", color = 'r
    ')
sns.kdeplot(group_2, ax = ax[1], x = "G3")
fig.suptitle(feature_name)
ax[0].set_title("{} final score".format(group1_name))
ax[1].set_title("{} final score".format(group2_name))

alpha = 0.05

z = ztest(group_1["G3"], group_2["G3"],value = 0)
#Test whether to reject the null hypothesis or not
    depending on the p-value
if z[1] <= alpha:
     print("There is sufficient evidence to reject the
         null hypothesis that there is no significant
```

```
        difference between the groups in terms of final
         score\np-value = {}".format(z[1]))
else:
    print("There is not sufficient evidence to reject
       the null hypothesis that there is no
       significant difference between the groups in
       terms of the final score\np-value = {}".format(
       z[1]))

from scipy.stats import chi2_contingency
#Define function for performing chi 2 analysis
def chi_2(cat_1, cat_2, df_math):

    #Selected category data taken from dataframe
    df_cat_1 = df_math[cat_1]
    df_cat_2 = df_math[cat_2]
    #Cross tabulation table created from the 2
       categories
    contingency_table = pd.crosstab(df_cat_1, df_cat_2
       )

    #Relevant values retrieved using chi2 contingency
       function
    chi2, p, dof, expected = chi2_contingency(
       contingency_table)

    #Significance level defined
    alpha = 0.05

    # Interpret the results
    if p < alpha:
        result = print("There is a significant
           association between {} and {} preferences.\
           nChi 2 value is {}\np-value is {}\nDegrees
           of freedom {}\nExpected value {}".format(
           cat_1,cat_2, chi2, p, dof, expected))
    else:
        result = print("There is no significant
           association between {} and {} preferences.\
           nChi 2 value is {}\np-value is {}\nDegrees
           of freedom {}\nExpected value {}".format(
           cat_1,cat_2, chi2, p, dof, expected))

    return result, contingency_table
```

```
cat_1 = str(input("Please choose the first category:")
    )
cat_2 = str(input("Please choose the second category
    :"))

chi_2_result, con_table = chi_2(cat_1, cat_2, df_math)
print(con_table)

from sklearn.preprocessing import StandardScaler

def singular_decomp(X, n_components):
    scaler = StandardScaler()
    #X data standardised
    X_stan = scaler.fit_transform(X)
    #SVD performed on x data
    U, S, Vt = np.linalg.svd(X_stan)

    X_reduced = np.dot(X_stan, Vt.T[:,:n_components])

    explained_variance_ratio = (S**2)/(np.sum(S**2))

    return X_reduced, Vt, explained_variance_ratio

#Drop the target variable 'G3' from the dataframe and
    assign the rest to the variable X
X = df_math_pre.drop(['G3'], axis = 1).values
#Assign column G3 to variable Y
Y = df_math_pre['G3'].values

#The number of Principal Components wanted
num_components = 10
X_reduced, Vt, explained_variance_ratio=
    singular_decomp(X, num_components)

#Plot of the explained variance ratio for each
    Principal Component
plt.bar(range(len(explained_variance_ratio)),
    explained_variance_ratio)

#Cumulative sum of the explained variance ratios
cum_evr = np.cumsum(explained_variance_ratio)
```

```
#Plot of the cumulative sum for each principal
    component
plt.bar(range(len(explained_varience_ratio)), cum_evr)
plt.xlabel("Principal Component")
plt.ylabel("Cumulative Sum of Explained Variance")
plt.axhline(y = 0.95, color = 'r', label = "95%
    Variance threshold")
plt.legend()

#Imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Standardize the feature data
scaler = StandardScaler()
X_stan = scaler.fit_transform(X)

#Add a column of ones to the start X to account for
    the intercept term
m = len(Y)
X_final = np.hstack((np.ones((m,1)), X_stan))

#Split both feature and target variables into training
     and test data
x_train, x_test, y_train, y_test = train_test_split(
    X_final, Y, test_size = 0.2)

#Define gradient descent function
def gradient_descent(X, Y, weight, iterations,
    learning_rate):

    m = len(Y)
    #Array to record change of the cost function with
        each iteration
    cost_history = []

    for i in range(iterations):

        #Gradient of cost function calculated from
            predicted value of y and difference between
             this and actual value
        prediction = X.dot(weight)
        error = prediction - Y
        gradient = (2/m) * (X.T.dot(error))
```

```
        #The weight values are modified
        weight -= learning_rate*gradient
        cost = (1/2*m) * np.sum(error**2)
        cost_history.append(cost)

        #Every 100th iteration, the ith value and cost
            are printed. (Been commented out to use
           hyperparameter optimisation model)
        #if i%100 == 0:
        #    print("Cost at {}th iteration is {}".
           format(i, np.round(cost, 2)))

    return weight, cost_history

#Define weight term, learning rate and number of
   iterations
theta = np.zeros(x_train.shape[1])
theta
learning_rate = 0.01
iterations = 1000


theta, cost_history = gradient_descent(x_train,
   y_train, theta, iterations, learning_rate)

plt.plot(range(len(cost_history)), cost_history)
plt.xlabel("Iteration number")
plt.ylabel("Cost function value")

#Define a function used for optimising the learning
   rate and iteration number
def gradient_descent_op(x_train, y_train, y_test,
   learning_rate_arr, iterations_arr):

    theta = np.zeros(x_train.shape[1])
    #Define array to store the mean square error
       values for different hyperparameter
       combinations
    MSE_arr = np.zeros(shape = (len(learning_rate_arr)
       , len(iterations_arr)))

    #Iterate through each hyperparameter combination
    for i in range(len(learning_rate_arr)):
        for j in range(len(iterations_arr)):
```

```
            #Hyperparameter values assigned to
                variables
            learn_rate = learning_rate_arr[i]
            iteration = iterations_arr[j]

            #Gradient descent performed using feature
                and target training data
            theta_r, cost_history_r = gradient_descent
                (x_train, y_train, theta, iteration,
                learn_rate)

            #Calculation of mean squared error
            predict_y = x_test.dot(theta_r)
            MSE = np.mean((predict_y - y_test)**2)

            #Mean squared error value stored in the
                array
            MSE_arr[i,j] = MSE

    #Index values calculated for where MSE is at a
        minimum
    op_index = np.unravel_index(MSE_arr.argmin(),
        MSE_arr.shape)

    #Index values used to find optimal hyperparameter
        values
    learning_rate_op = learning_rate_arr[op_index[0]]
    iterations_op = iterations_arr[op_index[1]]

    return learning_rate_op, iterations_op, np.min(
        MSE_arr)

#Define minimum and maximum learning rate for
    optimisation and create array of values
min_learning_rate = 0.001
max_learning_rate = 0.02
learning_step = 0.0005
learning_rate_arr = np.arange(min_learning_rate,
    max_learning_rate+learning_step, learning_step)

#Define minimum and maximum iteration number for
    optimisation and create array of values
min_it = 1200
```

```
max_it = 1400
it_step = 50
iterations_arr = np.arange(min_it, max_it + it_step,
    it_step)
#Find the optimum learning rate and number of
    iterations
learning_op, iterations_op, min_MSE =
    gradient_descent_op(x_train, y_train, y_test,
    learning_rate_arr, iterations_arr)

print("The optimal learning rate and number of
    iterations are {} and {}, respectively, with a mean
     squared error of {}".format(learning_op,
    iterations_op, min_MSE))
```