ECS8050 Final Assignment

Hugo Collins
Student Number: 40446925
Email: hcollins05@qb.ac.uk
MSc in Artificial Intelligence 2024

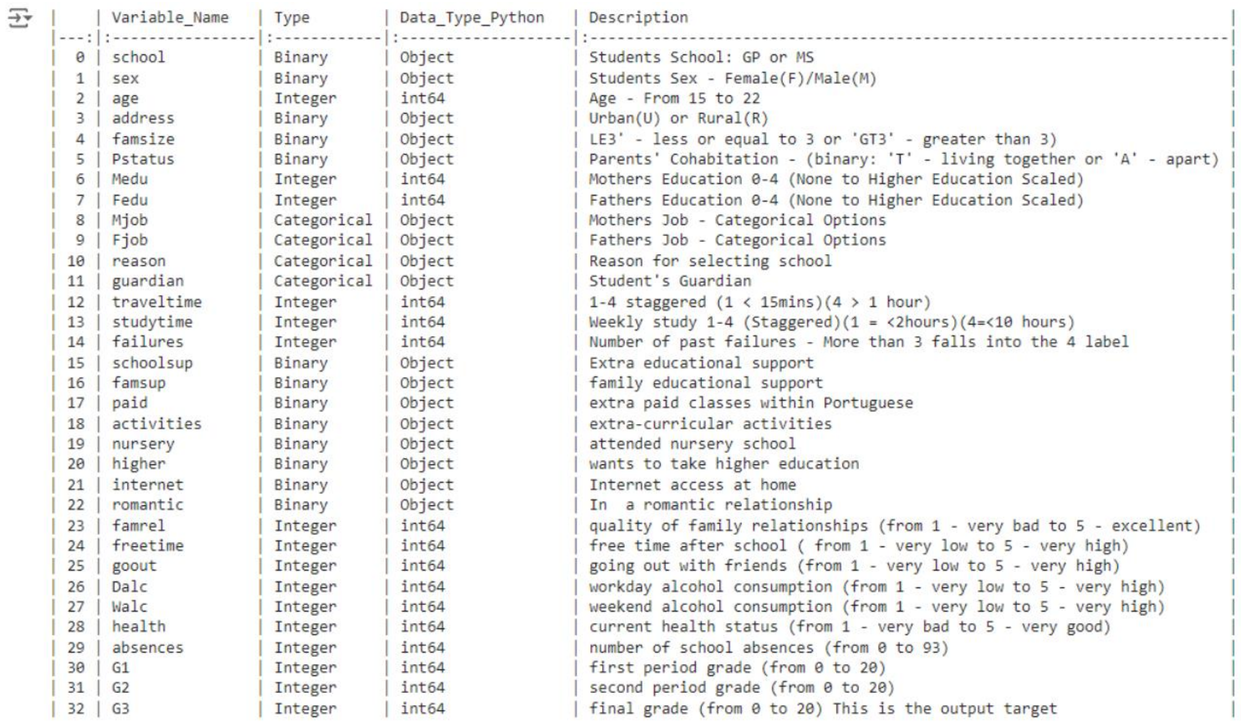
All code can be found below the report

Abstract

This report focuses on developing a regression model to predict new student performance in Mathematics using the Student Performance dataset from the UCI Repository. The model will apply the mathematical concepts I have learned over the past four weeks to the areas of optimisation, dimensionality reduction, and exploratory data analysis. To deepen my understanding and display my learnings, manual calculations will be used wherever possible instead of relying solely on built-in Python functions. During the report I built a linear regression model and a neural network from scratch and performed several tests on both. The final model is a collection of all my findings into one single regression model.

1. 1 Data

The dataset supplied contained information regarding the performance of students in the subject Portuguese. The data was provided by the UCI Repository. The data was collected from two Portuguese schools. The dataset contained 33 columns – 32 of which contained features for prediction and G3 (the target variable). The dataset contained three separate grades score; G1 (first period grade), G2 (second period grade) and G3 (the final year grade). Due to the nature of these results, there was a strong correlation between G3 and features G1 and G2, as both these features directly contribute to the final G3 mark. This is something I must consider. In matrix form the original dataset has a shape of [649,33] representing the 649 rows and 33 columns. One clear challenge from the data is that there are different data types within the features. There are many categorical entries which must be handled differently to those of the integer or numeric kind. There are also binary and ordinal columns which need to be properly dealt with too during preprocessing. The columns and what they represent can be seen in the table below which I created using pandas. [2] The overall aim of my report will be to apply the mathematical techniques that I have learned throughout this module and create a regression model which predicts the G3 score of new students sitting Portuguese.



```
|    | Variable_Name   | Type        | Data_Type_Python   | Description                                                              |
|---:|:----------------|:------------|:-------------------|:-------------------------------------------------------------------------|
|  0 | school          | Binary      | Object             | Students School: GP or MS                                                |
|  1 | sex             | Binary      | Object             | Students Sex - Female(F)/Male(M)                                         |
|  2 | age             | Integer     | int64              | Age - From 15 to 22                                                      |
|  3 | address         | Binary      | Object             | Urban(U) or Rural(R)                                                     |
|  4 | famsize         | Binary      | Object             | LE3' - less or equal to 3 or 'GT3' - greater than 3)                     |
|  5 | Pstatus         | Binary      | Object             | Parents' Cohabitation - (binary: 'T' - living together or 'A' - apart) |
|  6 | Medu            | Integer     | int64              | Mothers Education 0-4 (None to Higher Education Scaled)                  |
|  7 | Fedu            | Integer     | int64              | Fathers Education 0-4 (None to Higher Education Scaled)                  |
|  8 | Mjob            | Categorical | Object             | Mothers Job - Categorical Options                                        |
|  9 | Fjob            | Categorical | Object             | Fathers Job - Categorical Options                                        |
| 10 | reason          | Categorical | Object             | Reason for selecting school                                              |
| 11 | guardian        | Categorical | Object             | Student's Guardian                                                       |
| 12 | traveltime      | Integer     | int64              | 1-4 staggered (1 < 15mins)(4 > 1 hour)                                   |
| 13 | studytime       | Integer     | int64              | Weekly study 1-4 (Staggered)(1 = <2hours)(4=<10 hours)                   |
| 14 | failures        | Integer     | int64              | Number of past failures - More than 3 falls into the 4 label             |
| 15 | schoolsup       | Binary      | Object             | Extra educational support                                                |
| 16 | famsup          | Binary      | Object             | family educational support                                               |
| 17 | paid            | Binary      | Object             | extra paid classes within Portuguese                                     |
| 18 | activities      | Binary      | Object             | extra-curricular activities                                              |
| 19 | nursery         | Binary      | Object             | attended nursery school                                                  |
| 20 | higher          | Binary      | Object             | wants to take higher education                                           |
| 21 | internet        | Binary      | Object             | Internet access at home                                                  |
| 22 | romantic        | Binary      | Object             | In  a romantic relationship                                              |
| 23 | famrel          | Integer     | int64              | quality of family relationships (from 1 - very bad to 5 - excellent)    |
| 24 | freetime        | Integer     | int64              | free time after school ( from 1 - very low to 5 - very high)            |
| 25 | goout           | Integer     | int64              | going out with friends (from 1 - very low to 5 - very high)             |
| 26 | Dalc            | Integer     | int64              | workday alcohol consumption (from 1 - very low to 5 - very high)        |
| 27 | Walc            | Integer     | int64              | weekend alcohol consumption (from 1 - very low to 5 - very high)        |
| 28 | health          | Integer     | int64              | current health status (from 1 - very bad to 5 - very good)              |
| 29 | absences        | Integer     | int64              | number of school absences (from 0 to 93)                                 |
| 30 | G1              | Integer     | int64              | first period grade (from 0 to 20)                                        |
| 31 | G2              | Integer     | int64              | second period grade (from 0 to 20)                                       |
| 32 | G3              | Integer     | int64              | final grade (from 0 to 20) This is the output target                     |
```

1.2 Preprocessing:

The first step in preprocessing was to check for null values using *dfp.isnull().sum()* . Each column returned a count of zero, so I was satisfied that there were no null values. I then examined the central tendency and variability of the numerical columns with dfp.describe(). Most features had a standard deviation near 1, indicating clustering around the mean, while G3 had the highest standard deviation at 3.23, suggesting a wider spread. Notably, I found 15 rows with G3 = 0; these students had no absences but at least one G1 or G2 score, which needed more examination in EDA.

Next, I prepared the data which contained a mix of categorical (some ordinal) and numerical data. While the ordinal values were numeric, they were used as categorising labels.[8] The variable values varied significantly, for instance, "absences" had a maximum of 93, while "studytime" ranged from 1 to 4. This showed that feature scaling would be necessary before feeding this data into my model. Without scaling, features with larger ranges could dominate the distance calculations and lead to biased results [3] . Normalising the data would solve this. I created a function manual_standard_scaler() which manually standardised the data,using the formula below, rather than using the prebuild sklearn StandardScaler().

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

Figure 1: [3]

This formula was not applicable to the categorical data, which needed to be handled separately. One option I explored was using OneHotEncoding. This would create a new column for each entry but would significantly increase the dimensionality of my feature matrix resulting in lots of sparse vectors, which would increase complexity and impact my model's ability to generalise. As a result, I decided to use a combination of binary mapping for the binary columns and only used One-Hot Encoding for the remaining categorical variables. This approach allowed me to reduce the number of features from 44 (if only One-Hot Encoding was used) to 30. While this was still a lot of features, it was the best approach I could find to encoding the categorical data.

2. EDA

The first step of my EDA was to examine the distribution of G3 and analyse the features G1 and G2 and check for correlations. My first plot examined their distributions.

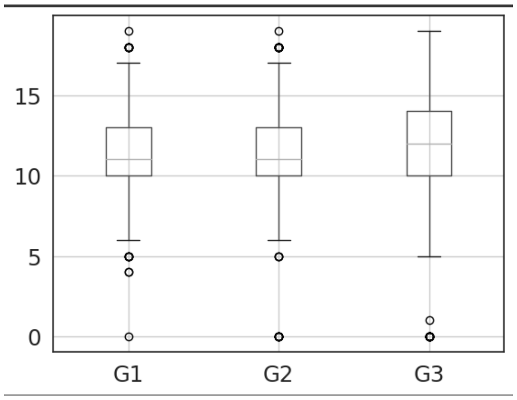


Fig 1.1 Boxplot of G1, G2 and G3

G1 and G2 followed a similar distribution whereas G3 had a wider box indicating greater variability around the mean. The wider whiskers suggested greater variability in the tails, especially towards the maximum, signifying left skew in the data. A cluster of outliers around zero lacked explanation as these students had G1 and G2 scores and zero absences. I considered replacing the 0 with the mean score but felt this would lead to bias in my data so I simply removed these 15 entries.

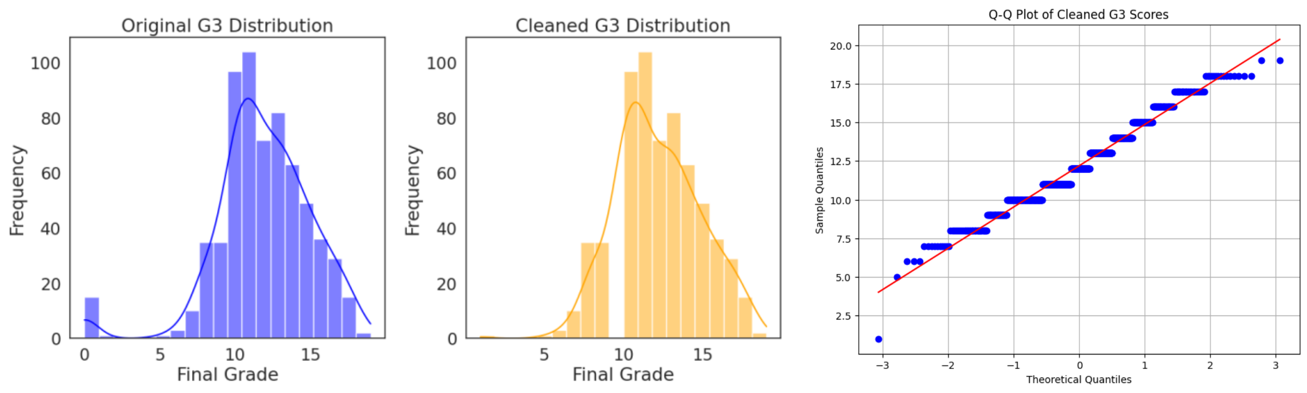
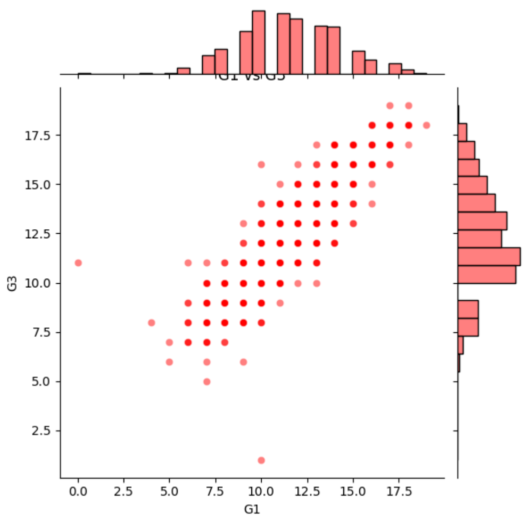
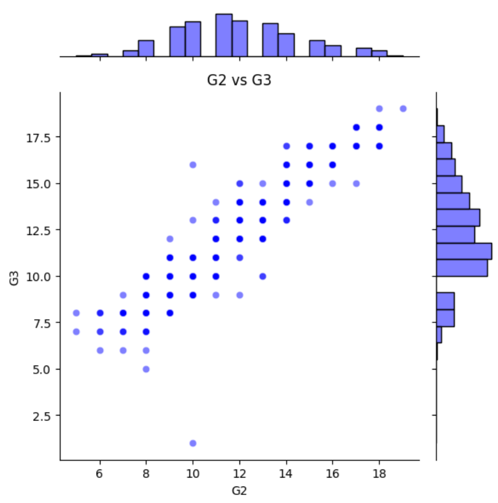


Fig 1.2 Distributions of G3 before and after outliers removed and qq-plot if cleaned G3

After removing the zero values, the data approximated a normal distribution with a slight left skew, as shown in the QQ-plot.

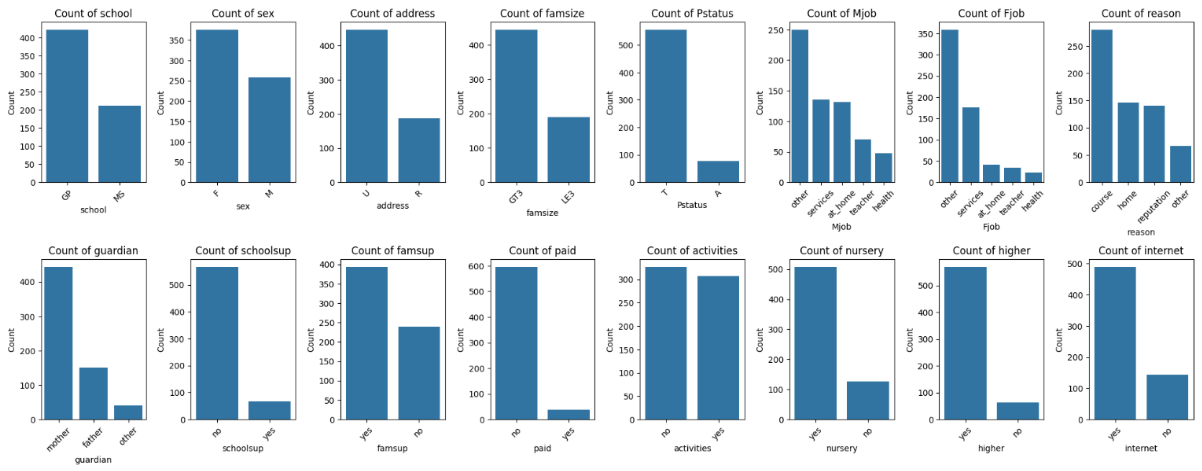
I plotted a jointplot using the seaborn library to see the correlation between G1 and G3 and G2 and G3. The results were as follows.

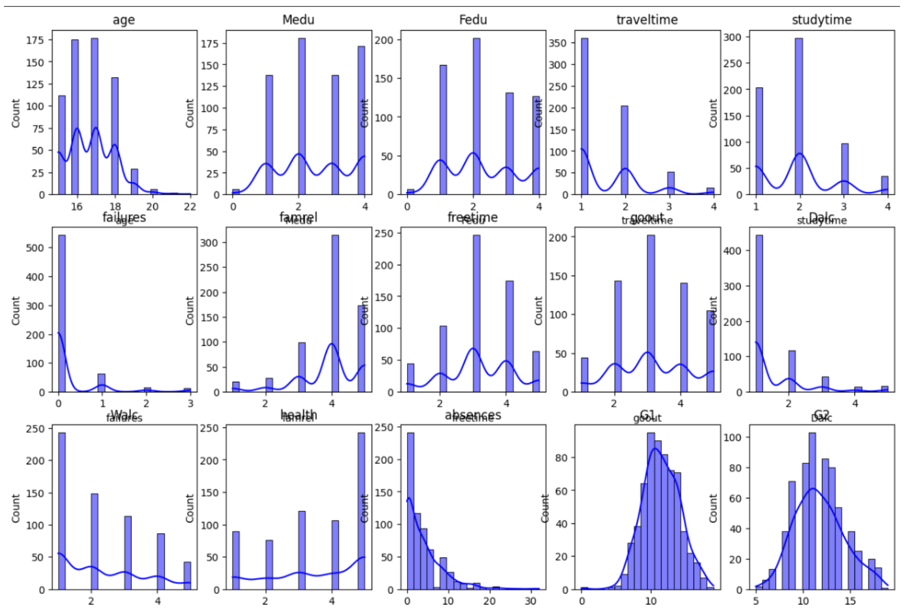




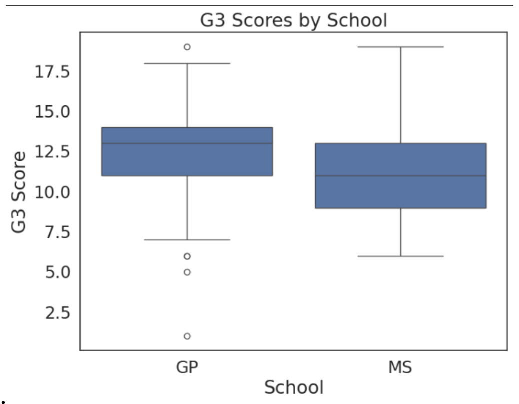
There was a very strong positive correlation between the two features and the target variable. This high correlation posed issues such as model stability and multicollinearity where the model would be overdependent on these variables and disregard the other features. Since I am trying to predict the G3 score for new students who wouldn't have these G1 and G2 scores I dropped G1 and G2 from my feature vector.

I then performed some univariate analysis on both the categorical and numerical columns to quickly examine their individual distributions.

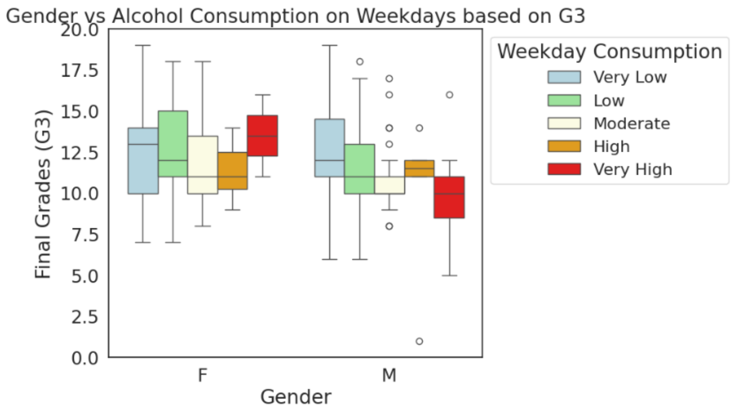




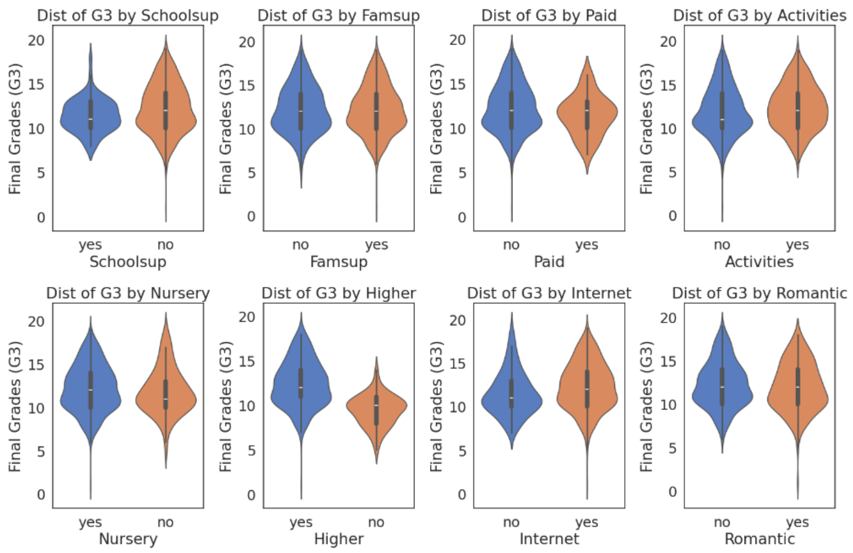
I then began my bivariate and multivariate analysis. In this, I hoped to examine the relationship between my feature variables and the target variable G3. I began with analysis of the categorical features.



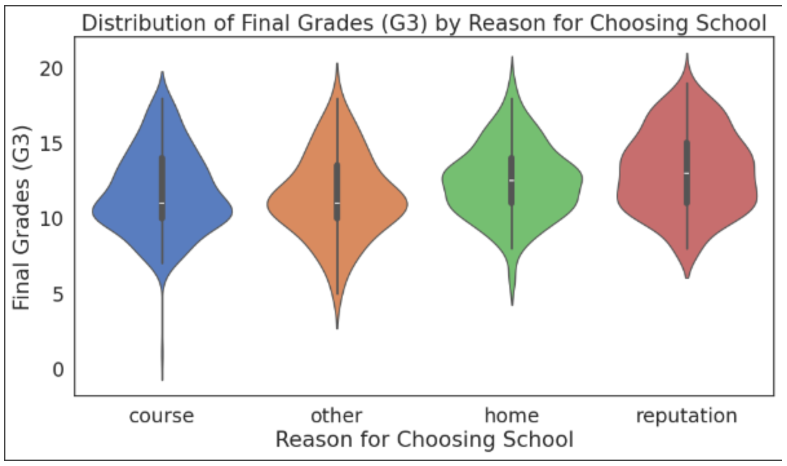
Here I plotted the G3 scores by school. GP students had a higher median and smaller interquartile range than MS students who had more variability. GP contains some outliers on both ends, however.



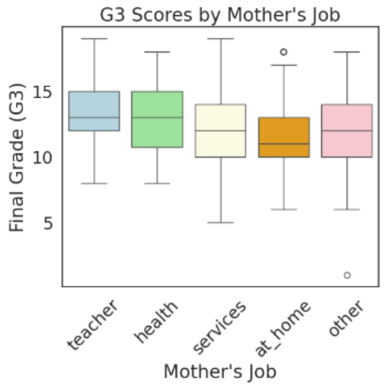
With regards to the females, there does not seem to be much of a correlation, with the median grades and spread roughly similar as alcohol consumption increases. The males however, as alcohol consumption increases, their grades are more negatively affected.

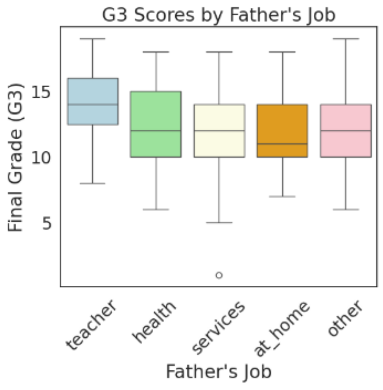


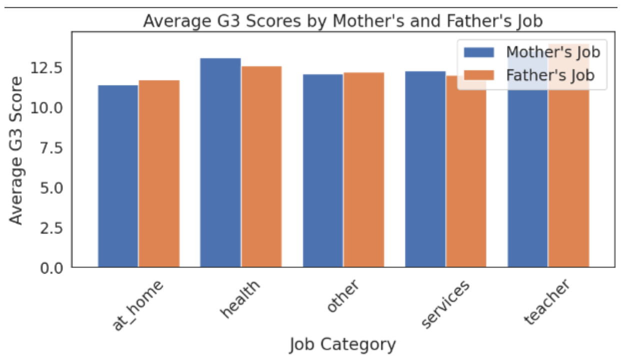
I then plotted violin plots of all the binary “yes”, “no” columns. The standout here was the “higher” column which clearly had a higher median and density at a higher G3 grade than the “no” column. Famsup appeared to have almost identical distributions and therefore may not provide much insight. Schoolsup was clustered around the median for the yes column whereas there is more spread in the no column. Internet, Romantic and Paid all had similar medians.



As the categorical features need one hot encoding, including them was computationally expensive as they added dimensions. I decided to analyse these columns in greater detail. Reputation had a higher density and probability of higher marks than the other reasons with course having the lowest median and highest probability of a lower mark. There does seem to be a vary in grades based on the reason.

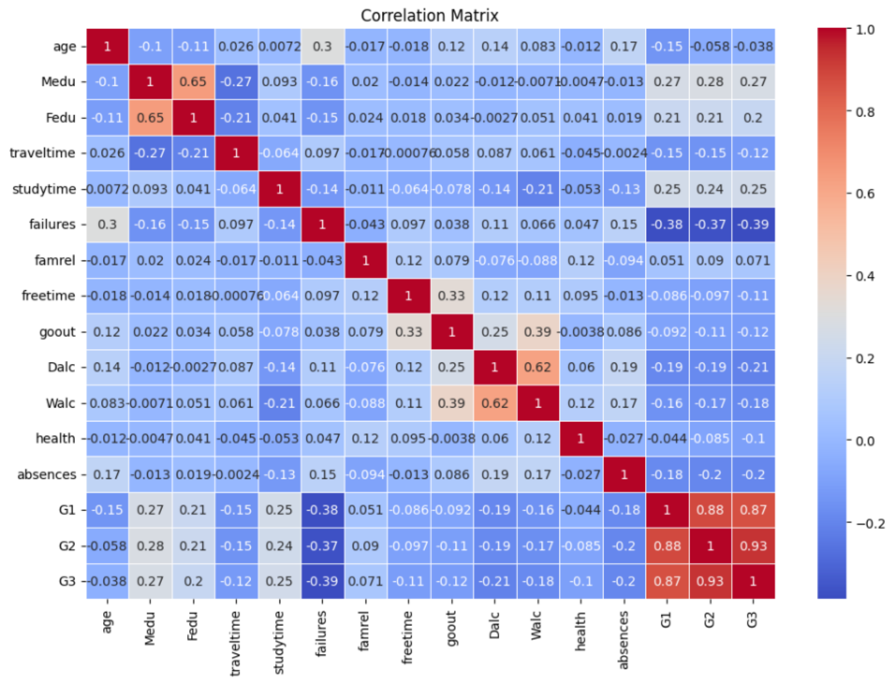






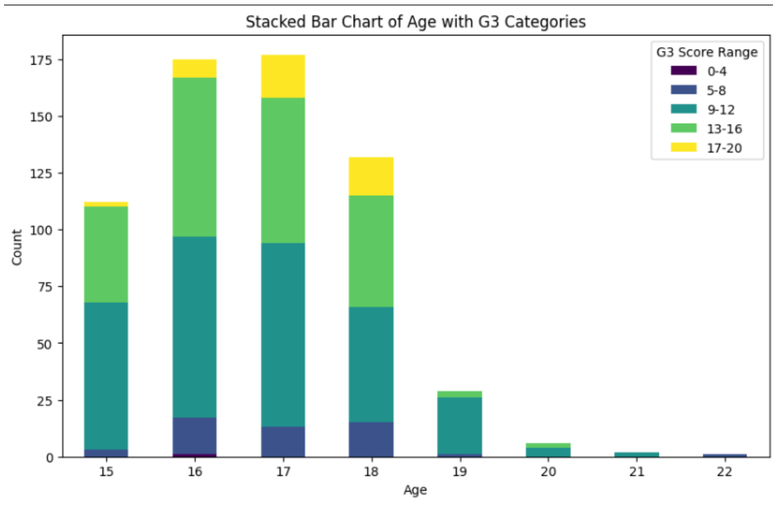
Here I wanted to analyse if there was a similarity between Mjob and Fjob and I'd be able to remove a column if there was. There was a clear similarity in the distributions of each job based on parents. I also analysed the mean scores, and these provided similar results.

Numerical Analysis – Correlation

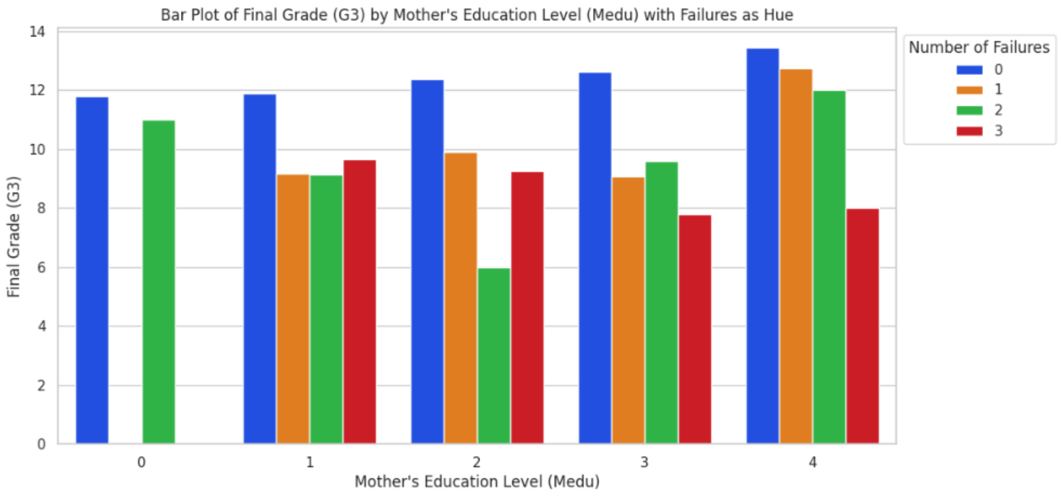

Correlation Matrix

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
age	1	-0.1	-0.11	0.026	0.0072	0.3	-0.017	-0.018	0.12	0.14	0.083	-0.012	0.17	-0.15	-0.058	-0.038
Medu	-0.1	1	0.65	-0.27	0.093	-0.16	0.02	-0.014	0.022	-0.012	-0.0071	0.0047	-0.013	0.27	0.28	0.27
Fedu	-0.11	0.65	1	-0.21	0.041	-0.15	0.024	0.018	0.034	-0.0027	0.051	0.041	0.019	0.21	0.21	0.2
traveltime	0.026	-0.27	-0.21	1	-0.064	0.097	-0.017	0.00076	0.058	0.087	0.061	-0.045	-0.0024	-0.15	-0.15	-0.12
studytime	0.0072	0.093	0.041	-0.064	1	-0.14	-0.011	-0.064	-0.078	-0.14	-0.21	-0.053	-0.13	0.25	0.24	0.25
failures	0.3	-0.16	-0.15	0.097	-0.14	1	-0.043	0.097	0.038	0.11	0.066	0.047	0.15	-0.38	-0.37	-0.39
famrel	-0.017	0.02	0.024	-0.017	-0.011	-0.043	1	0.12	0.079	-0.076	-0.088	0.12	-0.094	0.051	0.09	0.071
freetime	-0.018	-0.014	0.018	0.00076	-0.064	0.097	0.12	1	0.33	0.12	0.11	0.095	-0.013	-0.086	-0.097	-0.11
goout	0.12	0.022	0.034	0.058	-0.078	0.038	0.079	0.33	1	0.25	0.39	-0.0038	0.086	-0.092	-0.11	-0.12
Dalc	0.14	-0.012	-0.0027	0.087	-0.14	0.11	-0.076	0.12	0.25	1	0.62	0.06	0.19	-0.19	-0.19	-0.21
Walc	0.083	-0.0071	0.051	0.061	-0.21	0.066	-0.088	0.11	0.39	0.62	1	0.12	0.17	-0.16	-0.17	-0.18
health	-0.012	-0.0047	0.041	-0.045	-0.053	0.047	0.12	0.095	-0.0038	0.06	0.12	1	-0.027	-0.044	-0.085	-0.1
absences	0.17	-0.013	0.019	-0.0024	-0.13	0.15	-0.094	-0.013	0.086	0.19	0.17	-0.027	1	-0.18	-0.2	-0.2
G1	-0.15	0.27	0.21	-0.15	0.25	-0.38	0.051	-0.086	-0.092	-0.19	-0.16	-0.044	-0.18	1	0.88	0.87
G2	-0.058	0.28	0.21	-0.15	0.24	-0.37	0.09	-0.097	-0.11	-0.19	-0.17	-0.085	-0.2	0.88	1	0.93
G3	-0.038	0.27	0.2	-0.12	0.25	-0.39	0.071	-0.11	-0.12	-0.21	-0.18	-0.1	-0.2	0.87	0.93	1

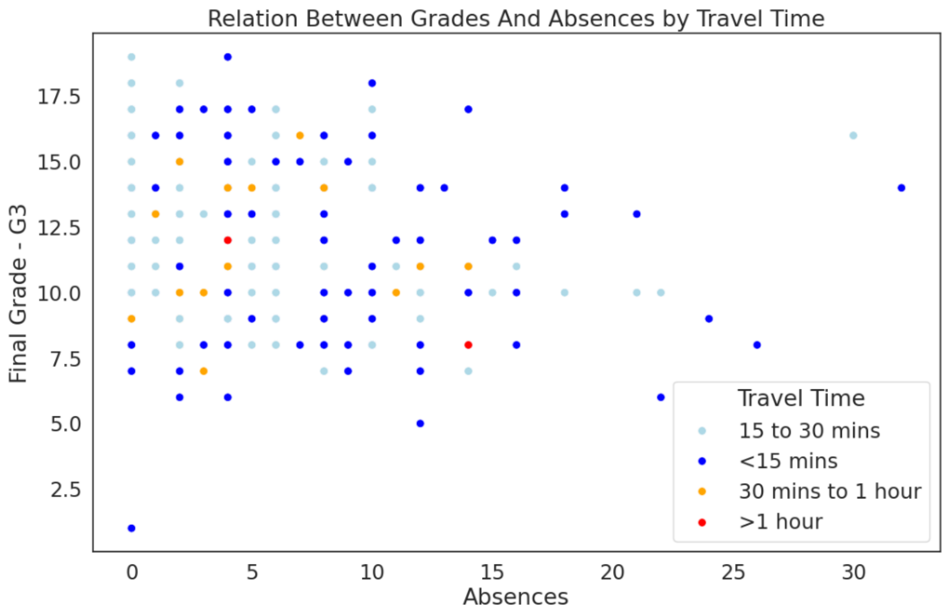
This correlation matrix revealed a lot of information. However, I had to remember that correlation does not imply correlation meaning that while correlation can reveal a lot about the way variables move together, it does not **always** imply one causes the other to occur. As expected, there was a high correlation between the G scores. There is also a high correlation between Dalc and Walc. Fedu and Medu also share a high correlation which is a cause for concern for multicollinearity. Overall, failures shared the highest correlation with G3 out of the numerical features but there was a pretty even spread otherwise, which could lead to an interesting model. Age and famrel appeared to have little correlation to G3 along with health and traveltime. I tested some of these findings in further detail.



This stacked bar chart supported the claim that age has little effect on the G3 score.



This bar chart indicated the relationship between failures and G3 was a strong predictor. There is a clear pattern that as failures increased, G3 decreased.



This scatter plot, surprisingly, showed that there is almost no correlation between G3 and absences. Most students' absences are between 0-20 but there was no clear pattern about how they performed. Also, this scatterplot showed that a longer distance to travel didn't necessarily mean more absences or a worse G3 score.

2.1 Hypothesis Testing

For my tests, since the sample size exceeded 30, I could apply the Central Limit Theorem and assume that the distribution of the means was approximately normal. Before performing a t-test, I checked for equal variances using F-tests. In cases where the variances were unequal, I used Welch's t-test by setting equal_varinaces = False [5].

Test Type	H0	H1	Statistical Value	P-Value at alpha = 0.05	Testing Outcome	Findings
Two Sample T-test	There is no significant difference in the mean G3 score based on School	There is a significant difference in the mean G3 score based on school.	5.662922	0.000000023	Reject H0	As there was a significant difference between school based on G3 I could use it as a feature
Two Sample T-test	There is no significant difference in G3 scores based on studytime	There is a significant difference in G3 scores based on studytime	-1.97037	0.04923	Reject H0 (Barely)	Same as above
Welchs Two Sample T-test	True difference in means is 0 (Dalc and Absences)	True difference in means is not 0 (Dalc and Absences)	-2.689063	0.01130	Reject H0	Due to each sample having uneven variances I had to use Welchs T-test. The findings showed a difference in means signifying an association between the two variables. This will be an important association to capture in my model.
T-test	There is no significant difference in mean scores based on age	There is a significant difference in mean based on age	0.662767	0.50772	Fail to Rejct H0	The t-test confirmed my EDA analysis and showed that there was no significant difference in mean G3 score based on age.
Chi Square	There is no significant association	There is a significant	135.1103073	0.000000	Reject H0	There is a significant association between Mjob and Fjob. This would indicate that if I needed to

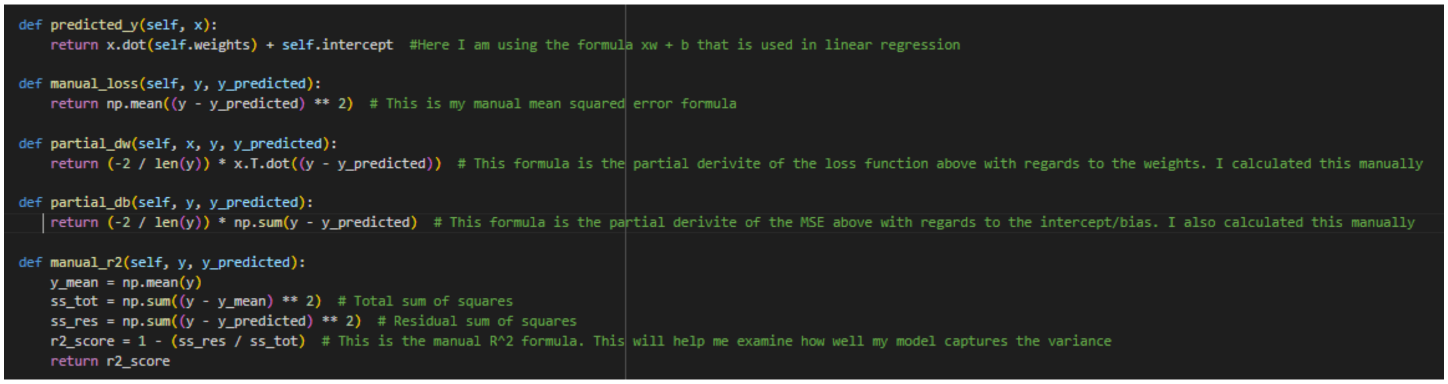
	between Mjob and Fjob	association between Mjob and Fjob				reduce the dimensionality of my feature vector, I could exclude one of them.
ANOVA	The mean G3 scores are equal across all Mjob categories.	The mean G3 scores are not equal across all Mjob categories.	9.0850533	0.0000004	Reject H0	Means were not equal, which indicated that this column contained interesting and informative information for my model.

3. Method

For my models I created two separate classes. One was a manual multivariate linear regression model, and the other was a neural network. I built both from scratch using the techniques learned in the module. I wanted to examine how well a linear regression tool could fit to the data and then compare this to the NN that takes a nonlinear approach using an activation function. I then tested SVD on both models, along with optimising the hyperparameters of both. For this testing phase, I included all the encoded features in the feature matrix. I examined the results of all these models, along with my EDA, and combined my findings into a final model which applied all my mathematical learnings.

Model 1: Multivariate Linear Regression

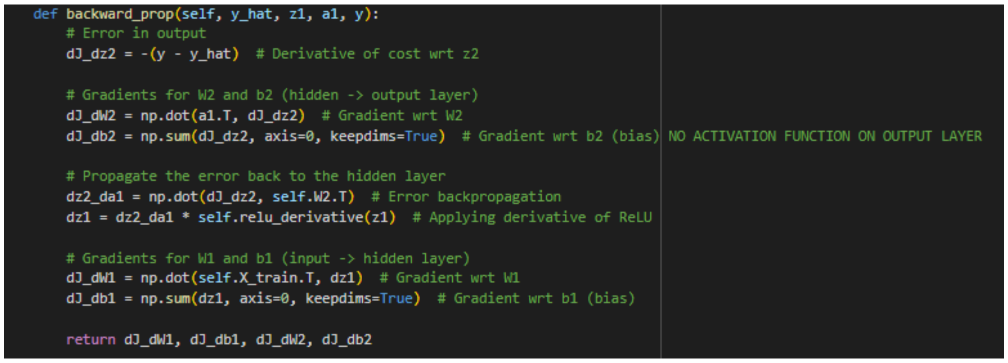
I started by defining mathematical calculations as functions. These functions and their explanations can be seen below:



```python
def predicted_y(self, x):
    return x.dot(self.weights) + self.intercept  #Here I am using the formula xw + b that is used in linear regression

def manual_loss(self, y, y_predicted):
    return np.mean((y - y_predicted) ** 2)  # This is my manual mean squared error formula

def partial_dw(self, x, y, y_predicted):
    return (-2 / len(y)) * x.T.dot((y - y_predicted))  # This formula is the partial derivite of the loss function above with regards to the weights. I calculated this manually

def partial_db(self, y, y_predicted):
    return (-2 / len(y)) * np.sum(y - y_predicted)  # This formula is the partial derivite of the MSE above with regards to the intercept/bias. I also calculated this manually

def manual_r2(self, y, y_predicted):
    y_mean = np.mean(y)
    ss_tot = np.sum((y - y_mean) ** 2)  # Total sum of squares
    ss_res = np.sum((y - y_predicted) ** 2)  # Residual sum of squares
    r2_score = 1 - (ss_res / ss_tot)  # This is the manual R^2 formula. This will help me examine how well my model captures the variance
    return r2_score
```

Model 2: Manual Neural Network

This NN followed the classic learning process of using a forward phase, MSE as the cost function and the backpropagation using gradient descent to update the weights and bias. Originally, I had used sigmoid for my activation function but had an error due to saturation and an error with exploding gradients, so I used RelU instead as it was less computationally expensive [1]. As this was a linear NN for regression there was no activation function on the output. Backward propagation code can be seen below;

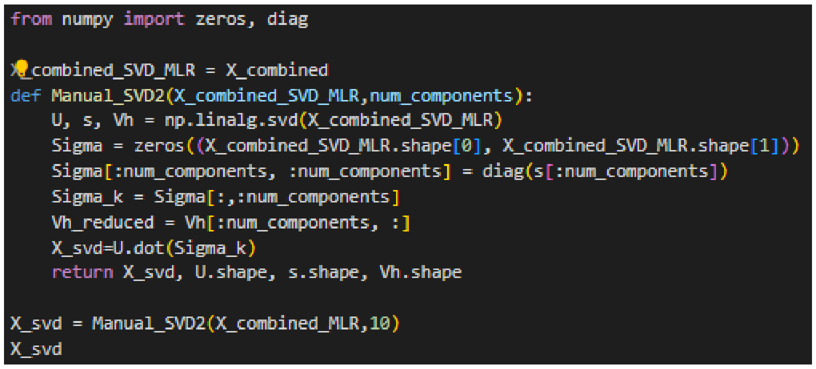


```python
def backward_prop(self, y_hat, z1, a1, y):
    # Error in output
    dJ_dz2 = -(y - y_hat)  # Derivative of cost wrt z2

    # Gradients for W2 and b2 (hidden -> output layer)
    dJ_dW2 = np.dot(a1.T, dJ_dz2)  # Gradient wrt W2
    dJ_db2 = np.sum(dJ_dz2, axis=0, keepdims=True)  # Gradient wrt b2 (bias) NO ACTIVATION FUNCTION ON OUTPUT LAYER

    # Propagate the error back to the hidden layer
    dz2_da1 = np.dot(dJ_dz2, self.W2.T)  # Error backpropagation
    dz1 = dz2_da1 * self.relu_derivative(z1)  # Applying derivative of ReLU

    # Gradients for W1 and b1 (input -> hidden layer)
    dJ_dW1 = np.dot(self.X_train.T, dz1)  # Gradient wrt W1
    dJ_db1 = np.sum(dz1, axis=0, keepdims=True)  # Gradient wrt b1 (bias)

    return dJ_dW1, dJ_db1, dJ_dW2, dJ_db2
```

3.1 Singular Value Decomposition:

I decided to use SVD as my feature reduction algorithm. Due to the number of columns in my data and having to one hot encode the categorical entries, my model was at risk of the curse of dimensionality. As my data was an M x N shaped feature matrix, I found

SVD to be an excellent choice for dimension reduction as it can be applied to a matrix of any shape unlike eigen decomposition. My hope was that the SVD would compress the features without losing any key information. As we learned in this module, SVD breaks the original feature matrix into three separate matrices – U (an MxM orthogonal matrix), Sigma (M x N diagonal matrix) and V_Transpose (Transpose of n x n orthogonal matrix).[1] The code can be seen below:



```
from numpy import zeros, diag

X_combined_SVD_MLR = X_combined
def Manual_SVD2(X_combined_SVD_MLR,num_components):
    U, s, Vh = np.linalg.svd(X_combined_SVD_MLR)
    Sigma = zeros((X_combined_SVD_MLR.shape[0], X_combined_SVD_MLR.shape[1]))
    Sigma[:num_components, :num_components] = diag(s[:num_components])
    Sigma_k = Sigma[:,:num_components]
    Vh_reduced = Vh[:num_components, :]
    X_svd=U.dot(Sigma_k)
    return X_svd, U.shape, s.shape, Vh.shape

X_svd = Manual_SVD2(X_combined_MLR,10)
X_svd
```

3.2 Optimisation

Gradient Descent

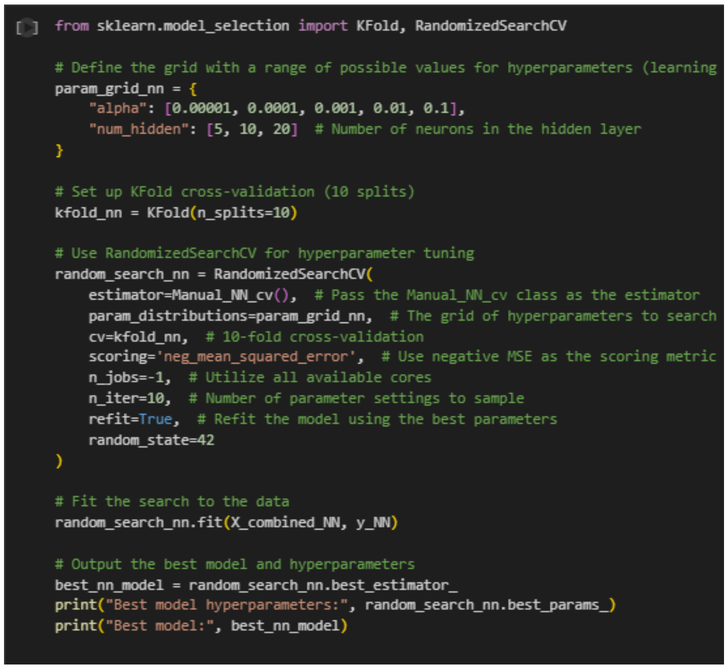
For both models, I used gradient descent as my optimisation algorithm. As I didn't have an excessive amount of data I didn't need to use stochastic gradient descent. My model uses the functions I defined above to iteratively update the weights and bias terms using matrix multiplication and this minimises the Mean Squared Error (MSE) between the predicted and actual target values. For each epoch, I use the partial derivate of the loss functions (gradients) and multiplied this by the learning rate. This continued until num_iterations was reached.

Early Stopping to avoid Overfitting

For my Neural Network I also decided to include an early stopping method which uses a patience of 10 iterations and stops after validation cost beings into increase as this points to overfitting.

Hyperparameter tuning

For my Manual_MLR class I used the learning rate (alpha) as my hyperparameter for tuning. Due to the computational cost, I used Random Search over Grid Search. I set a range for each hyperparameter and used 10-fold cross validation and then iterated over the dataset until I got the best model and hyperparameters. For my neural network hyperparameters I chose to include the number of hidden layers too.

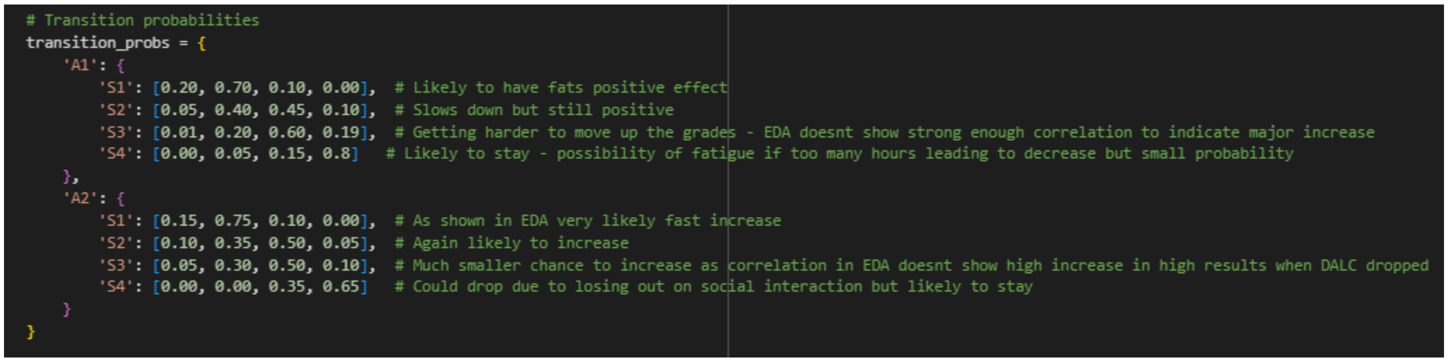


```
from sklearn.model_selection import KFold, RandomizedSearchCV

# Define the grid with a range of possible values for hyperparameters (learning
param_grid_nn = {
    "alpha": [0.00001, 0.0001, 0.001, 0.01, 0.1],
    "num_hidden": [5, 10, 20]  # Number of neurons in the hidden layer
}

# Set up KFold cross-validation (10 splits)
kfold_nn = KFold(n_splits=10)

# Use RandomizedSearchCV for hyperparameter tuning
random_search_nn = RandomizedSearchCV(
    estimator=Manual_NN_cv(),  # Pass the Manual_NN_cv class as the estimator
    param_distributions=param_grid_nn,  # The grid of hyperparameters to search
    cv=kfold_nn,  # 10-fold cross-validation
    scoring='neg_mean_squared_error',  # Use negative MSE as the scoring metric
    n_jobs=-1,  # Utilize all available cores
    n_iter=10,  # Number of parameter settings to sample
    refit=True,  # Refit the model using the best parameters
    random_state=42
)

# Fit the search to the data
random_search_nn.fit(X_combined_NN, y_NN)

# Output the best model and hyperparameters
best_nn_model = random_search_nn.best_estimator_
print("Best model hyperparameters:", random_search_nn.best_params_)
print("Best model:", best_nn_model)
```

3.3 Markov Decision Process

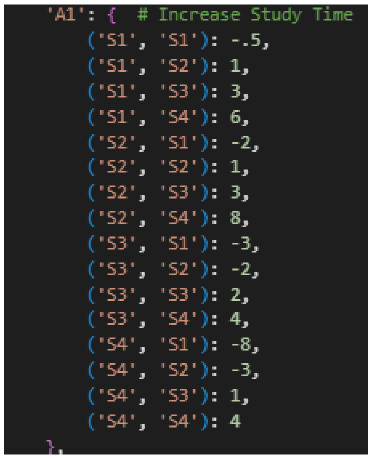
For my Markov decision process, I decided to examine two of the key features that I noticed had a strong correlation with G3. I used a policy iteration approach. I set my states as [**S1**: Failing, **S2**: Barely Passing, **S3**: Performing Well, **S4**: Excellent] and my actions as [**A1**: Increase Study Time, **A2**: Decrease Daily Alcohol Consumption]. When using a MDP I was confused about the transitional probabilities at first but after some research, I discovered I was not just picking the highest transition probability but picking the transition probability that associated with which actions lead to the best long-term results. [8]

As there were no prior transitional probabilities available in the data, I created my own based on trends and analysis made in my exploratory analysis. They can be seen below:

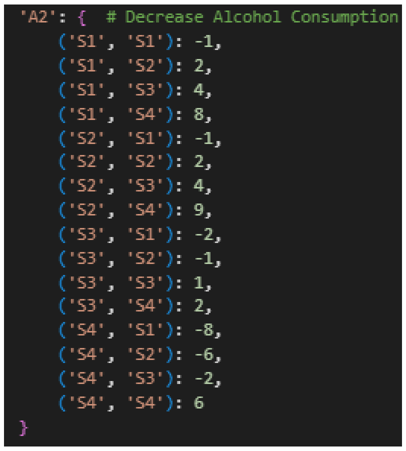


```
# Transition probabilities
transition_probs = {
    'A1': {
        'S1': [0.20, 0.70, 0.10, 0.00],  # Likely to have fats positive effect
        'S2': [0.05, 0.40, 0.45, 0.10],  # Slows down but still positive
        'S3': [0.01, 0.20, 0.60, 0.19],  # Getting harder to move up the grades - EDA doesnt show strong enough correlation to indicate major increase
        'S4': [0.00, 0.05, 0.15, 0.8]   # Likely to stay - possibility of fatigue if too many hours leading to decrease but small probability
    },
    'A2': {
        'S1': [0.15, 0.75, 0.10, 0.00],  # As shown in EDA very likely fast increase
        'S2': [0.10, 0.35, 0.50, 0.05],  # Again likely to increase
        'S3': [0.05, 0.30, 0.50, 0.10],  # Much smaller chance to increase as correlation in EDA doesnt show high increase in high results when DALC dropped
        'S4': [0.00, 0.00, 0.35, 0.65]   # Could drop due to losing out on social interaction but likely to stay
    }
}
```

My Reward functions:



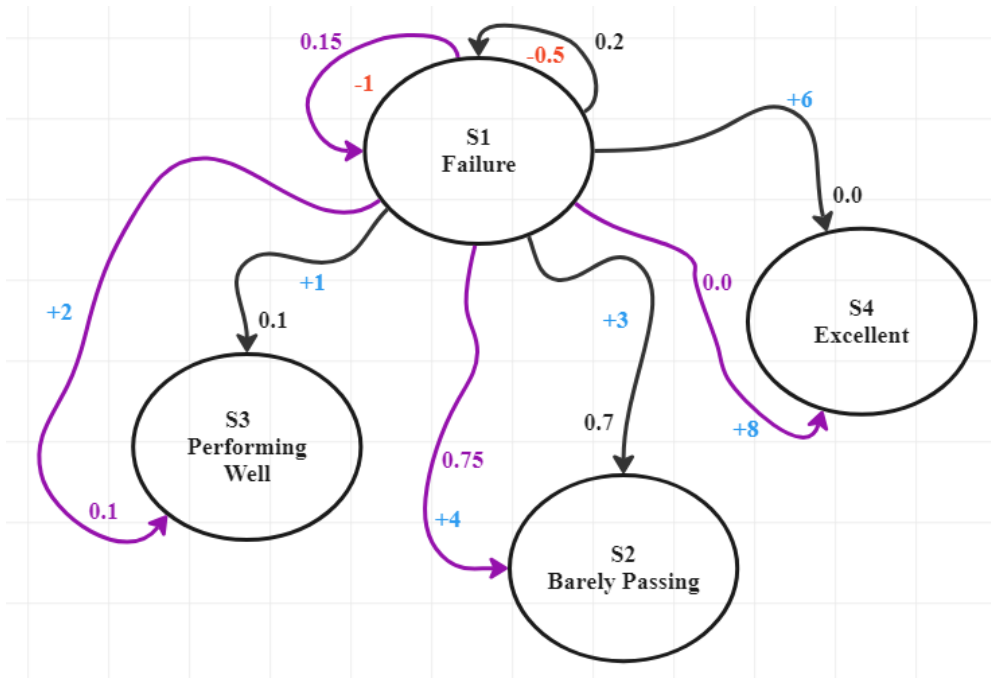
```
'A1': {  # Increase Study Time
    ('S1', 'S1'): -.5,
    ('S1', 'S2'): 1,
    ('S1', 'S3'): 3,
    ('S1', 'S4'): 6,
    ('S2', 'S1'): -2,
    ('S2', 'S2'): 1,
    ('S2', 'S3'): 3,
    ('S2', 'S4'): 8,
    ('S3', 'S1'): -3,
    ('S3', 'S2'): -2,
    ('S3', 'S3'): 2,
    ('S3', 'S4'): 4,
    ('S4', 'S1'): -8,
    ('S4', 'S2'): -3,
    ('S4', 'S3'): 1,
    ('S4', 'S4'): 4
},
```



```
'A2': {  # Decrease Alcohol Consumption
    ('S1', 'S1'): -1,
    ('S1', 'S2'): 2,
    ('S1', 'S3'): 4,
    ('S1', 'S4'): 8,
    ('S2', 'S1'): -1,
    ('S2', 'S2'): 2,
    ('S2', 'S3'): 4,
    ('S2', 'S4'): 9,
    ('S3', 'S1'): -2,
    ('S3', 'S2'): -1,
    ('S3', 'S3'): 1,
    ('S3', 'S4'): 2,
    ('S4', 'S1'): -8,
    ('S4', 'S2'): -6,
    ('S4', 'S3'): -2,
    ('S4', 'S4'): 6
}
```

These were also based on prior findings.

Below is a diagram I created using Miro to give an example of the Markov Decision Process from State 1 showing the transition probabilities from each state based on each action and the reward associated with each transition. [7]



Example of Actions and Transition Probabilities from State 1

4. Results and Discussion:

Testing Models:

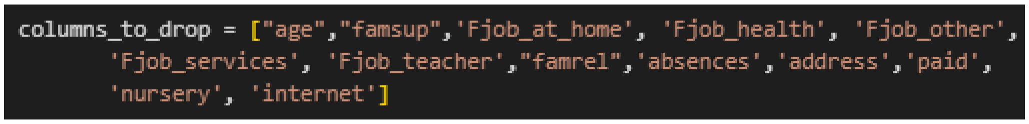
Model	**No of Features**	**Optimisation Method**	**Hyper-parameter Tuning**	**MSE on Test set**	**R^2 on Test set**
Multivariate Linear Regression	43 (All features without G1 and G2, one_hot ec)	Gradient Descent	No	8.1786	0.00208
Manual Neural Network	43	Gradient Desc using Backpropogation, RelU as activation funct and early stopping	No	5.766	0.139
Multivariate Linear Regression using SVD (10 Components)	43 (10 componets)	Gradient Descent	No	7.7214	0.05786
Manual Neural Network using SVD	43 (10 componets)	Gradient Desc using Backpropogation, RelU as activation funct and early stopping	No	5.90333	0.119
Multivariate Linear Regression with hyperparmeter tuning	43	Gradient Descent	Yes	6.3811	0.2214
Manual Neural Network with	43	Gradient Desc using Backpropogation, RelU as activation funct	Yes	6.3905	0.2203

hyperparameter Tuning		(Didnt have early stopping)			
Optimal Model	30 (8 componets)	Gradient Desc using Backpropogation, RelU as activation funct and early stopping	Yes	5.523	0.1757

These results were very interesting. The original models both had higher MSE with the NN having a better MSE score signifying that the non-linear aspect is helping to model some of the more complex relationships. The SVD models show that SVD has successfully captured the information of all 43 features into 10 components without majorly affecting the MSE, in the case of MLRE it actually improved the MSE. Finally, the tuning of the learning rate improves MSE scores for both methods but majorly improves the R^2 score. There is no doubt that as I applied the mathematical techniques to the data, the overall model improved and MSE scores decreased.

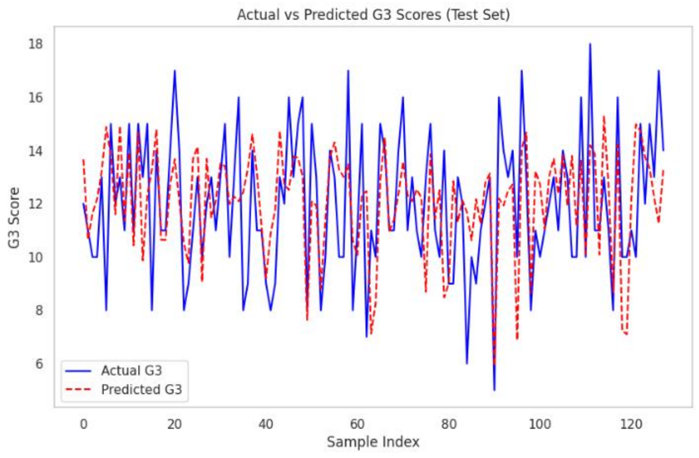
Optimal Model:

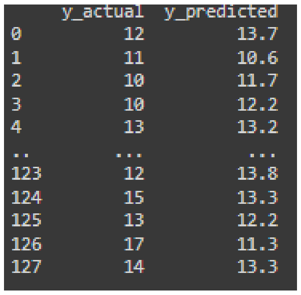
The final optimal model was a combination of the models above and the analysis done during EDA. It used my manual NN with tuned hyperparameters of learning rate set to 0.0001 and num_hidden = 5. I also removed a number of columns which following my previous analysis, I felt didn't offer valuable insight for my model. They were



```
columns_to_drop = ["age","famsup",'Fjob_at_home', 'Fjob_health', 'Fjob_other',
       'Fjob_services', 'Fjob_teacher',"famrel",'absences','address','paid',
       'nursery', 'internet']
```

All the results in this section are all from the optimal model.



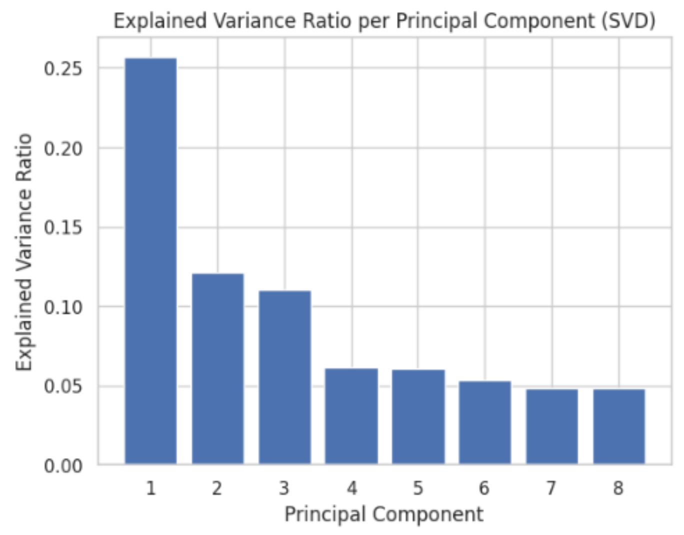


```
     y_actual  y_predicted
0          12         13.7
1          11         10.6
2          10         11.7
3          10         12.2
4          13         13.2
..        ...          ...
123        12         13.8
124        15         13.3
125        13         12.2
126        17         11.3
127        14         13.3
```

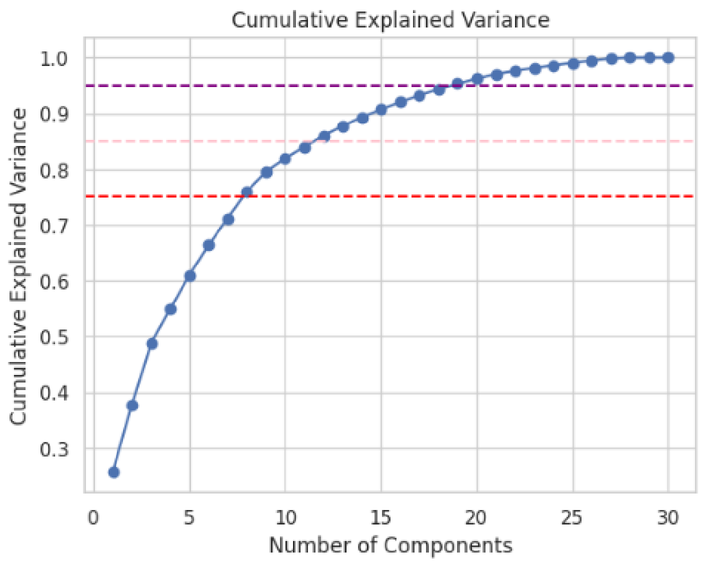
The results above show that the optimal model is performing well and has detected noise quite well but is also capable of predicting some of the results that are further from the mean.

4.1 SVD Results

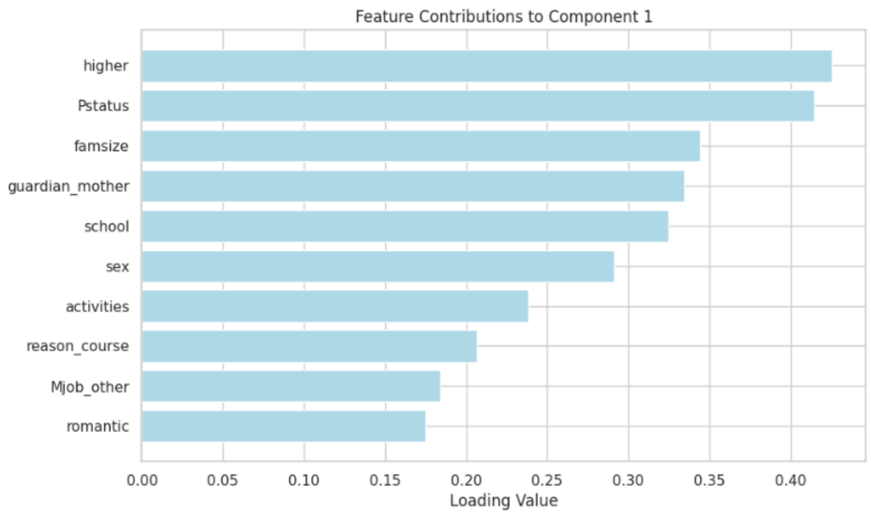
To examine the components, I first checked the shape of the three matrices. U had a shape of (634,634) which was expected as it was a MxM matrix. I converted s into a sigma matrix which contained my principal components along the diagonals. Finally, I checked the Vh matrix and had shape (43x43) so I was able to proceed. I calculated the explained variance ratio using the formula (s ** 2) / np.sum(s ** 2) which is the variance of each component divided by the sum of all the variances and this helped me select the number of components to include. The cumulative variance then told me how much of the total variance explained by the first n components. the results can be seen below:

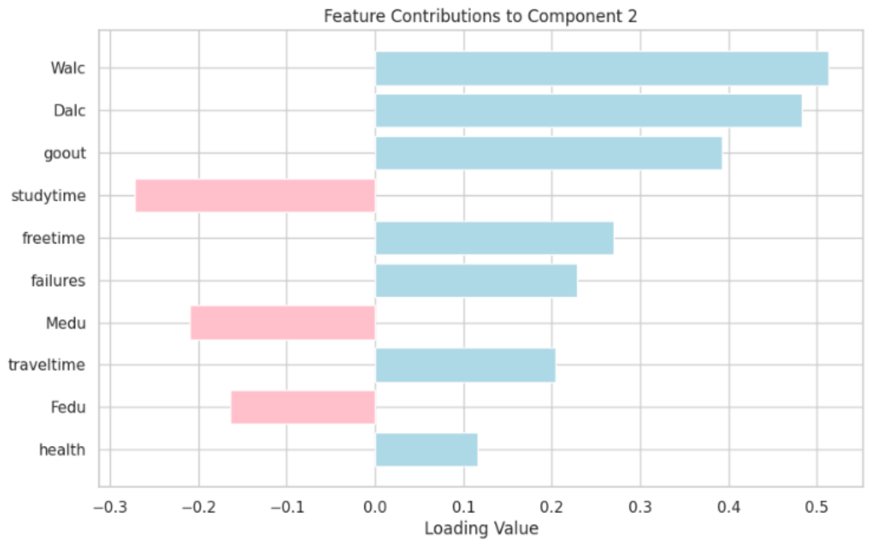


```
Total Variance Covered by the first 8 components: 0.7603
```



The plots above show the explained and cumulative variance for the components of the optimal model. I chose 8 components as this covers roughly 75% of the variance. I have three lines across the cumulative variance plot at 75%, 85% and 95%. To analyse the important features in my key components I plotted a bar chart of the loading value of each feature for the first two components.

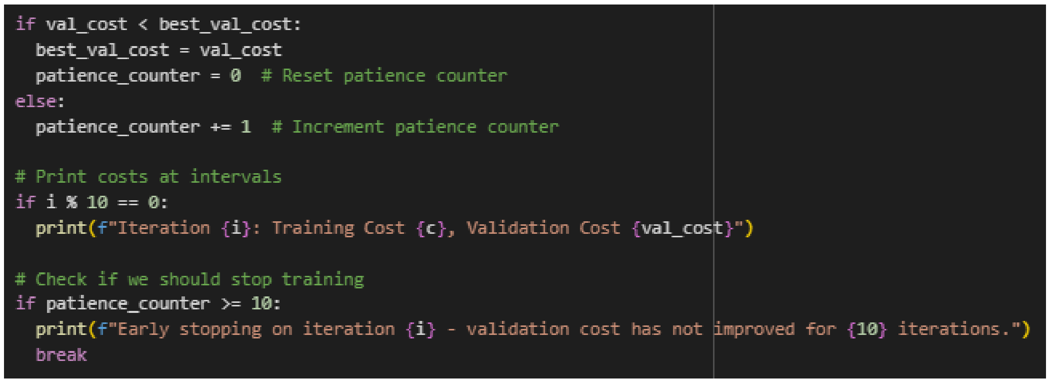




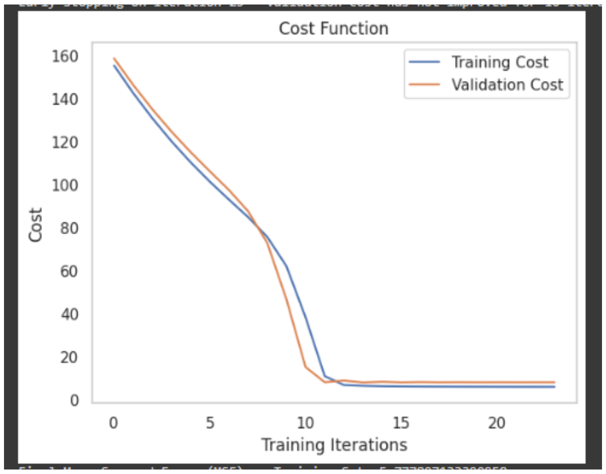
The key components were in line with the analysis from my EDA. The violin plot showing the distribution difference in the catgories of higher looks to have been a valuable insight. I was slightly surprised by famsize having such a big loading value in the key component.

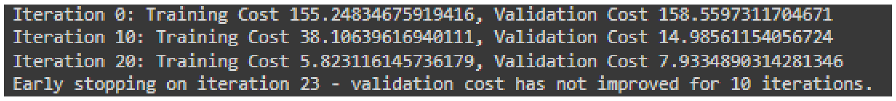
4.2 Optimisation

For my optimisation results I examine the cost function which plotted the training and validation cost. I used a patience counter which I set to 10 and if the validation cost failed to for 10 iterations, then the model stopped iterating. This was added as a measure as it showed convergence whilst also stopping before overfitting started taking place.



```
if val_cost < best_val_cost:
  best_val_cost = val_cost
  patience_counter = 0  # Reset patience counter
else:
  patience_counter += 1  # Increment patience counter

# Print costs at intervals
if i % 10 == 0:
  print(f"Iteration {i}: Training Cost {c}, Validation Cost {val_cost}")

# Check if we should stop training
if patience_counter >= 10:
  print(f"Early stopping on iteration {i} - validation cost has not improved for {10} iterations.")
  break
```



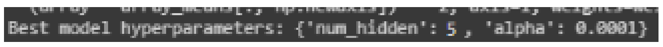


```
Iteration 0: Training Cost 155.24834675919416, Validation Cost 158.5597311704671
Iteration 10: Training Cost 38.10639616940111, Validation Cost 14.98561154056724
Iteration 20: Training Cost 5.823116145736179, Validation Cost 7.9334890314281346
Early stopping on iteration 23 - validation cost has not improved for 10 iterations.
```

Both the training and validation loss drop quickly, which showed a quick convergence from the initial random weights which was expected. In the next iterations the cost drop decreases, and this is because my model is moving closer to the minimum of the cost function. The model stops after 23 iterations. I had tested with 2000 and this only led to overfitting and cost did not show major improvement passed 23 which shows convergence, so I stopped here.

Hyperparameter Tuning:

As this model was a NN, I tuned the number of hidden layers and the learning rate. The results were as follows:

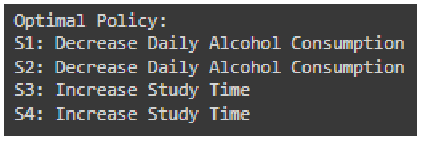


```
Best model hyperparameters: {'num_hidden': 5 , 'alpha': 0.0001}
```

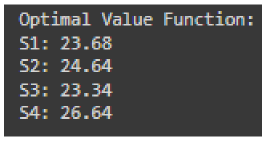
5 hidden layers proved to give the best results as 10 layers seemed to lead to overfitting and a single layer didn't have enough complexity to capture some of the more complex relationships. The model also used a small alpha which allowed a slower learning rate and this provided a better fit to the training data.

4.3 Markov Decision Process

The Optimal Policy was as follows:



```
Optimal Policy:
S1: Decrease Daily Alcohol Consumption
S2: Decrease Daily Alcohol Consumption
S3: Increase Study Time
S4: Increase Study Time
```



```
Optimal Value Function:
S1: 23.68
S2: 24.64
S3: 23.34
S4: 26.64
```

The key takeaways here show that for students with a low G3 score trying to improve or get a passing grade, they would be best suited to reducing their daily alcohol intake as a first measure. This approach is more effective when students are aiming to improve lower grades. However, once students reach the final two states—where their grades are already higher—the greatest return on their time investment comes from increasing study time. From my results, the clearest decision to make comes at state 4, as this has the highest reward of 26.64 over time.

4.4 Final Discussion of Mathematical Concepts:

Throughout the report I discussed my results and y findings but here I will do an overall summary of the mathematical concepts here:

- **Outliers:** One step improved MSE significantly initially was dealing with the outliers in the G3. As the MSE is very prone to outliers, they were affecting my results.
- **Model:** NN's nonlinear component was more effective than just linear regression.
- **EDA:** Key findings from EDA indicated removing G1 and G2 scores.
- **Correlation Analysis:** Helped reveal strong relationships between features and helped identify significant predictors for G3
- **Hypothesis Testing:** My t-tests and ANOVA demonstrated significant differences among groups, showing that variables like school type, study time, and parental job types provided meaningful insights for my model.
- **Singular Value Decomposition (SVD):** SVD allowed me to reduce my dimensionality, enabling the model to retain essential information while avoiding the curse of dimensionality. The MSE scores were not negatively effected, showing that it worked as I wanted.
- **Gradient Descent Optimisation:** Using gradient descent for weight updates allowed me to iteratively improve my model, by reaching a minimum of the cost function and finding optimal weights.
- **Hyperparameter Tuning:** Adjusting the learning rate and number of hidden layers showed how hyperparameter optimisation improved my model, leading to lower mean squared error (MSE) and improved R^2 scores.
- **Markov Decision Process (MDP):** My MDP evaluated the impact of different interventions on G3 scores, highlighting the importance of reducing alcohol consumption and increasing study time as effective strategies for improving student performance at different levels.

5. Ethical Considerations

One important ethical consideration in this project was explainable AI. To make my processes as clear and transparent as possible I created well-structured classes and functions and ensured that my code was thoroughly commented. I also took measures to prevent bias in my model by carefully examining the dataset to ensure no inherent biases were present. For example, I removed potentially sensitive features like addresses to avoid disadvantaging specific groups. Another concern was that this dataset contained sensitive student data, so proper data anonymization was essential. Overall, I made a strong effort to ensure there were no ethical concerns related to the use of explainable AI by clearly outlining my methodology and justifying the decisions made by my code.

6. Word Count (Excluding Front Page, References, Figure Titles, Titles and Tables): **3271**

References

1. Bai, L (2024), Lecture Notes 1-12 *ECS8050* Queens University Belfast 16 September-13 October.
2. Astanin, S. (2022). tabulate: Pretty-print tabular data. [online] PyPI. Available at: https://pypi.org/project/tabulate/.
3. atoti. (2021). Machine Learning: When to perform a Feature Scaling? [online] Available at: https://www.atoti.io/articles/when-to-perform-a-feature-scaling/.
4. Brownlee, J. (2018). How to Calculate the SVD from Scratch with Python - MachineLearningMastery.com. [online] MachineLearningMastery.com. Available at: https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning [Accessed 07 Oct. 2024].
5. GeeksforGeeks (2022). Welch's tTest in Python. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/welchs-t-test-in-python [Accessed 4 Oct. 2024].
6. Medium. (2024). Medium. [online] Available at: https://hersanyagci.medium.com/feature-scaling-with-scikit-learn-for-data-science-8c4cbcf2daff. [Accessed 1 Oct. 2024].
7. Miro (n.d.). Team collaboration software & online whiteboard for teams | Miro. [online] https://miro.com/. Available at: https://miro.com/.
8. Umberto Michelucci (2024). Fundamental Mathematical Concepts for Machine Learning in Science. Springer.

Code:

1. "9. Classes — Python 3.8.4rc1 Documentation." Docs.python.org, docs.python.org/3/tutorial/classes.html.
2. brKdgl. "How to Add My Customized Class to Sklearn Pipeline Properly?" Stack Overflow, 2024, stackoverflow.com/questions/76799442/how-to-add-my-customized-class-to-sklearn-pipeline-properly.
3. Chanaka Prasanna. "Evaluating Model Performance with K-Fold Cross-Validation — a Practical Example." Medium, Aug. 2024, medium.com/@chanakapinfo/evaluating-model-performance-with-k-fold-cross-validation-a-practical-example-485aaeb01dc0.
4. Coursesteach. "Deep Learning (Part 25)-Derivatives of Activation Functions." Medium, 7 Feb. 2024, medium.com/@Coursesteach/deep-learning-part-25-derivatives-of-activation-functions-4bbd7c7c7a1c.
5. Curious2learn. "What Is the Purpose of Static Methods? How Do I Know When to Use One?" Stack Overflow, 2024, stackoverflow.com/questions/2438473/what-is-the-purpose-of-static-methods-how-do-i-know-when-to-use-one.
6. "Machine Learning - Perceptron." Tutorialspoint.com, 2024, www.tutorialspoint.com/machine_learning/machine_learning_perceptron.htm.
7. Mousa, Waleed. "Building a Neural Network from Scratch Using Numpy and Math Libraries: A Step-By-Step Tutorial In…." Medium, 15 Mar. 2023, medium.com/@waleedmousa975/building-a-neural-network-from-scratch-using-numpy-and-math-libraries-a-step-by-step-tutorial-in-608090c20466.
8. "Markov Decision Process: Policy Iteration with Code Implementation." Medium, 20 Dec. 2021, medium.com/@ngao7/markov-decision-process-policy-iteration-42d35ee87c82.SS-YS.
9. "MDP-With-Value-Iteration-And-Policy-Iteration/PolicyIteration.py at Main · SS-YS/MDP-With-Value-Iteration-And-Policy-Iteration." GitHub, 2021, github.com/SS-YS/MDP-with-Value-Iteration-and-Policy-Iteration/blob/main/policyIteration.py.

Appendix – Code

```
# -*- coding: utf-8 -*-
"""ECS8050.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1ps1tf0wCGDKdRrcJdRpUPRt8zCe0BQ7T

## Preprocessing

Loading in the data from my google drive and examining the format
"""

import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

file_path2 = '/content/drive/MyDrive/Queens/Module1/Project/student-por.csv'
dfp = pd.read_csv(file_path2,sep=";")

"""Storing all my imports here"""

from tabulate import tabulate
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import pearsonr, spearmanr, shapiro, f, ttest_ind, chi2_contingency, f_oneway
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split, KFold, RandomizedSearchCV

from sklearn.metrics import mean_squared_error, r2_score

from numpy import diag, zeros

from sklearn.base import BaseEstimator, RegressorMixin



print(dfp.head())

dfp.shape



dfp.columns



object_columns = dfp.columns[dfp.dtypes == 'object']



print(dfp[object_columns].head())



dfp.dtypes



"""## Make Table



Creating a table for my report uisng the tabulate library

"""



data = [["school", "Binary", "Object", "Students School: GP or MS"],

        ["sex", "Binary", "Object", "Students Sex - Female(F)/Male(M)"],

        ['age', "Integer", "int64", "Age - From 15 to 22"],

        ['address',"Binary","Object", "Urban(U) or Rural(R)"],

        ["famsize", "Binary","Object", "LE3' - less or equal to 3 or 'GT3' - greater than 3)" ],

        ['Pstatus', "Binary","Object", "Parents' Cohabitation - (binary: 'T' - living together or 'A' - apart)"],
['Medu',"Integer","int64","Mothers Education 0-4 (None to Higher Education Scaled)"], ['Fedu',"Integer","int64", "Fathers Education
0-4 (None to Higher Education Scaled)"],

       ['Mjob',"Categorical","Object", "Mothers Job - Categorical Options"], ['Fjob',"Categorical","Object", "Fathers Job - Categorical
Options"], ['reason',"Categorical","Object","Reason for selecting school"], ['guardian',"Categorical","Object","Student's Guardian"],
['traveltime',"Integer","int64", "1-4 staggered (1 < 15mins)(4 > 1 hour)"], ['studytime',"Integer","int64", "Weekly study 1-4
(Staggered)(1 = <2hours)(4=<10 hours)"],
```

['failures',"Integer","int64", "Number of past failures - More than 3 falls into the 4 label "], ['schoolsup',"Binary","Object","Extra educational support"], ['famsup',"Binary","Object","family educational support"], ['paid',"Binary","Object","extra paid classes within Portuguese"], ['activities',"Binary","Object", "extra-curricular activities "], ['nursery',"Binary","Object","attended nursery school"],

['higher',"Binary","Object","wants to take higher education"], ['internet',"Binary","Object","Internet access at home"], ['romantic',"Binary","Object","In a romantic relationship"], ['famrel',"Integer","int64","quality of family relationships (from 1 - very bad to 5 - excellent)"], ['freetime',"Integer","int64","free time after school (from 1 - very low to 5 - very high)"], ['goout',"Integer","int64","going out with friends (from 1 - very low to 5 - very high)"], ['Dalc',"Integer","int64","workday alcohol consumption (from 1 - very low to 5 - very high)"],

['Walc',"Integer","int64","weekend alcohol consumption (from 1 - very low to 5 - very high)"], ['health',"Integer","int64","current health status (from 1 - very bad to 5 - very good)"], ['absences',"Integer","int64", "number of school absences (from 0 to 93)"], ['G1',"Integer","int64","first period grade (from 0 to 20)"], ['G2',"Integer","int64","second period grade (from 0 to 20)"], ['G3',"Integer","int64","final grade (from 0 to 20) This is the output target"]]

#define header names

col_names = ["Variable_Name","Type", "Data_Type_Python", "Description"]

print(tabulate(data, headers=col_names, tablefmt="pipe" ,showindex="always"))

"""## Checking for **Nulls**"""

dfp

dfp.count()

dfp.describe()

"""There are no missing values so can proceed with EDA"""

dfp.isnull().sum()

"""Want to examine these zero values so use indexing to do so"""

dfp[dfp["G3"] == 0]

"""## Analysing G3 also plot g1 and g2

```
Comapare the G scores using a boxplot

"""

dfp.boxplot(["G1","G2","G3"])

"""Here we can see that G1 and G2 have very similar distributions. There seems to be a cluster of outliers at G3 which I will examine further."""

plt.figure(figsize=(10,6))

sns.histplot(dfp['G3'], bins=20, kde=True, color='blue')
plt.xlabel("Final Grade")
plt.ylabel("Frequency")
plt.title("Distribution of Final Grades")

"""There is a clear left skew in the distribution here - data resembles normal distribution with exception to the 0 tail."""

dfp[dfp["G3"] == 0]

"""Based on data description above it is highly unlikely that a student would recieve a 0 score for G3. My only thought would be that the students didnt show up for their final exam but they all have absesnses of 0. Therefore, I am going to treat this data as false and as outliers and it will not help my model in it's current state. Rather than remove the data I am going to replace the data with the mean value of G1 and G2."""

#I want to plot data before zeros are removed
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(dfp['G3'], bins=20, kde=True, color='blue')
plt.title('Original G3 Distribution')
plt.xlabel('Final Grade')
plt.ylabel('Frequency')
```

```
# Data after removing 0 values
dfp = dfp[dfp['G3'] > 0]  # Here I am removing the 0 G3 scores rows
plt.subplot(1, 2, 2)
sns.histplot(dfp['G3'], bins=20, kde=True, color='orange')
plt.title('Cleaned G3 Distribution')
plt.xlabel('Final Grade')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 6))
stats.probplot(dfp["G3"], dist="norm", plot=plt)
plt.title("Q-Q Plot of Cleaned G3 Scores")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.grid()
plt.show()

"""From this we can see that the data G3 follows a roughly normal disttribugtion and we can use this in our probailistic tests"""

# Calculate corrleation just to examine
pearson_corr_g1_g3, _ = pearsonr(dfp['G1'], dfp['G3'])
pearson_corr_g2_g3, _ = pearsonr(dfp['G2'], dfp['G3'])

print("Pearson Correlation:")
print(f"G1 and G3: {pearson_corr_g1_g3}")
print(f"G2 and G3: {pearson_corr_g2_g3}")

plt.figure(figsize=(12, 6))

# I am using the seaborn library to plot a jointplot of G1 and G3
```

```
g1 = sns.JointGrid(data=dfp, x="G1", y="G3")

g1.plot(sns.scatterplot, sns.histplot, color="red", alpha=0.5)

g1.ax_joint.set_xlabel("G1")

g1.ax_joint.set_ylabel("G3")

g1.ax_joint.set_title("G1 vs G3")


# Same as above but for G2 and G3

plt.figure(figsize=(12, 6))

g2 = sns.JointGrid(data=dfp, x="G2", y="G3")

g2.plot(sns.scatterplot, sns.histplot, color="blue", alpha=0.5)

g2.ax_joint.set_xlabel("G2")

g2.ax_joint.set_ylabel("G3")

g2.ax_joint.set_title("G2 vs G3")


plt.tight_layout()

plt.show()


"""From this we can see that there is a strong correlation between G1 and G2 which could lead to multicollineraity and also as
students new wouldnt have these scores we will remove them


## Univariate Analysis of features distributions

"""


#Extracting my numerical features based on types only

numerical_features = dfp.select_dtypes(include=['float64', 'int64']).columns[:15]  # Take the 15

n_num_features = len(numerical_features)

plt.figure(figsize=(15, 10))


# Here I am using a loop to plot the figures in a tidy fashion

for i, feature in enumerate(numerical_features):

    plt.subplot(3, 5, i + 1)

    sns.histplot(dfp[feature], bins=20, kde=True, color='blue')

    plt.title(f'{feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')


plt.tight_layout()
plt.show()


# This is my univariate analysis of my categorical features.
# I am just plotting countplots of all the features
plt.figure(figsize=(20, 15))
categorical_features = dfp.select_dtypes(include=['object']).columns
n_cat_features = len(categorical_features)


for i, feature in enumerate(categorical_features):
    plt.subplot(4, 8, i + 1)  # This was put in to fix my problem of how many plots per row
    sns.countplot(data=dfp, x=feature, order=dfp[feature].value_counts().index)
    plt.title(f'Count of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability so that labels aren't on top of each other


plt.tight_layout()
plt.show()


"""## “Correlation does not imply causation.” - numerical"""


# Here I am plotting a correlation matrix of all my numeric features
numerical_dfp = dfp.select_dtypes(include=['int64', 'float64']) # I did this as correlation could not be computed for categorical or float types
corr = numerical_dfp.corr()


plt.figure(figsize=(12, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()


#For thsi to work I needed to split the G3 category into subsections
bins = [0, 4, 8, 12, 16, 20]
labels = ['0-4', '5-8', '9-12', '13-16', '17-20']


dfp['G3_category'] = pd.cut(dfp['G3'], bins=bins, labels=labels, include_lowest=True) #This then creates a new category column for me


age_g3_counts = dfp.groupby(['age', 'G3_category']).size().unstack(fill_value=0) # uisng the groupby here to use both age and G3 category


age_g3_counts.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='viridis') # Plot my stacked bar chart


plt.title('Stacked Bar Chart of Age with G3 Categories')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='G3 Score Range')


#Map the ordinal data 1-4 to it's meaning
traveltime_mapping = {
    1: "<15 mins",
    2: "15 to 30 mins",
    3: "30 mins to 1 hour",
    4: ">1 hour"
}


#I assign the variable label with traveltime data
label = dfp['traveltime'].map(traveltime_mapping)
plt.figure(figsize=(13, 8))
sns.set(font_scale=1.6)
sns.set_style('white')
sns.scatterplot(y='G3', x='absences', hue=label, data=dfp,s=50 ,palette=["lightblue", "blue", "orange","red"]) # Set hue as the traveltime labels
```

```
plt.xlabel('Absences')

plt.ylabel('Final Grade - G3')

plt.title('Relation Between Grades And Absences by Travel Time')

plt.legend(title='Travel Time')

plt.show()


numerical_dfp = dfp.select_dtypes(include=['int64', 'float64']) #Craete a dataframe of just numerical features so I can reference this


corr = numerical_dfp.corr() #I create the correlation matrix

corr_df = corr.unstack().reset_index()

corr_df.columns = ['Feature_1', 'Feature_2', 'Correlation'] #rename columns

corr_df = corr_df[corr_df['Feature_1'] != corr_df['Feature_2']] #I have no interest in self correlations as these equal 1

strong_corr_df = corr_df[abs(corr_df['Correlation']) > 0.5] # Strong correlatiosn are over 0.5

strong_corr_df = strong_corr_df.sort_values(by='Correlation', key=abs, ascending=False) #I put them in descending order


print("Strongest Correlations:")

print(strong_corr_df)


sns.set(style="whitegrid") #Style

plt.figure(figsize=(12, 6))

sns.barplot(data=dfp, x='Medu', y='G3', hue='failures',palette='bright') # I create Simple Bar chart with hue set to failures

plt.title('Bar Plot of Final Grade (G3) by Mother\'s Education Level (Medu) with Failures as Hue')

plt.xlabel('Mothers Education Level (Medu)')

plt.ylabel('Final Grade (G3)')


plt.legend(title='Number of Failures', loc='upper left', bbox_to_anchor=(1, 1)) # reasearched this to be able to move the legend to the
side as it was in the way originally

plt.show()


"""## Categorical"""


Categorical_dfp = dfp.select_dtypes(include=['object']) # Same as my numerical process but categorical fetaures this time
```

```
Categorical_dfp.columns


"""Want to examine some of the key reasons"""


sns.countplot(data=dfp, x='reason', order=dfp['reason'].value_counts().index) # Simple countplot of the reasons


"""Boxplot of schools by G3 score"""


plt.figure(figsize=(8, 6))
sns.boxplot(x='school', y='G3', data=dfp)
plt.title('G3 Scores by School')
plt.xlabel('School')
plt.ylabel('G3 Score')
plt.show()


categorical_columns = Categorical_dfp.columns  # I am extracting column names
num_columns_per_row = 4 #For my display
num_columns = len(categorical_columns)
num_rows = (num_columns + num_columns_per_row - 1) // num_columns_per_row  # So that it will fit in my report


fig, axes = plt.subplots(num_rows, num_columns_per_row, figsize=(16, 4 * num_rows))
axes = axes.flatten()  # Had to do this for it to work


for i, col in enumerate(categorical_columns):
    mean_g3 = dfp.groupby(col)['G3'].mean()
    mean_g3.plot(kind='bar', ax=axes[i])
    axes[i].set_title(f'Average G3 Scores by {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Average G3 Score')
    axes[i].tick_params(axis='x', rotation=0)  #Loop which plots each mean G3 score by category


for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j]) #remove any unwanted ones


plt.tight_layout()

plt.subplots_adjust(hspace=0.4, wspace=0.3)  # I Add space between plots

plt.show()


"""Boxplot of G3 scores by school"""


plt.figure(figsize=(8, 6))

sns.boxplot(x="sex",y="age",hue="G3_category",data=dfp)

plt.title('G3 Scores by Sex and Age')

plt.xlabel('Sex')

plt.ylabel('Age')

plt.legend(title='G3 Category', fontsize='small', title_fontsize='medium', loc='upper left', bbox_to_anchor=(1, 1)) # i am moving the legend again here

plt.show()


alc_mapping = { #Map the ordinal values again as I did before

    1: 'Very Low',

    2: 'Low',

    3: 'Moderate',

    4: 'High',

    5: 'Very High'

}


# I store my mappings in variable alc

alc = dfp['Dalc'].map(alc_mapping)




plt.figure(figsize=(10, 6))

sns.boxplot(x='sex', y='G3', hue=alc, hue_order= ['Very Low', 'Low', 'Moderate', 'High', 'Very High'],data=dfp, palette={

    'Very Low': 'lightblue',

    'Low': 'lightgreen',
    'Moderate': 'lightyellow', # I need to ensure colours match up so it is easy to notice differences
    'High': 'orange',
    'Very High': 'red'
})

plt.title('Gender vs Alcohol Consumption on Weekdays based on G3')
plt.xlabel('Gender')
plt.ylabel('Final Grades (G3)')
plt.ylim(0, 20)
plt.legend(title='Weekday Consumption', fontsize='small', title_fontsize='medium', loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

"""Want to examine this for all yes/no binary features sos that I can select during featur enginerring"""

# I extract all the binary columns
binary_columns = ["schoolsup","famsup","paid","activities","nursery","higher","internet","romantic"]

# This process is the same as previous which I explained above
num_columns = len(binary_columns)
plots_per_row = 4
num_rows = (num_columns + plots_per_row - 1) // plots_per_row
plt.figure(figsize=(15, 5 * num_rows))

for i, col in enumerate(binary_columns):
    plt.subplot(num_rows, plots_per_row, i + 1)
    sns.violinplot(x=col, y='G3', data=dfp, palette='muted')
    plt.title(f'Dist of G3 by {col.capitalize()}')
    plt.xlabel(col.capitalize())
    plt.ylabel('Final Grades (G3)')

plt.tight_layout()
plt.show()
plt.figure(figsize=(10, 6))

sns.violinplot(x='reason', y='G3', data=dfp, palette='muted') # Violin plot to show distribution

plt.title('Distribution of Final Grades (G3) by Reason for Choosing School')

plt.xlabel('Reason for Choosing School')

plt.ylabel('Final Grades (G3)')

plt.tight_layout()

plt.show()


colour_mapping = { # i need to map the colours again to ensure consistency in both graphs

    'teacher': 'lightblue',

    'health': 'lightgreen',

    'services': 'lightyellow',

    'at_home': 'orange',

    'other': 'pink'

}


job_order = ['teacher', 'health', 'services', 'at_home', 'other']


plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1) # I set this as the first subplot

sns.boxplot(x='Mjob', y='G3', data=dfp,order=job_order, palette=colour_mapping)

plt.title("G3 Scores by Mother's Job")

plt.xlabel('Mother\'s Job')

plt.ylabel('Final Grade (G3)')

plt.xticks(rotation=45)


plt.subplot(1, 2, 2) # I set this as the second subplot

sns.boxplot(x='Fjob', y='G3', data=dfp, order=job_order, palette=colour_mapping)

plt.title("G3 Scores by Father's Job")

plt.xlabel('Father\'s Job')

plt.ylabel('Final Grade (G3)')

plt.xticks(rotation=45)
plt.tight_layout()

plt.show()


plt.figure(figsize=(10, 6))


# I Calculate the mean G3 scores for both Mjob and Fjob

mean_g3_mjob = dfp.groupby('Mjob')['G3'].mean()

mean_g3_fjob = dfp.groupby('Fjob')['G3'].mean()

#Wasnt sure what way to approach this but this method worked best by craeting a DF

g3_jobs_df = pd.DataFrame({

    'Mother\'s Job': mean_g3_mjob,

    'Father\'s Job': mean_g3_fjob

})

g3_jobs_df.plot(kind='bar', figsize=(10, 6), width=0.8)


plt.title('Average G3 Scores by Mother\'s and Father\'s Job')

plt.xlabel('Job Category')

plt.ylabel('Average G3 Score')

plt.xticks(rotation=45) # Helps read labels

plt.tight_layout()

plt.show()


"""## Probabilistic Tests


T-test to examine means per school

"""


group1 = dfp[dfp['school'] == 'GP']['G3'] #I group data by school

group2 = dfp[dfp['school'] == 'MS']['G3']


alpha = 0.05 # I set the threshold at 5%
```

```
# I do a quick F-test to examine varinaces

var_gp = np.var(group1, ddof=1)  # Sample variance for GP

var_ms = np.var(group2, ddof=1)  # Sample variance for MS

# I Calculate the F-statistic

f_statistic = var_gp / var_ms

dof1 = len(group1) - 1  # Degrees of freedom for group1

dof2 = len(group2) - 1  # Degrees of freedom for group2

p_value_f_test = 1 - f.cdf(f_statistic, dof1, dof2) #This is the f-test - used an online source here https://www.geeksforgeeks.org/how-to-perform-an-f-test-in-python/

print(f"\nF-test for Equality of Variances:")

print(f"F-statistic: {f_statistic}, P-value: {p_value_f_test}")


# Here I set my threshold and if less than 5% I can reject the Null hypothesis of whether the varinaces are equal

if p_value_f_test < alpha:

    print("Reject the null hypothesis: The variances are significantly different.")

else:

    print("Fail to reject the null hypothesis: The variances are equal.")


# As F-test was clear I can perform normal t-test

# Here I am running a t-test to examine if there is a difference in the means based on school G3 scores

group1 = dfp[dfp['school'] == 'GP']['G3']

group2 = dfp[dfp['school'] == 'MS']['G3']


print(np.var(group1))

print(np.var(group2))

#I am setting alpha threshold to 5%

alpha = 0.05


t_stat, p_value = ttest_ind(group1, group2)

print(f"T-test: t-statistic = {t_stat}, p-value = {p_value:.9f}")


# Null hypothesis was set in report before test
```

```
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the groups.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference between the groups.")


"""This is a chi-square test to meausure if there is a significant association between these twi categorical variables"""


# I Create a contingency table for Mjob and higher education
contingency_table = pd.crosstab(dfp['Mjob'], dfp['Fjob'])


# I perform a chi-square test
chi2_stat, p_val, dof, ex = chi2_contingency(contingency_table)


print("Chi-Square Statistic:", chi2_stat)
print("P-value:", p_val)
print("Degrees of Freedom:", dof)
print("Expected Frequencies Table:")
print(ex)


alpha = 0.05


print(f"Chi-Square Statistic: {chi2_stat}, P-value: {p_val:.9f}")


if p_val < alpha:
    print("There is a signicant assoiciation between Mjob and Fjob.")
else:
    print("There is no signicant assoiciation between Mjob and Fjob.") #this concludes i COULD remove one as they are no
independent


#From my EDA plot above varinace appears roughly normal
#Follows normal distribution as stated in report
```

```
# I use ANOVA here to analyse means
anova_stat, p_val = f_oneway(dfp[dfp['Mjob'] == 'teacher']['G3'],
                             dfp[dfp['Mjob'] == 'health']['G3'],
                             dfp[dfp['Mjob'] == 'services']['G3'],
                             dfp[dfp['Mjob'] == 'at_home']['G3'],
                             dfp[dfp['Mjob'] == 'other']['G3'])

print(f"ANOVA F-statistic: {anova_stat}, P-value: {p_val:.7f}")
if p_val < alpha:
    print("Rejct the Null: There is a significant difference in G3 scores across the Mjob categories.")
else:
    print("Fail to rejct the Null: There is no significant difference in G3 scores across the Mjob categories.")

    #will be a useful feature for my model

# Here I separate G3 scores based on study time
g3_low_study = dfp[dfp['studytime'] <= 3]['G3']
g3_high_study = dfp[dfp['studytime'] > 3]['G3']
var_gp = np.var(g3_low_study, ddof=1)  # Sample variance for GP
var_ms = np.var(g3_high_study, ddof=1)  # Sample variance for MS

# Same as previous - same source
f_statistic = var_gp / var_ms
dof1 = len(g3_low_study) - 1  # Degrees of freedom for group1
dof2 = len(g3_high_study) - 1  # Degrees of freedom for group2
p_value_f_test = 1 - f.cdf(f_statistic, dof1, dof2)

print(f"\nF-test for Equality of Variances:")
print(f"F-statistic: {f_statistic:.5f}, P-value: {p_value_f_test:.5f}")

if p_value_f_test < alpha:
    print("Reject the null hypothesis: The variances are significantly different.")
else:
    print("Fail to reject the null hypothesis: The variances are equal.")


# Run my t-test
t_stat, p_val = ttest_ind(g3_low_study, g3_high_study)


print(f"T-statistic for Study Time: {t_stat:.5f}, P-value: {p_val:.5f}")
if p_val < alpha:
    print("Rejct the Null: There is a significant difference in G3 scores based on studytime.")
else:
    print("Fail to rejct the Null: There is no significant difference in G3 scores based on studytime.")


    # studytime will be a useful feature for my model


"""Need to examine if i shoudl use wleches t-test as varainces are not the sa,e"""


# Here I split DALC into two groups - high and low consumption
low_alc = dfp[dfp['Dalc'] <= 3]['absences']
high_alc = dfp[dfp['Dalc'] > 3]['absences']


# Run t-test for alcohol consumption
t_stat, p_val = ttest_ind(low_alc, high_alc,equal_var=False) # I use Welchs t-test here as varinaces are not equal
print(f"T-statistic for Alcohol Consumption (Dalc): {t_stat}, P-value: {p_val:.5f}")
if p_val < alpha:
    print("Rejct the Null: Daily Alchohol consumption affect attendence.")
else:
    print("Fail to rejct the Null: Daily Alchohol consumption does not affect attendce.")


# I Split age into groups using the median
median_age = dfp['age'].median()
g3_younger = dfp[dfp['age'] <= median_age]['G3']
g3_older = dfp[dfp['age'] > median_age]['G3']
t_stat_age, p_val_age = ttest_ind(g3_younger, g3_older)
```

```
print(f"T-statistic for Age: {t_stat_age}, P-value: {p_val_age:.5f}")


if p_val_age < alpha:
    print("Reject the Null: There is a significant difference in G3 scores based on age.")
else:
    print("Fail to rejct the Null: There is no significant difference in G3 scores based on age.")


"""## One_Hot Encoding"""


print(dfp) # A check that it still has right shape


X = dfp.drop(columns=['G1', 'G2', 'G3','G3_category'])  # Features
y = dfp['G3']


print(X.shape)


y.shape


"""Preparing data for encoding and preprocessing"""


categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
print(len(categorical_cols))
print(len(numerical_cols))


categorical_data = X[categorical_cols]
numerical_data = X[numerical_cols]


# I print the number of columns for each type
print(f"Number of categorical columns: {len(categorical_cols)}")
print(f"Number of numerical columns: {len(numerical_cols)}")
```

```
"""Manual Scaling using Z formula = Xnew = (Xi-mean(X))/std(X)"""


def manual_standard_scaler(numerical_data):


    X_scaled_df = pd.DataFrame() #I will store my scaled data in this df


    for col in numerical_data.columns:
        mean = numerical_data[col].mean()
        std_dev = numerical_data[col].std()
        X_scaled_df[col] = (numerical_data[col] - mean) / std_dev # Here I am scaling each column indidually using the formula stated
above


    return X_scaled_df


X_scaled_df = manual_standard_scaler(numerical_data)
print(X_scaled_df) # I am checking output


"""Now have to deal wit categorical encoding"""


print(categorical_cols)


"""As I want to deal with different encoding techniques I need to split up the data"""


binary_column_names = ["schoolsup", "famsup", "paid", "activities", "nursery", "higher", "internet", "romantic"]
binary_column_names_other = ['school', 'sex', 'address', 'famsize', 'Pstatus']
cat_col_names = ["Mjob","Fjob","reason","guardian"]


binary_cols_yes_no = dfp[binary_column_names] #These are the columns which have Yes or No as values
binary_cols_other = dfp[binary_column_names_other] # These are columns that have non yes or no binary values eg GS or MP
(school)
cat_cols= dfp[cat_col_names] # These are the other categrical colunms


encoder = OneHotEncoder(sparse_output=False)  # I use sparse_output=False to get a dense array to try limit sparsity
```

```
X_encoded = encoder.fit_transform(cat_cols) # I use one hot encoding for my categorical columns

encoded_column_names = encoder.get_feature_names_out(cat_cols.columns)

X_encoded_df = pd.DataFrame(X_encoded, columns=encoded_column_names) # I convert the result back to a DataFrame


#now Binary

binary_cols_mapped = binary_cols_yes_no.replace({'yes': 1, 'no': 0}) #Binary mapping for my yes no columns

binary_mapping = {

    'school': {'GP': 1, 'MS': 0},

    'sex': {'F': 1, 'M': 0},

    'address': {'U': 1, 'R': 0}, #I create a mapping for non yes no columns

    'famsize': {'GT3': 1, 'LE3': 0},

    'Pstatus': {'T': 1, 'A': 0}

}


binary_cols_mapped_other = binary_cols_other.replace(binary_mapping)


# I combine my encoded binary columns

binary_cols_combined = pd.concat([binary_cols_mapped, binary_cols_mapped_other], axis=1)


cat_df = pd.concat([X_encoded_df, binary_cols_combined.reset_index(drop=True)], axis=1) # I combined all my encoded data into
one df again - used the pandas help guide for this

print("Final Combined DataFrame:")

print(cat_df)


"""I make sure shapes are correct"""


print("Shape of scaled numerical data:", X_scaled_df.shape)

print("Shape of categorical data:", cat_df.shape)


"""Combine into my final X_combined feature matrix"""


X_combined = pd.concat([

    X_scaled_df.reset_index(drop=True),
    cat_df.reset_index(drop=True)
], axis=1)


print("Shape of combined data:", X_combined.shape)
print("Combined DataFrame head:\n", X_combined)


"""## MultiLinearRegression as a class"""


m=len(y)
X_combined_MLR = X_combined #I assign this to a new variable so that theres no overlap in future code
X_combined_MLR.shape


y.shape


class Manual_MLR: # I originally used just features but found that to be too confsuing and messy - I instead created this class - used https://docs.python.org/3/tutorial/classes.html as a guide


    def __init__(self, learning_rate=0.001, epochs=2000): #Here I have to set up an initailisation of my variables
        self.learning_rate = learning_rate #the self allows me to access the methods and variables wuthin my class
        self.epochs = epochs
        self.weights = None
        self.intercept = None


    def predicted_y(self, x):
        return x.dot(self.weights) + self.intercept  #Here I am using the formula xw + b that is used in linear regression


    def manual_loss(self, y, y_predicted):
        return np.mean((y - y_predicted) ** 2)  # This is my manual mean squared error formula


    def partial_dw(self, x, y, y_predicted):
        return (-2 / len(y)) * x.T.dot((y - y_predicted))  # This formula is the partial derivite of the loss function above with regards to the weights. I calculated this manually
    def partial_db(self, y, y_predicted):
        return (-2 / len(y)) * np.sum(y - y_predicted)  # This formula is the partial derivite of the MSE above with regards to the intercept/bias. I also calculated this manually


    def manual_r2(self, y, y_predicted):
        y_mean = np.mean(y)
        ss_tot = np.sum((y - y_mean) ** 2)  # Total sum of squares
        ss_res = np.sum((y - y_predicted) ** 2)  # Residual sum of squares
        r2_score = 1 - (ss_res / ss_tot)  # This is the manual R^2 formula. This will help me examine how well my model captures the variance
        return r2_score


    def fit(self, x, y): # Here I perform my gradient descent
        np.random.seed(42) # So that my weighst dont keep changing
        self.weights = np.random.randn(x.shape[1]) #I initialise my random weights
        self.intercept = 0  # I initialise my random intercept
        linear_loss = [] # This will be used to help me plot the loss


        for _ in range(self.epochs): # I continue to update weights and bias over total number of epochs
            y_predicted = self.predicted_y(x) #Make a prediction using current weights and bias
            # Here I am updating the weights by multipying the learning rate by the partial derivite of the features matrix multiplied by the -2/len(m) multiplied by the differnce between y and its prediction y_hat
            self.weights -= self.learning_rate * self.partial_dw(x, y, y_predicted)
            self.intercept -= self.learning_rate * self.partial_db(y, y_predicted)
            # I store the loss
            linear_loss.append(self.manual_loss(y, y_predicted))


        # I plot loss over epochs
        plt.plot(np.arange(1, self.epochs), linear_loss[1:])
        plt.xlabel("Number of Epochs")
        plt.ylabel("Loss (MSE)")
        plt.title("Loss over Epochs")
        plt.show()
    def train_test_model(self, X_combined, y, test_size=0.2):

        # Check if inputs are pandas DataFrame or Series and convert to numpy

        if isinstance(X_combined, pd.DataFrame): # I was having some formatting issues and found this solution on stackoverflow

            X_combined = X_combined.values

        if isinstance(y, pd.Series):

            y = y.values

        # I split the data into 80/20 train test

        X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=test_size, random_state=42)

        # I train the model on training data using gradient descent through my fit() function

        self.fit(X_train, y_train)



        y_pred_test = self.predicted_y(X_test)

        test_loss = self.manual_loss(y_test, y_pred_test) # See how my model perfoms on the test set

        test_r2 = self.manual_r2(y_test, y_pred_test)

        print(f"Test Loss (MSE) on Unseen Data: {test_loss}")

        print(f"Test R^2 Score on Unseen Data: {test_r2}")



        # NOTE - I printed the MSE and R2 scores from sklearn to compare to my scores and ensure that my formulas were working
correctly

        mse = mean_squared_error(y_test, y_pred_test)

        print(f"Mean Squared Error (MSE): {mse}")

        r2 = r2_score(y_test, y_pred_test)

        print(f"R² Score: {r2}")



        return self.weights, self.intercept, y_pred_test, y_test



    def predict(self, inp):

        return self.predicted_y(inp)



MLR_model = Manual_MLR(learning_rate=0.001, epochs=2000) # use my class

weights, intercept, y_pred_test, y_test = MLR_model.train_test_model(X_combined_MLR, y) #Train and test the model on my actual
feature set
```

```
print("Weights:", weights)

print("Intercept (bias):", intercept)

y_pred = MLR_model.predict(X_combined_MLR)

print("Predicted y values on test set:", y_pred_test) # compare actual values

print("Actual values for the test set:", y_test)


"""Wanted to plot the actual vs predicted"""


df_pred_test = pd.DataFrame()

df_pred_test["y_actual"] = y_test

df_pred_test["y_predicted"] = np.round(y_pred_test, 1)  # I round predicted values to 1 decimal place

print(df_pred_test)


plt.figure(figsize=(10, 6))

plt.plot(range(len(y_test)), y_test, label='Actual G3', color='blue')  # Actual values (Test Set)

plt.plot(range(len(y_test)), y_pred_test, label='Predicted G3', color='red', linestyle='--')  # Predicted values (Test Set)


plt.title('Actual vs Predicted G3 Scores (Test Set)')

plt.xlabel('Sample Index')

plt.ylabel('G3 Score')

plt.legend()

plt.grid()

plt.show()


"""## Manual Neural Network"""


X_combined.shape


X_combined_NN = X_combined # again I dont want my orogianl feature matrix to be changed so store in a new variable


y_NN = y.values.reshape(-1, 1) # Had to do this to feed my values in
```

```
y_NN.shape


"""Sources used include notes, https://www.tutorialspoint.com/machine_learning/machine_learning_perceptron.htm and
https://medium.com/@waleedmousa975/

building-a-neural-network-from-scratch-using-numpy-and-math-libraries-a-step-by-step-tutorial-in-608090c20466

"""



class Manual_NN: # Same as above I made this into a class



    def __init__(self, X_combined_NN, y_NN, num_hidden=5, alpha=0.00001, num_iterations=500, test_size=0.2,val_size = 0.1,
random_seed=42):

        # Iniitalise my variables

        self.x = X_combined_NN  # My input feature matrix

        self.y = y_NN  # My actual values

        self.num_hidden = num_hidden  # Number of hidden layers

        self.alpha = alpha  # My learning rate

        self.num_iterations = num_iterations  # The number of epochs i should use

        self.test_size = test_size  # Size of test set

        self.val_size = val_size # My validation set size

        self.random_seed = random_seed # To avoid random changes



        # Initialise weights and biases. Use self to initailise variables in a class

        self.W1 = None

        self.b1 = None

        self.W2 = None

        self.b2 = None

        self.val_cost_history = []

        self.cost_history = []  # To store the cost during training

        self.final_train_cost = None

        self.final_test_cost = None

        self.final_test_r2 = None



        self.X_train, self.X_temp, self.y_train, self.y_temp = train_test_split(

            self.x, self.y, test_size=self.test_size + self.val_size, random_state=self.random_seed) #
        # Split my remaining data into validation and testing sets
        self.X_val, self.X_test, self.y_val, self.y_test = train_test_split(
            self.X_temp, self.y_temp, test_size=self.test_size / (self.test_size + self.val_size), random_state=self.random_seed)

        self.initialise_weights()

    def initialise_weights(self):
        np.random.seed(self.random_seed) #Setting random seed used above
        num_input = self.X_train.shape[1]
        num_output = 1  # (for regression, I want a single output)
        self.W1 = np.random.randn(num_input, self.num_hidden) * 0.001  # Smaller initial weights
        self.b1 = np.zeros((1, self.num_hidden))  # Bias for hidden layer
        self.W2 = np.random.randn(self.num_hidden, num_output) * 0.001  # Smaller initial weights
        self.b2 = np.zeros((1, num_output))  # Bias for output layer

    @staticmethod
    def relu(z): #Here I am using RelU instead of sigmoid as I was having issues with exploding gradients which sets negative values to
0 #Lecture 10 Notes
        return np.maximum(0, z) #Formula I took from notes
#
    @staticmethod
    def relu_derivative(z):
        return np.where(z > 0, 1, 0) #This is the derivite of RelU for backpropogation #https://medium.com/@Coursesteach/deep-
learning-part-25-derivatives-of-activation-functions-4bbd7c7c7a1c

    def forward_prop(self, x): # My fowrad learning phase
        z1 = np.dot(x, self.W1) + self.b1  # Linear combination from linear regression (input -> hidden) - sum of weighted input
        act1 = self.relu(z1)  # Apply my ReLU activation (hidden layer output)
        z2 = np.dot(act1, self.W2) + self.b2  # Linear combination (hidden -> output) to feed into output
        y_hat = z2  # For regression, I have no activation at the output layer (linear output) as I dont want values between 0 and 1
        return z1, act1, z2, y_hat
    @staticmethod # Included this method as found it here https://stackoverflow.com/questions/2438473/what-is-the-purpose-of-static-methods-how-do-i-know-when-to-use-one

    def cost_function(y, y_hat):

        m = len(y)

        J = np.mean((y - y_hat) ** 2)  # MSE formula again

        return J

#I reference the labs for this code along with https://medium.com/@waleedmousa975/building-a-neural-network-from-scratch-using-numpy-and-math-libraries-a-step-by-step-tutorial-in-608090c20466

    def backward_prop(self, y_hat, z1, act1, y):  # Backward propagation (Gradient computation) to update weights

        dCost_z2 = -(y - y_hat)  # Derivative of cost wrt z2 - Looks at teh error between my actual and preidted values and prepars them to be fed back



        # Gradients for W2 and b2 (hidden -> output layer)

        dCost_W2 = np.dot(act1.T, dCost_z2)  # Gradient with regards to W2 and B2

        dCost_b2 = np.sum(dCost_z2, axis=0, keepdims=True)  # Gradient wrt b2 (bias) NO ACTIVATION FUNCTION ON OUTPUT LAYER

        dz2_act1 = np.dot(dCost_z2, self.W2.T)  #Here I am backpropogating the error back through the layers

        dz1 = dz2_act1 * self.relu_derivative(z1)  # Now applying the DERIVATIVE of RElU



        dCost_W1 = np.dot(self.X_train.T, dz1)  # Gradient with regads to my W1

        dCost_b1 = np.sum(dz1, axis=0, keepdims=True)  # Gradient with regards to my bias



        return dCost_W1, dCost_b1, dCost_W2, dCost_b2



    def train(self): #Here is the actual training of my NN

        best_val_cost = float('inf')

        patience_counter = 0

        for i in range(self.num_iterations):

            z1, act1, z2, y_hat = self.forward_prop(self.X_train) # My forward propogation

            dCost_W1, dCost_b1, dCost_W2, dCost_b2 = self.backward_prop(y_hat, z1, act1, self.y_train) # My backward propagation to calculate gradient

            self.W1 -= self.alpha * dCost_W1

            self.b1 -= self.alpha * dCost_b1

            self.W2 -= self.alpha * dCost_W2  # Here I am updating the weights and biases using gradient descent

            self.b2 -= self.alpha * dCost_b2
```

```
      c = self.cost_function(self.y_train, y_hat) #i am tracking cost so that I can plot it
      self.cost_history.append(c)

      _, _, _, y_val_hat = self.forward_prop(self.X_val)
      val_cost = self.cost_function(self.y_val, y_val_hat)
      self.val_cost_history.append(val_cost) # Storing validation cost

      if val_cost < best_val_cost: # If validation loss improves
        best_val_cost = val_cost # I set this as the new best
        patience_counter = 0  # AND i reset patience counter
      else:
        patience_counter += 1  # If my validation doesnt improve I increase patince counter by 1

      # So I can see the cost
      if i % 10 == 0:
        print(f"Iteration {i}: Training Cost {c}, Validation Cost {val_cost}")

      # If after 10 iterations in a row validation has failed to imporve I stop it so that theres no overfitting
      if patience_counter >= 10:
        print(f"Early stopping on iteration {i} - validation cost has not improved for {10} iterations.")
        break

  def evaluate(self):
    self.final_train_cost = self.cost_function(self.y_train, self.forward_prop(self.X_train)[-1]) #This is my final MSE on the training data
    print(f"Final Mean Squared Error (MSE) on Training Set: {self.final_train_cost}")

    _, _, _, y_test_hat = self.forward_prop(self.X_test)
    self.final_test_cost = self.cost_function(self.y_test, y_test_hat) #Testing on test set
    self.final_test_r2 = r2_score(self.y_test, y_test_hat) # R^2 score
    print(f"Final Mean Squared Error (MSE) on Test Set: {self.final_test_cost}")
    print(f"Final R^2 on Test Set: {self.final_test_r2}")
```

```
        # I wwant to print first 10 actual vs predicted values on the test set
        print("\nFirst 10 Actual vs Predicted values on Test Set:")
        for i in range(10):
            print(f"Actual: {self.y_test[i][0]:.4f}, Predicted: {y_test_hat[i][0]:.4f}")

    def plot_cost(self):  # I am plotting the cost function to check for convergence
      num_iterations_completed = len(self.cost_history)
      iterations = range(num_iterations_completed)
      plt.plot(iterations, self.cost_history, label='Training Cost')
      plt.plot(iterations, self.val_cost_history, label='Validation Cost')
      plt.title('Cost Function')
      plt.xlabel('Training Iterations')
      plt.ylabel('Cost')
      plt.legend()
      plt.grid()
      plt.show()

nn = Manual_NN(X_combined_NN, y_NN) #using the class on my data
nn.train()
nn.plot_cost()
nn.evaluate()

_, _, _, y_test_hat = nn.forward_prop(nn.X_test)  # Get predictions from my nn trained model
df_pred_test = pd.DataFrame()
df_pred_test["y_actual"] = nn.y_test.flatten()
df_pred_test["y_predicted"] = np.round(y_test_hat.flatten(), 1)
print(df_pred_test)
plt.figure(figsize=(10, 6))
plt.plot(range(len(nn.y_test)), nn.y_test.flatten(), label='Actual G3', color='blue')  # Actual values (Test Set)
plt.plot(range(len(nn.y_test)), y_test_hat.flatten(), label='Predicted G3', color='red', linestyle='--')  # Predicted values (Test Set)

plt.title('Actual vs Predicted G3 Scores (Test Set)')
plt.xlabel('Sample Index')

plt.ylabel('G3 Score')

plt.legend()

plt.grid()

plt.show()


"""## SVD LR"""


X_combined_SVD_MLR = X_combined


#The notes and labs from Lecture 4 were used to help this code


def Manual_SVD2(X_combined_SVD_MLR, num_components, feature_names): #This is my function for performing my SVD

    U, s, Vh = np.linalg.svd(X_combined_SVD_MLR) #Here I am splitting the original matrix into my U, s and Vh matrices

    Sigma = zeros((X_combined_SVD_MLR.shape[0], X_combined_SVD_MLR.shape[1]))  # Then I create the Sigma matrix as zeros
first

    Sigma[:num_components, :num_components] = diag(s[:num_components]) #Then fill this with the diagonal values from the s
matrix

    Sigma_k = Sigma[:, :num_components]

    Vh_reduced = Vh[:num_components, :] #reduce my matrices to the size of number of components

    X_svd = U.dot(Sigma_k) #Here I perform the final step of the SVD by creating the output matrix by multiplying the sigma reduced
matrix by the U (row x row )matrix




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

    cumulative_variance_ratio = np.cumsum(explained_variance_ratio) #This is the formula for the expalined vatraince ratio taht i need
to plot

    total_variance_covered = np.sum(explained_variance_ratio[:num_components])

    plt.bar(range(1, num_components + 1), explained_variance_ratio[:num_components], align="center")

    plt.xlabel("Principal Component")

    plt.ylabel("Explained Variance Ratio")

    plt.title("Explained Variance Ratio per Principal Component (SVD)")

    plt.xticks(range(1, num_components + 1))  # I set x-ticks to match principal components

    plt.show()
    print(f"Total Variance Covered by the first {num_components} components: {total_variance_covered:.4f}") #Total variance shown as text


    # I plot cumulative explained variance
    plt.plot(range(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, marker='o')
    plt.axhline(y=0.75, color='red', linestyle='--', label='75% Explained Variance')
    plt.axhline(y=0.85, color='pink', linestyle='--', label='85% Explained Variance')
    plt.axhline(y=0.95, color='purple', linestyle='--', label='95% Explained Variance')
    plt.xlabel("Number of Components")
    plt.ylabel("Cumulative Explained Variance")
    plt.title("Cumulative Explained Variance")
    plt.grid(True)
    plt.show()


    # For futher analysis I wanted to examine the key features in each component
    for component_idx in range(min(num_components, 2)):  # I plot the first two components
        comp= Vh_reduced[component_idx]  # Loadings for this component
        idx = np.argsort(np.abs(comp))[::-1] #sort by index
        sorted_features = [feature_names[i] for i in idx]
        sorted_loadings = comp[idx]


        plt.figure(figsize=(10, 6))
        plt.barh(sorted_features[:10], sorted_loadings[:10], color=['lightblue' if l > 0 else 'pink' for l in sorted_loadings[:10]]) # I use lightblue and pink as they looked better than when I used blue and red
        plt.xlabel('Loading Value')
        plt.title(f'Feature Contributions to Component {component_idx + 1}')
        plt.gca().invert_yaxis()
        plt.grid(True)
        plt.show()


    return X_svd, U.shape, s.shape, Vh.shape
```

```
X_svd = Manual_SVD2(X_combined_MLR,10,X_combined_SVD_MLR.columns)

X_svd


# Initialise my MLR class used above

MLR_model = Manual_MLR(learning_rate=0.001, epochs=2000)


# Train my model on my SVD data and get predictions on the test set

weights, intercept, y_pred_test, y_test = MLR_model.train_test_model(X_svd[0], y)


# Print learned parameters

print("Optimal Weights:", weights)

print("Optimal Intercept (bias):", intercept)


# Print predictions for the test set

print("Predicted y values on test set:", y_pred_test)

print("Actual values for the test set:", y_test)


"""Same code as before so no need to comment"""


df_pred_test = pd.DataFrame()

df_pred_test["y_actual"] = y_test

df_pred_test["y_predicted"] = np.round(y_pred_test, 1)

print(df_pred_test)

plt.figure(figsize=(10, 6))

plt.plot(range(len(y_test)), y_test, label='Actual G3', color='blue')  # Actual values (Test Set)

plt.plot(range(len(y_test)), y_pred_test, label='Predicted G3', color='red', linestyle='--')  # Predicted values (Test Set)

plt.title('Actual vs Predicted G3 Scores (Test Set)')

plt.xlabel('Sample Index')

plt.ylabel('G3 Score')

plt.legend()

plt.grid()

plt.show()
```

```
"""## SVD NN"""


X_combined_SVD_NN = X_combined # AGAIN I dont want X_Combined to be changed so set a new variable


X_svd_NN = Manual_SVD2(X_combined_SVD_NN,10,X_combined_SVD_NN.columns)
print(X_svd_NN[0].shape)


"""This was just included to test if my SVD was generating the same componets as the built in method"""


#from sklearn.decomposition import TruncatedSVD


#svd = TruncatedSVD(n_components=10)
#svd.fit(X_combined)
#result = svd.transform(X_combined)
#print(result)


svd_nn = Manual_NN(X_svd_NN[0], y_NN)
svd_nn.train()
svd_nn.plot_cost()
svd_nn.evaluate()


"""## Cross Validation and hyperparameter tuning


This section is not coded as it follows the exact same process as the previous Manual_MLR class but I had to make a few changes to parameter and function names to make it work with the RandomizedSearchCV method from SkLearn: Used https://stackoverflow.com/questions/76799442/how-to-add-my-customized-class-to-sklearn-pipeline-properly
"""


class Manual_MLR(BaseEstimator, RegressorMixin):
    def __init__(self, learning_rate=0.001, epochs=2000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.intercept = None


    def predicted_y(self, x):
        return x.dot(self.weights) + self.intercept


    def manual_loss(self, y, y_predicted):
        return np.mean((y - y_predicted) ** 2)


    def partial_dw(self, x, y, y_predicted):
        return (-2 / len(y)) * x.T.dot((y - y_predicted))


    def partial_db(self, y, y_predicted):
        return (-2 / len(y)) * np.sum(y - y_predicted)


    def manual_r2(self, y, y_predicted):
        y_mean = np.mean(y)
        ss_tot = np.sum((y - y_mean) ** 2)
        ss_res = np.sum((y - y_predicted) ** 2)
        return 1 - (ss_res / ss_tot)


    def fit(self, x, y):
        if isinstance(x, pd.DataFrame):
            x= x.values
        if isinstance(y, pd.Series):
            y = y.values


        print(f"Shape of x: {x.shape}, Shape of y: {y.shape}")
        np.random.seed(42)
        self.weights = np.random.randn(x.shape[1])
        self.intercept = 0
        linear_loss = []
```

```
        for _ in range(self.epochs):
            y_predicted = self.predicted_y(x)
            self.weights -= self.learning_rate * self.partial_dw(x, y, y_predicted)
            self.intercept -= self.learning_rate * self.partial_db(y, y_predicted)
            linear_loss.append(self.manual_loss(y, y_predicted))

        plt.plot(np.arange(1, self.epochs), linear_loss[1:])
        plt.xlabel("Number of Epochs")
        plt.ylabel("Loss (MSE)")
        plt.title("Loss over Epochs")
        plt.show()

        return self

    def predict(self, inp):
        return self.predicted_y(inp)

    def score(self, X, y):
        y_pred = self.predict(X)
        return -self.manual_loss(y, y_pred)


    def train_test_model(self, X_combined, y, test_size=0.2):
        if isinstance(X_combined, pd.DataFrame):
            X_combined = X_combined.values
        if isinstance(y, pd.Series):
            y = y.values
        X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=test_size, random_state=42)
        self.fit(X_train, y_train)
        y_pred_test = self.predict(X_test)
        test_loss = self.manual_loss(y_test, y_pred_test)
        test_r2 = self.manual_r2(y_test, y_pred_test)

        print(f"Test Loss (MSE) on Unseen Data (manual): {test_loss:.4f}")
        print(f"Test R^2 Score on Unseen Data (manual): {test_r2:.4f}")
        mse = mean_squared_error(y_test, y_pred_test)
        r2 = r2_score(y_test, y_pred_test)

        print(f"Mean Squared Error (MSE) with sklearn: {mse:.4f}")
        print(f"R² Score with sklearn: {r2:.4f}")

        return self.weights, self.intercept, y_pred_test, y_test

MLR_model_cv = Manual_MLR(learning_rate=0.05, epochs=5000) #updated after running hyperparameter updates
weights, intercept, y_pred_test, y_test = MLR_model_cv.train_test_model(X_combined_MLR, y)

print("Weights:", weights)
print("Intercept (bias):", intercept)
y_pred = MLR_model_cv.predict(X_combined_MLR)
print("Predicted y values:", y_pred)
print("Predicted y values on test set:", y_pred_test)
print("Actual values for the test set:", y_test)

#Here I am defining the grid with a rang eof posssible values for each hyperparameter (Learning rate and epoch)
param_grid = {
    "learning_rate": [0.001, 0.005, 0.01,0.05,0.1]
}

#I amn using the build in KFold cross validation library from sklearn and setting k = 10 based on previous literature
kfold = KFold(n_splits=10)

# Here I am using the Random Grid package from sklearn and setting my class Manual_MLR as the fucntion to use as my estimator
random_search = RandomizedSearchCV(
    estimator=Manual_MLR(),
    param_distributions=param_grid, #Telling the grid to use the ranges I have specified above
    cv=kfold, #Telling the search to use my 10 fold cross validation
    scoring='neg_mean_squared_error', #I discovered during research that the scoring only takes Negative MSE because -
    n_jobs=-1,                    # - minimising MSE is the same as maximising the negative MSE
    n_iter=10,
    refit=True,
    random_state=42
)

#Fit this to my MLR feature matrix and outputs
random_search.fit(X_combined_MLR, y)

# Output the best model and hyperparameters which I can then sub back in
best_model = random_search.best_estimator_
print("Best model hyperparameters:", random_search.best_params_)
print("Best model:", best_model)

"""## Cross Validation NN

This section is not coded for teh same reason as above. It follows the exact same process as the previous Manual_NN class but I had to make a few changes to parameter and function names to make it work with the RandomizedSearchCV method from SkLearn: Used https://stackoverflow.com/questions/76799442/how-to-add-my-customized-class-to-sklearn-pipeline-properly
"""

class Manual_NN_cv(BaseEstimator, RegressorMixin):
    def __init__(self, num_hidden=5, alpha=0.00001, num_iterations=500, test_size=0.2, random_seed=42):
        self.num_hidden = num_hidden
        self.alpha = alpha
        self.num_iterations = num_iterations
        self.test_size = test_size
        self.random_seed = random_seed
        self.cost_history = []
        self.final_train_cost = None
        self.final_test_cost = None
        self.final_test_r2 = None

    def initialize_weights(self, num_input, num_output):
        np.random.seed(self.random_seed)
        self.W1 = np.random.randn(num_input, self.num_hidden) * 0.001
        self.b1 = np.zeros((1, self.num_hidden))
        self.W2 = np.random.randn(self.num_hidden, num_output) * 0.001
        self.b2 = np.zeros((1, num_output))

    @staticmethod
    def relu(z):
        return np.maximum(0, z)

    @staticmethod
    def relu_derivative(z):
        return np.where(z > 0, 1, 0)

    @staticmethod
    def cost_function(y, y_hat):
        return np.mean((y - y_hat) ** 2)

    def forward_prop(self, x):
        z1 = np.dot(x, self.W1) + self.b1
        act1 = self.relu(z1)
        z2 = np.dot(act1, self.W2) + self.b2
        y_hat = z2
        return z1, act1, z2, y_hat

    def backward_prop(self, y_hat, z1, act1, y):
        dCost_z2 = -(y - y_hat)
        dCost_W2 = np.dot(act1.T, dCost_z2)
        dCost_b2 = np.sum(dCost_z2, axis=0, keepdims=True)
```

```
        dz2_act1 = np.dot(dCost_z2, self.W2.T)
        dz1 = dz2_act1 * self.relu_derivative(z1)
        dCost_W1 = np.dot(self.X_train.T, dz1)
        dCost_b1 = np.sum(dz1, axis=0, keepdims=True)

        return dCost_W1, dCost_b1, dCost_W2, dCost_b2

    def fit(self, X, y):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=self.test_size, random_state=self.random_seed)
        num_input = self.X_train.shape[1]
        num_output = 1
        self.initialize_weights(num_input, num_output)
        for i in range(self.num_iterations):
            z1, act1, z2, y_hat = self.forward_prop(self.X_train)
            dCost_W1, dCost_b1, dCost_W2, dCost_b2 = self.backward_prop(y_hat, z1, act1, self.y_train)
            self.W1 -= self.alpha * dCost_W1
            self.b1 -= self.alpha * dCost_b1
            self.W2 -= self.alpha * dCost_W2
            self.b2 -= self.alpha * dCost_b2
            cost = self.cost_function(self.y_train, y_hat)
            self.cost_history.append(cost)

        return self

    def predict(self, X):
        _, _, _, y_hat = self.forward_prop(X)
        return y_hat

    def score(self, X, y):
        y_pred = self.predict(X)
        return r2_score(y, y_pred)
    def evaluate(self):
        y_train_pred = self.predict(self.X_train)
        y_test_pred = self.predict(self.X_test)
        self.final_train_cost = self.cost_function(self.y_train, y_train_pred)
        self.final_test_cost = self.cost_function(self.y_test, y_test_pred)
        self.final_test_r2 = r2_score(self.y_test, y_test_pred)
        print(f"Final MSE on Training Set: {self.final_train_cost:.4f}")
        print(f"Final MSE on Test Set: {self.final_test_cost:.4f}")
        print(f"Final R² on Test Set: {self.final_test_r2:.4f}")

    def plot_cost(self):
        plt.plot(range(self.num_iterations), self.cost_history)
        plt.title('Cost Function over Iterations')
        plt.xlabel('Training Iterations')
        plt.ylabel('Cost (MSE)')
        plt.grid()
        plt.show()

nn = Manual_NN_cv(num_hidden=10, alpha=0.0001)
nn.fit(X_combined_NN, y_NN)
nn.plot_cost()
nn.evaluate()

"""Same Code as previous just different parameter grid"""

param_grid_nn = {
    "alpha": [0.00001, 0.0001, 0.001, 0.01, 0.1],
    "num_hidden": [5, 10, 20]
}
kfold_nn = KFold(n_splits=10)
random_search_nn = RandomizedSearchCV(
    estimator=Manual_NN_cv(),
    param_distributions=param_grid_nn,
    cv=kfold_nn,

    scoring='neg_mean_squared_error',

    n_jobs=-1,

    n_iter=10,

    refit=True,

    random_state=42

)


random_search_nn.fit(X_combined_NN, y_NN)

best_nn_model = random_search_nn.best_estimator_

print("Best model hyperparameters:", random_search_nn.best_params_)

print("Best model:", best_nn_model)


"""## Optimal Model


For this model I decided to use a mix of both SVD and feature selection based on my previous analysis

"""


X_combined.shape #Just a sanity check for me


columns_to_drop = ["age","famsup",'Fjob_at_home', 'Fjob_health', 'Fjob_other',

      'Fjob_services', 'Fjob_teacher',"famrel",'absences','address','paid',

      'nursery', 'internet'] #I dropped these features based on all the previous analysis done in this notebook both using probabilistic tests and also plotting their distributions


X_optimal= X_combined.drop(columns=columns_to_drop)

print(X_optimal.shape)


X_optimal_svd_NN = Manual_SVD2(X_optimal,8,X_optimal.columns)

X_optimal_svd_NN_input = X_optimal_svd_NN[0]

print(X_optimal_svd_NN_input.shape)


opt_nn = Manual_NN(X_optimal_svd_NN_input, y_NN,num_hidden=5, alpha=0.0001, num_iterations=150)
```

```
opt_nn.train()

opt_nn.plot_cost()

opt_nn.evaluate()


_, _, _, y_test_hat = opt_nn.forward_prop(opt_nn.X_test)  # Get predictions from my nn trained model

df_pred_test = pd.DataFrame()

df_pred_test["y_actual"] = opt_nn.y_test.flatten()

df_pred_test["y_predicted"] = np.round(y_test_hat.flatten(), 1)

print(df_pred_test)

plt.figure(figsize=(10, 6))

plt.plot(range(len(opt_nn.y_test)), opt_nn.y_test.flatten(), label='Actual G3', color='blue')  # Actual values (Test Set)

plt.plot(range(len(opt_nn.y_test)), y_test_hat.flatten(), label='Predicted G3', color='red', linestyle='--')  # Predicted values (Test Set)


plt.title('Actual vs Predicted G3 Scores (Test Set)')

plt.xlabel('Sample Index')

plt.ylabel('G3 Score')

plt.legend()

plt.grid()

plt.show()


"""## Markov Decision Process


For my MDP I decided to take studytime and DALC as my two featues to focus on and create actions for them. I had no experince in this so used https://github.com/SS-YS/MDP-with-Value-Iteration-and-Policy-Iteration/blob/main/policyIteration.py and https://medium.com/@ngao7/markov-decision-process-policy-iteration-42d35ee87c82 to help

"""


# These are my four states

states = ['S1', 'S2', 'S3', 'S4']

state_mapping = {  #Here I am mapping the states to their meaning

    'S1': 'Failing',

    'S2': 'Barely Passing',

    'S3': 'Performing Well',

    'S4': 'Excellent'
```

```
}


actions = {
    'A1': 'Increase Study Time', #My two actions
    'A2': 'Decrease Daily Alcohol Consumption'
}


# These are the transiotion probs I created based on knowledeg and trends I learned from analysing them
transition_probs = {
    'A1': {
        'S1': [0.20, 0.70, 0.10, 0.00],  # Likely to have fast positive effect
        'S2': [0.05, 0.40, 0.45, 0.10],  # Slows down but still positive
        'S3': [0.01, 0.20, 0.60, 0.19],  # Getting harder to move up the grades - EDA doesnt show strong enough correlation to indicate major increase
        'S4': [0.00, 0.05, 0.15, 0.8]   # Likely to stay - possibility of fatigue if too many hours leading to decrease but small probability
    },
    'A2': {
        'S1': [0.15, 0.75, 0.10, 0.00],  # As shown in EDA very likely fast increase
        'S2': [0.10, 0.35, 0.50, 0.05],  # Again likely to increase
        'S3': [0.05, 0.30, 0.50, 0.10],  # Much smaller chance to increase as correlation in EDA doesnt show high increase in high results when DALC dropped
        'S4': [0.00, 0.00, 0.35, 0.65]   # Could drop due to losing out on social interaction but likely to stay
    }
}


# Again here I come up with releavnt rewards for each transotion between states
rewards = {
    'A1': {  # Increase Study Time
        ('S1', 'S1'): -.5,
        ('S1', 'S2'): 1,
        ('S1', 'S3'): 3,
        ('S1', 'S4'): 6,
        ('S2', 'S1'): -2,
        ('S2', 'S2'): 1,
        ('S2', 'S3'): 3,
        ('S2', 'S4'): 8,
        ('S3', 'S1'): -3,
        ('S3', 'S2'): -2,
        ('S3', 'S3'): 2,
        ('S3', 'S4'): 4,
        ('S4', 'S1'): -8,
        ('S4', 'S2'): -3,
        ('S4', 'S3'): 1,
        ('S4', 'S4'): 4
    },
    'A2': {  # Decrease Alcohol Consumption
        ('S1', 'S1'): -1,
        ('S1', 'S2'): 2,
        ('S1', 'S3'): 4,
        ('S1', 'S4'): 8,
        ('S2', 'S1'): -1,
        ('S2', 'S2'): 2,
        ('S2', 'S3'): 4,
        ('S2', 'S4'): 9,
        ('S3', 'S1'): -2,
        ('S3', 'S2'): -1,
        ('S3', 'S3'): 1,
        ('S3', 'S4'): 2,
        ('S4', 'S1'): -8,
        ('S4', 'S2'): -6,
        ('S4', 'S3'): -2,
        ('S4', 'S4'): 6
    }
}
# I am using a policy iteration method to optimise my decision-making process.
policy = {state: 'A1' for state in states}  # I start with action 'A1' for every state.
value_function = {state: 0 for state in states}  # I initialise the value function to 0 for all states.
```

```
# Policy Evaluation Function
def policy_eval(policy, value_function, transition_probabilities, rewards, gamma=0.9, theta=0.0001):
    while True:  # I will keep evaluating the policy until it converges.
        delta = 0  # I set delta to track the maximum change in value function.
        for state in states:  # I loop through each state to update its value.
            v = value_function[state]  # I save the current value of the state.
            action = policy[state]  # I find out what action is taken for this state.

            # I calculate the new value for the state based on the expected rewards and discounted future values.
            value_function[state] = sum(
                prob * (rewards[action].get((state, next_state), 0) + gamma * value_function[next_state])
                for next_state, prob in zip(states, transition_probabilities[action][state])
            )
            # I update delta to track the largest change I made during this iteration.
            delta = max(delta, abs(v - value_function[state]))

        # If the maximum change is less than the threshold, I stop evaluating.
        if delta < theta:
            break

# Policy Improvement Function
def policy_improvement(value_function, policy, transition_probabilities, rewards, gamma=0.9):
    policy_stable = True  # I start by assuming the policy is stable.
    for state in states:  # I loop through each state to improve the policy.
        old_action = policy[state]  # I save the action that was previously taken for this state.

        # I choose the action that maximises expected utility for the current state.
        policy[state] = max(actions.keys(),
                            key=lambda action: sum(prob * (rewards[action].get((state, next_state), 0) + gamma * value_function[next_state])
                                                   for next_state, prob in zip(states, transition_probabilities[action][state])))

        # If the action for the state has changed, the policy is no longer stable.
        if old_action != policy[state]:
            policy_stable = False

    return policy_stable  # I return whether the policy is stable or not.


# Policy Iteration Process
is_policy_stable = False  # I begin with an unstable policy.
while not is_policy_stable:  # I will iterate until the policy becomes stable.
    policy_eval(policy, value_function, transition_probs, rewards)  # I evaluate the current policy.
    is_policy_stable = policy_improvement(value_function, policy, transition_probs, rewards)  # I improve the policy.


# Final Policy and Value Function
print("Optimal Policy:")  # I output the optimal policy.
for state in states:  # I loop through each state to print its optimal action.
    print(f"{state}: {actions[policy[state]]}")  # I show the best action for each state.


print("\nOptimal Value Function:")  # I output the optimal value function.
for state in states:  # I loop through each state to print its value.
    print(f"{state}: {value_function[state]:.2f}")  # I display the value for each state, formatted to two decimal places.
```