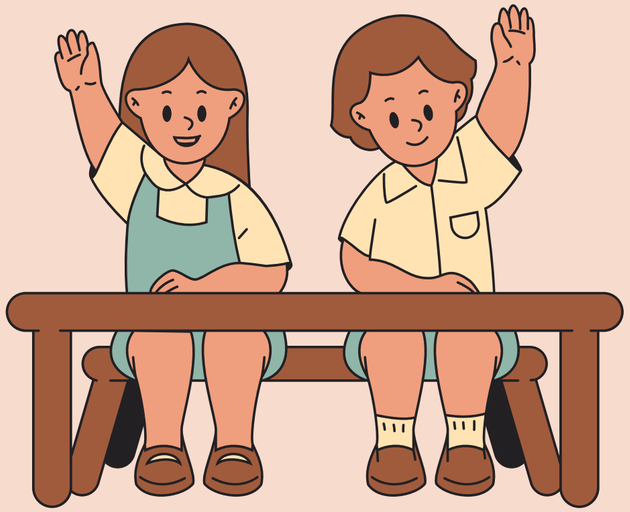
Bhavan's Vivekananda College
Sainikpuri

ANALYSIS ON STUDENT PERFORMANCE DATASET



PRESENTED BY :

- VANI SUMEDHA
- N. HARIKA
- PAKKI VENKATA TEJA

Bhavan's Vivekananda College | B.Sc Hons Data Science

CONTENTS

- INTRODUCTION
- OBJECTIVE
- CLEANING OF DATA
- EDA
- ANALYSIS
- CONCLUSION

DATA DESCRIPTION

- The data was collected by using school reports and questionnaires which is regarding the performance in one subject : Mathematics.
- The Target attribute G3 (final year marks) has a strong correlation with attributes G2 (2nd year marks) and G1(1st year marks).

OBJECTIVE

Application of different machine learning (ML) algorithms to find the best model in predicting the student's performance in secondary education.

ABOUT THE DATA

The dataset consists of 32 feature columns and 1 target column.
G3 is the target variable.

Numerical columns

age
absences
G1
G2
G3

Categorical columns

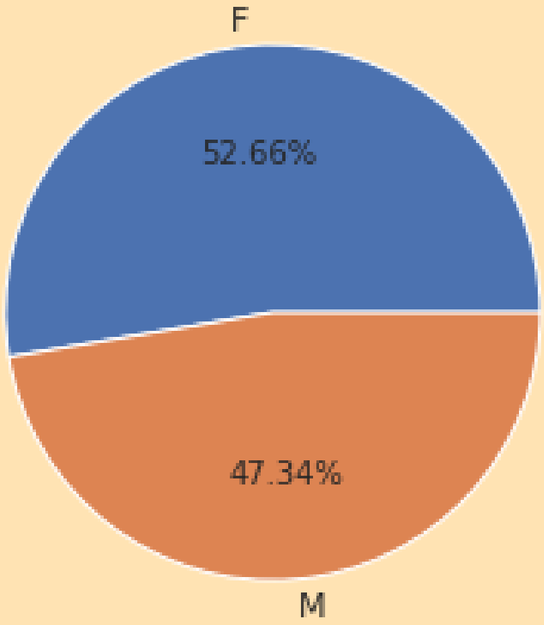
Medu	famrel	Pstatus
Fedu	freetime	schoolsup
Mjob	goout	famsup
Fjob	Dalc	paid
reason	Walc	activities
guardian	health	nursery
tarveltime	school	higher
studytime	sex	internet
failures	address	romantic
	famsize	

*refer to appendix 1

```
school      0
sex         0
age         0
address     0
famsize     0
Pstatus     0
Medu        0
Fedu        0
Mjob        0
Fjob        0
reason      0
guardian    0
traveltime  0
studytime   0
failures    0
schoolsup   0
famsup      0
paid        0
activities  0
nursery     0
higher      0
internet    0
romantic    0
famrel      0
freetime    0
goout       0
Dalc        0
Walc        0
health      0
absences    0
G1          0
G2          0
G3          0
dtype: int64
```

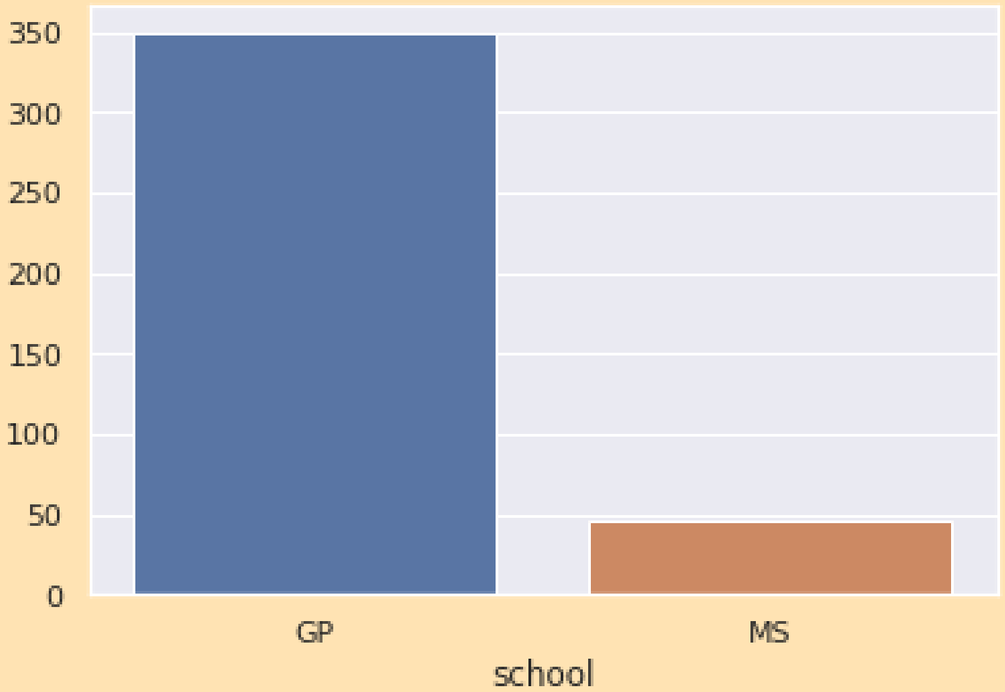
There are no missing values in the given data.

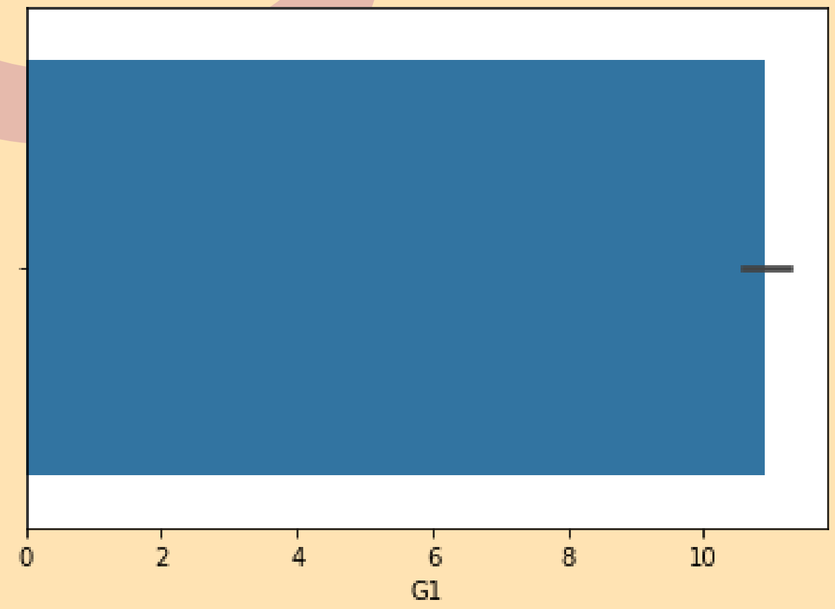
EXPLORATORY DATA ANALYSIS



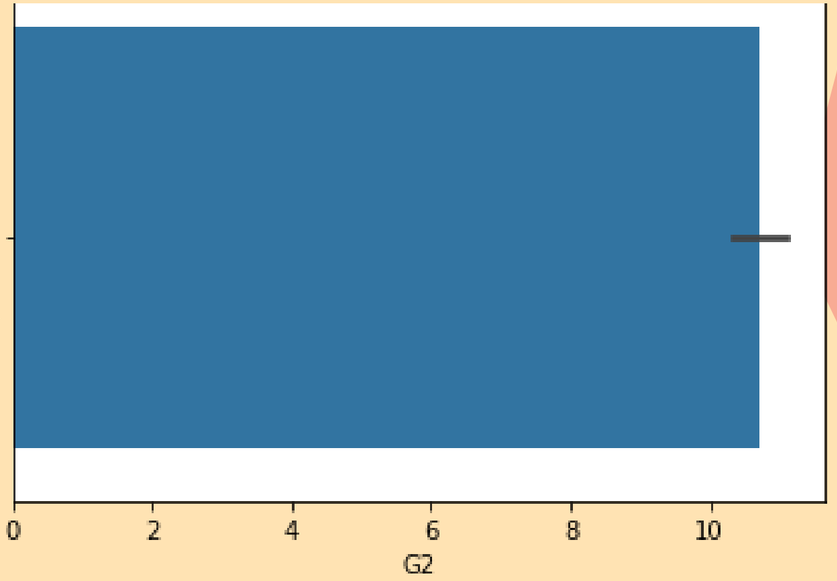
Depicts the no. of male and female students

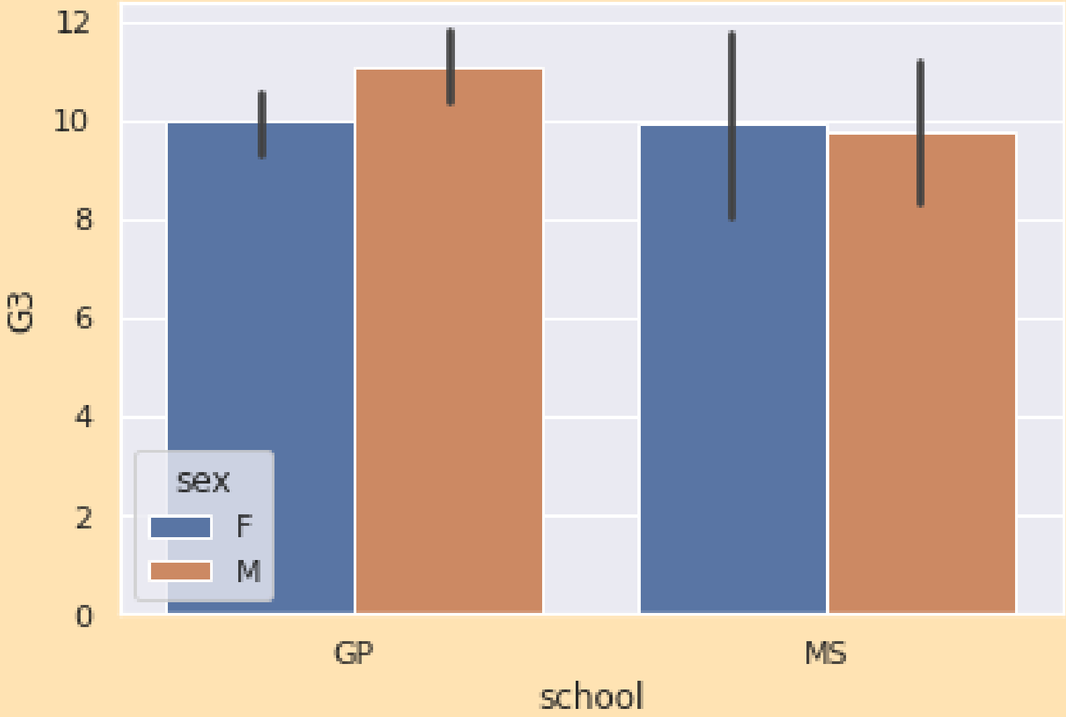
Shows the comparison of student's responses of two different schools.



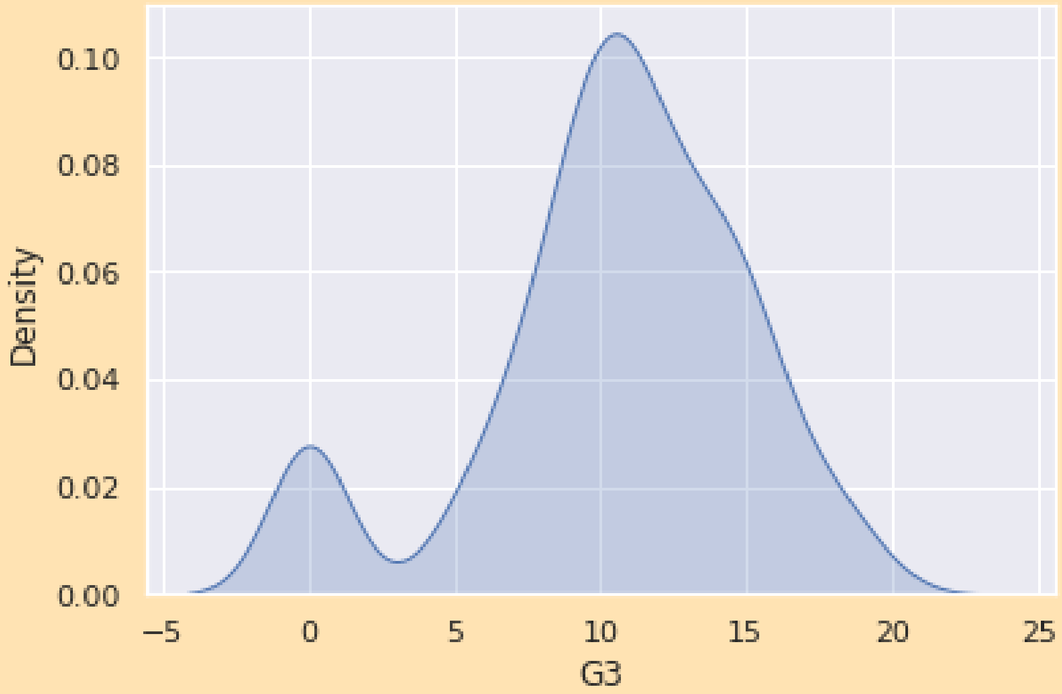


Mean of G1 = 10.90
Mean of G2 = 10.71



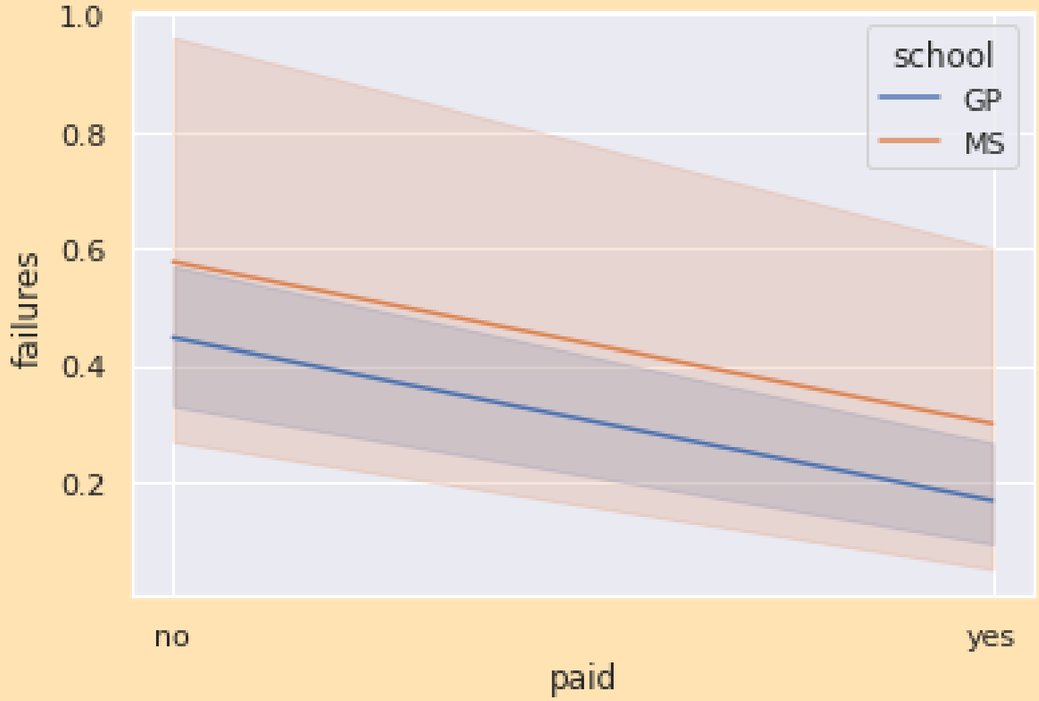


Comparison of two schools to analyse the the marks of the students



The most frequent marks scored by the students

The no. of the failures in two schools



LINEAR REGRESSION

Without considering G1 & G2 variables

Train/test split	MAE
70:30	2.8896
75:25	3.4607
80:20	3.3421
60:40	3.7905

→ The best mae is for 70:30 ratio.

*refer to appendix 2, 3 & 4

With considering G1 & G2 variables

x = independent attributes

y = it contains the target attribute i.e., G3

Train/test split	MAE
70:30	1.1616
75:25	1.1895
80:20	1.1724
60:40	1.3351

In conclusion, we get the best mae with 70:30 ratio for this dataset.

*refer to appendix 2, 3 & 4

NEURAL NETWORKS

optimizer	train test	layers	epochs	mae
adam	75 25	10--8--1	20	1.3165
			50	1.1505
			100	1.0732
			200	1.1354
			300	1.0561
			500	1.0637
adam	75 25	13--7--6--1	20	1.3029
			50	1.1331
			100	1.0771
			200	1.1309
			300	1.0702
			500	1.0576
SGD	75 25	13--7--6--1	20	4.1624
			50	3.2404
			100	1.6311
			200	2.7392
			300	2.4971

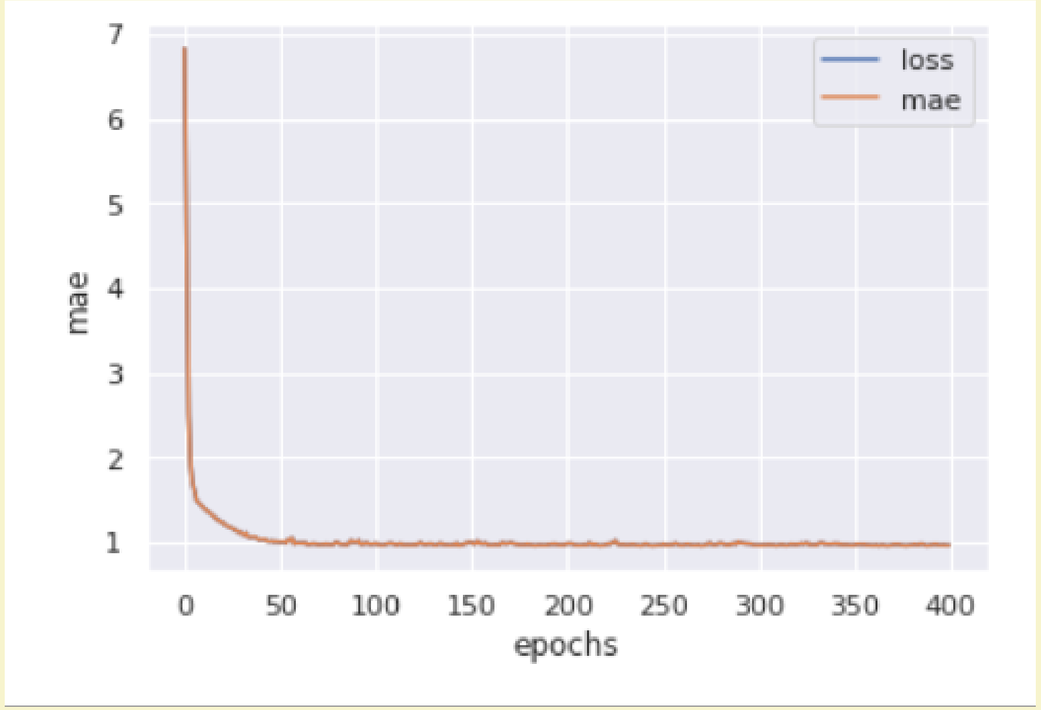
ratio	opt	layer	epochs	mae
70-30	adam	10-8-1'	20	1.143
	adam	10-8 1	50	1.0765
	adam	10-8-1	100	1.0488
	adam	10-8-1	200	1.05
	adam	10-8-1	300	1.0581
	adam	10-8-1	500	1.0565
ratio	opt	layer	epochs	mae
70-30	SGD	13-7-6-1	20	1.3155
	SGD		50	1.0816
	SGD		100	1.0519
	SGD		200	1.0513
	SGD		300	1.0829
	SGD		500	1.0535

*refer to appendix 5, 6 & 7

ratio	layers	opt	epochs	mae					opt	mae
80-20	13-7-6-1	Adam	20	1.284					SGD	4.2936
80-20	13-7-6-1	Adam	50	1.164					SGD	4.7012
80-20	13-7-6-1	Adam	100	1.0295					SGD	1.9303
80-20	13-7-6-1	Adam	200	1.0158					SGD	2.6095
80-20	13-7-6-1	Adam	300	1.0204					SGD	1.2337
80-20	13-7-6-1	Adam	500	1.0192					SGD	1.2276
80-20	10--8--1	Adam	20	1.1506					SGD	1.5219
80-20	10--8--1	Adam	50	1.0671					SGD	2.9474
80-20	10--8--1	Adam	100	1.0209					SGD	1.4314
80-20	10--8--1	Adam	200	1.0264					SGD	2.2895
80-20	10--8--1	Adam	300	1.0259					SGD	1.1735
80-20	10--8--1	Adam	500	1.0169					SGD	2.0398
80-20	10-7-6-5-3-1	Adam	20	1.356					SGD	2.1297
80-20	10-7-6-5-3-1	Adam	50	1.1603					SGD	2.1727
80-20	10-7-6-5-3-1	Adam	100	1.1915					SGD	1.3289
80-20	10-7-6-5-3-1	Adam	200	1.0285					SGD	1.8006
80-20	10-7-6-5-3-1	Adam	300	1.0356					SGD	2.2135
80-20	10-7-6-5-3-1	Adam	500	1.0226					SGD	1.3986
80-20	8--1	Adam	20	1.6705					SGD	1.2434
80-20	8--1	Adam	50	1.2741					SGD	3.254
80-20	8--1	Adam	100	1.1965					SGD	1.3534
80-20	8--1	Adam	200	1.0222					SGD	1.2784
80-20	8--1	Adam	300	1.0192					SGD	1.3154
80-20	8--1	Adam	500	1.0231					SGD	1.4959

- We found the maximum MAE value when SGD was the optimizer.
- The range of the MAE values are between 1 .0158 - 1.6705, when Adam is the optimizer.

*refer to appendix 5, 6 & 7

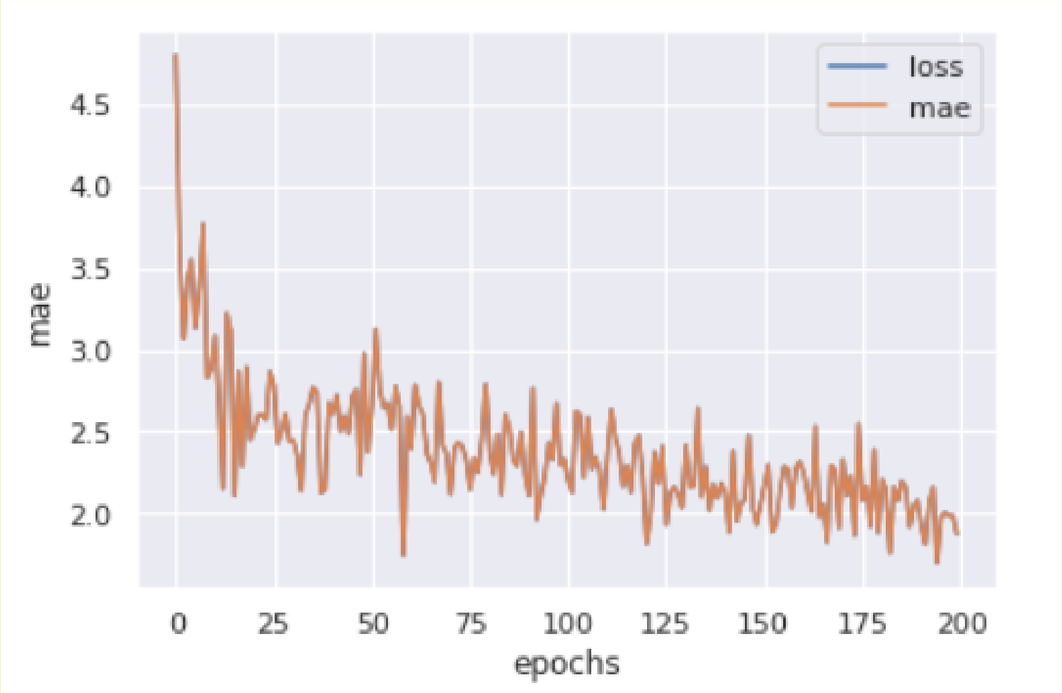


- ***MAE Value : 1.0158***

Epochs vs MAE graph when Adam is the optimizer

- ***MAE Value : 1.0513***

Epochs vs MAE graph when SGD is the optimizer



*refer to appendix 5, 6 & 7

INSIGHTS

The features that makes the neural network model the best :

- Adam is the best optimizer [gives us the better results than SGD]
- The architecture 13-7-6-1 is the optimum layer.
- Train and test split ratio 80 : 20.
- Epochs = 200
- The best MAE is 1.0158

*refer to appendix 5, 6 & 7

BAGGING

Train / test split	MAE
60:40	1.0729
70:30	1.0879
75:25	1.0867
80:20	1.2254

*refer to appendix 8

BOOSTING

ADAPTIVE BOOSTING :

Train/Test Split	MAE
70:30	1.2646
75:25	1.2596
80:20	1.3081
60:40	1.1387

*refer to appendix 9

GRADIENT BOOSTING

Train/Test Split	MAE
70:30	1.5240
75:25	1.4429
80:20	1.5915
60:40	1.4627

XGBOOSTING

Train/Test Split	MAE
70:30	1.2762
75:25	1.1456
80:20	1.2762
60:40	1.1314

*refer to appendix 10 & 11

COMPARISON OF THE MODELS

TRAIN/TEST SPLIT RATIO - 80 : 20

Linear Regression	1.1724
Neural Networks	1.0158
Bagging	1.2254
Adaptive Boosting	1.3081
Gradient Boosting	1.5915
XGBoosting	1.2762

CONCLUSION

- After application of different ML algorithms to the dataset, the best mae value is given by Neural Networks.
- In conclusion Neural Networks is the best fit for the dataset.

Thank you!

Colab Link :
https://colab.research.google.com/drive/1v3GRShcqm88cPFauwJD4HsgoTlZjHsQW?usp=sharing

APPENDIX

ATTRIBUTE INFORMATION

1 school - student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
2 sex - student's sex (binary: 'F' - female or 'M' - male)
3 age - student's age (numeric: from 15 to 22)
4 address - student's home address type (binary: 'U' - urban or 'R' - rural)
5 famsize - family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
6 Pstatus - parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
7 Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
8 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
9 Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
10 Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
11 reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')
12 guardian - student's guardian (nominal: 'mother', 'father' or 'other')
13 traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15 failures - number of past class failures (numeric: n if 1<=n<3, else 4)
16 schoolsup - extra educational support (binary: yes or no)
17 famsup - family educational support (binary: yes or no)
18 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
19 activities - extra-curricular activities (binary: yes or no)
20 nursery - attended nursery school (binary: yes or no)
21 higher - wants to take higher education (binary: yes or no)
22 internet - Internet access at home (binary: yes or no)
23 romantic - with a romantic relationship (binary: yes or no)
24 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
25 freetime - free time after school (numeric: from 1 - very low to 5 - very high)
26 goout - going out with friends (numeric: from 1 - very low to 5 - very high)
27 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
28 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
29 health - current health status (numeric: from 1 - very bad to 5 - very good)
30 absences - number of school absences (numeric: from 0 to 93)

these grades are related with the course subject, Math or Portuguese:
31 G1 - first period grade (numeric: from 0 to 20)
31 G2 - second period grade (numeric: from 0 to 20)
32 G3 - final grade (numeric: from 0 to 20, output target)

PREDICTIVE ANALYSIS

```
lm3 = smf.ols(formula='G3 ~ Sch_int+gender+add_int+reason_int+studytime+failures+schoolsup_int+paid_int+act_int+health+absences+G
lm3.params
```

```
Intercept       -1.892788
Sch_int          0.065265
gender           0.248612
add_int          0.027831
reason_int       0.123367
studytime       -0.120276
failures        -0.269688
schoolsup_int   -0.605915
paid_int        -0.154515
act_int          0.324360
health           0.088296
absences         0.039895
G1               0.176712
G2               0.964827
dtype: float64
```

```
[5] lm3.summary()
```

OLS Regression Results

Dep. Variable:	G3	**R-squared:**	0.833
Model:	OLS	**Adj. R-squared:**	0.828
Method:	Least Squares	**F-statistic:**	146.6
Date:	Mon, 22 Aug 2022	**Prob (F-statistic):**	2.77e-139
Time:	10:26:38	**Log-Likelihood:**	-807.29
No. Observations:	395	**AIC:**	1643.
Df Residuals:	381	**BIC:**	1698.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.8928	0.598	-3.167	0.002	-3.068	-0.718
Sch_int	0.0653	0.322	0.203	0.839	-0.568	0.698
gender	0.2486	0.211	1.179	0.239	-0.166	0.663
add_int	0.0278	0.246	0.113	0.910	-0.455	0.511
reason_int	0.1234	0.102	1.207	0.228	-0.078	0.324
studytime	-0.1203	0.126	-0.951	0.342	-0.369	0.128
failures	-0.2697	0.143	-1.890	0.059	-0.550	0.011
schoolsup_int	-0.6059	0.302	-2.009	0.045	-1.199	-0.013
paid_int	-0.1545	0.201	-0.768	0.443	-0.550	0.241
act_int	0.3244	0.196	1.652	0.099	-0.062	0.710
health	0.0883	0.071	1.252	0.211	-0.050	0.227
absences	0.0399	0.012	3.251	0.001	0.016	0.064
G1	0.1767	0.058	3.057	0.002	0.063	0.290
G2	0.9648	0.050	19.206	0.000	0.866	1.064

Omnibus:	202.797	**Durbin-Watson:**	1.880
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1098.945
Skew:	-2.212	**Prob(JB):**	2.33e-239
Kurtosis:	9.871	**Cond. No.**	114.

Fitting the linear regression to the complete data and the obtained r-square value is 0.833.

```
X = data[["Sch_int","gender","add_int","reason_int","studytime","failures","schoolsup_int","paid_int","act_int","health","absences"]]
y = data.G3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
lm3 = LinearRegression()
lm3.fit(X_train, y_train)
y_pred = lm3.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
3.8310262023511332
```

```
mae = metrics.mean_absolute_error(y_test, y_pred)  ##MAE (Mean absolute error)
mse = metrics.mean_squared_error(y_test, y_pred)   ##MSE (Mean Squared Error)
rmse = np.sqrt(mse) # or mse**(0.5)                ##RMSE (Root Mean Squared Error)
r2 = metrics.r2_score(y_test, y_pred)              ##R-squared (Coefficient of determination)
print(mae)
print(mse)
print(rmse)
print(r2)
print(100-(1.96*rmse))
```

```
2.936829139590375
14.676761763100945
3.8310262023511332
0.16431060736434566
92.49118864339178
```

Removing the G1 and G2 variables, the mean square error (mse) value has increased and the r2 value has dropped.

After training and testing the model, linear regression is fit, considering G1 & G2 variables.

```
X = data[["Sch_int","gender","add_int","reason_int","studytime","failures","schoolsup_int","paid_int","act_int","health","absences","G1","G2"]]
y = data.G3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
lm3 = LinearRegression()
lm3.fit(X_train, y_train)
y_pred = lm3.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
1.6384051763809078
```

```
[95] mae = metrics.mean_absolute_error(y_test, y_pred)  ##MAE (Mean absolute error)
     mse = metrics.mean_squared_error(y_test, y_pred)   ##MSE (Mean Squared Error)
     rmse = np.sqrt(mse) # or mse**(0.5)                ##RMSE (Root Mean Squared Error)
     r2 = metrics.r2_score(y_test, y_pred)              ##R-squared (Coefficient of determination)
     print(mae)
     print(mse)
     print(rmse)
     print(r2)
     print(100-(1.96*rmse))
```

```
1.1789113518185959
2.6843715219917534
1.6384051763809078
0.8691116321440738
96.78872585429342
```

NEURAL NETWORKS

```
x= a.drop("G3", axis=1)
y=a["G3"]
x.head
```

```
<bound method NDFrame.head of      G2  gender  paid_int  absences  schoolsup_int  failures  health  act_int  \
0     6       0         1         6              0         0       3        1
1     5       0         1         4              1         0       3        1
2     8       0         0        10              0         3       3        1
3    14       0         0         2              1         0       5        0
4    10       0         0         4              1         0       5        1
..   ..     ...       ...       ...            ...       ...     ...      ...
390   9       1         0        11              1         2       4        1
391  16       1         1         3              1         0       2        1
392   8       1         1         3              1         3       3        1
393  12       1         1         0              1         0       5        1
394   9       1         1         5              1         0       5        1

     G1  add_int  reason_int  studytime  Sch_int
0     5        0           2          2        0
1     5        0           2          2        0
2     7        0           3          2        0
3    15        0           0          3        0
4     6        0           0          2        0
..   ..      ...         ...        ...      ...
390   9        0           2          2        1
391  14        0           2          1        1
392  10        1           2          1        1
393  11        1           2          1        1
394   8        0           2          1        1

[395 rows x 13 columns]>
```

• OPTIMIZER : ADAM

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.20, random_state=42)
y_test
```

```
tf.random.set_seed(42)

model = tf.keras.Sequential([
                              tf.keras.layers.Dense(13),
                              tf.keras.layers.Dense(7),
                              tf.keras.layers.Dense(6),
                              tf.keras.layers.Dense(1)
])

model.compile(loss= tf.keras.losses.mae,
              optimizer= tf.keras.optimizers.Adam(),
              metrics= ['mae'])

# STEP1: Fit the model

history = model.fit(x_train, y_train, epochs=200, verbose=0)
```

```
model.evaluate(x_test, y_test)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 1.0158 - mae: 1.0158
[1.015805721282959, 1.015805721282959]
```

By trying different train and test splits, we chose the best model in the ratio of 80:20.

• OPTIMIZER : SGD

```
[99] from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.30, random_state=42)
     y_test
```

```
[108] tf.random.set_seed(42)

      # STEP1: Creating the model

      model= tf.keras.Sequential([
                                  tf.keras.layers.Dense(13),
                                  tf.keras.layers.Dense(7),
                                  tf.keras.layers.Dense(6),
                                  tf.keras.layers.Dense(1)
      ])

      # STEP2: Compiling the model

      model.compile(loss= tf.keras.losses.mae,
                    optimizer= tf.keras.optimizers.SGD(),
                    metrics= ['mae']
                    )

      # STEP3: Fit the model

      h3= model.fit(x_train, y_train, epochs= 100, verbose=0)
```

```
[109] model.evaluate(x_test,y_test)
```

```
      4/4 [==============================] - 0s 2ms/step - loss: 1.1991 - mae: 1.1991
      [1.1990532875061035, 1.1990532875061035]
```

By trying different train and test splits, we chose the best model in the ratio of 70:30.

BAGGING :

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingRegressor
```

```
[90] x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2, random_state=42)
```

```
[91] bgr = BaggingRegressor(n_estimators=100, random_state=1)
     bgr.fit(x_train, y_train)
```

```
BaggingRegressor(n_estimators=100, random_state=1)
```

```
[92] bgr.score(x_test, y_test)
```

```
0.8053875261776013
```

```
[93] mae = metrics.mean_absolute_error(y_test, bgr.predict(x_test))
     print(mae)
```

```
1.2254430379746837
```

BOOSTING :

ADAPTIVE BOOSTING

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
```

```
[104] x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.4, random_state=42)
```

```
[105] # Create adaboost classifer object
      abc = AdaBoostRegressor(n_estimators=100,
                              learning_rate=0.6)
      # Train Adaboost Classifer
      model = abc.fit(x_train, y_train)
      model.score(x_test, y_test)
```

```
0.8474046327598734
```

```
[106] mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
      print(mae)
```

```
1.1387809410109313
```

GRADIENT BOOSTING

```
[107] from sklearn.ensemble import GradientBoostingRegressor
```

```
[143] x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.25, random_state=42)
```

```
[144] reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.6,max_depth=1,
       random_state=0).fit(x_train, y_train)
      reg.score(x_test, y_test)
```

```
0.8180597501318607
```

```
[145] mae = metrics.mean_absolute_error(y_test, reg.predict(x_test))
      print(mae)
```

```
1.442962140534077
```

XG BOOSTING

```
[123] from xgboost import XGBRegressor
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.4, random_state=42)
```

```
[137] xgbr = XGBRegressor()
      xgbr.fit(x_train, y_train)
```

```
[13:53:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor()
```

```
[138] mae = metrics.mean_absolute_error(y_test, xgbr.predict(x_test))
      print(mae)
```

```
1.1314074895049953
```

```
[139] xgbr.score(x_test, y_test)
```

```
0.8584528252545545
```