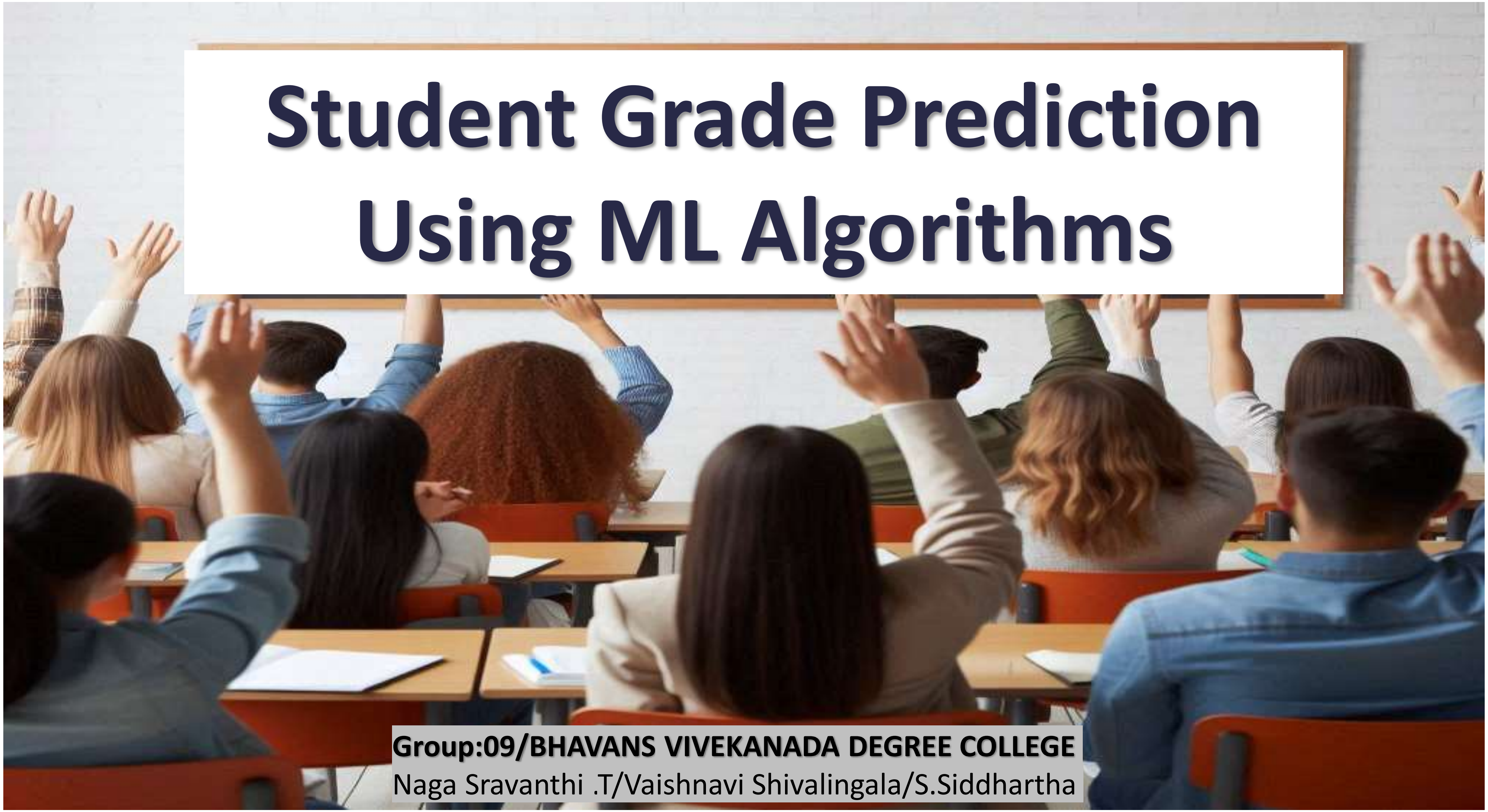
Student Grade Prediction Using ML Algorithms



Group:09/BHAVANS VIVEKANADA DEGREE COLLEGE
Naga Sravanthi .T/Vaishnavi Shivalingala/S.Siddhartha

Abstract

- The project aims to create a model that can accurately predict student grades, which will help improve learning and support students who may need extra help.

- This study looks at different machine learning methods, like Linear Regression, Decision Trees, Random Forests, K-Nearest Neighbors, Support Vector Machines, Bagging, and Boosting, to forecast how well students will do. By using these models, teachers can spot students who are struggling and take action to help them succeed.

Objective

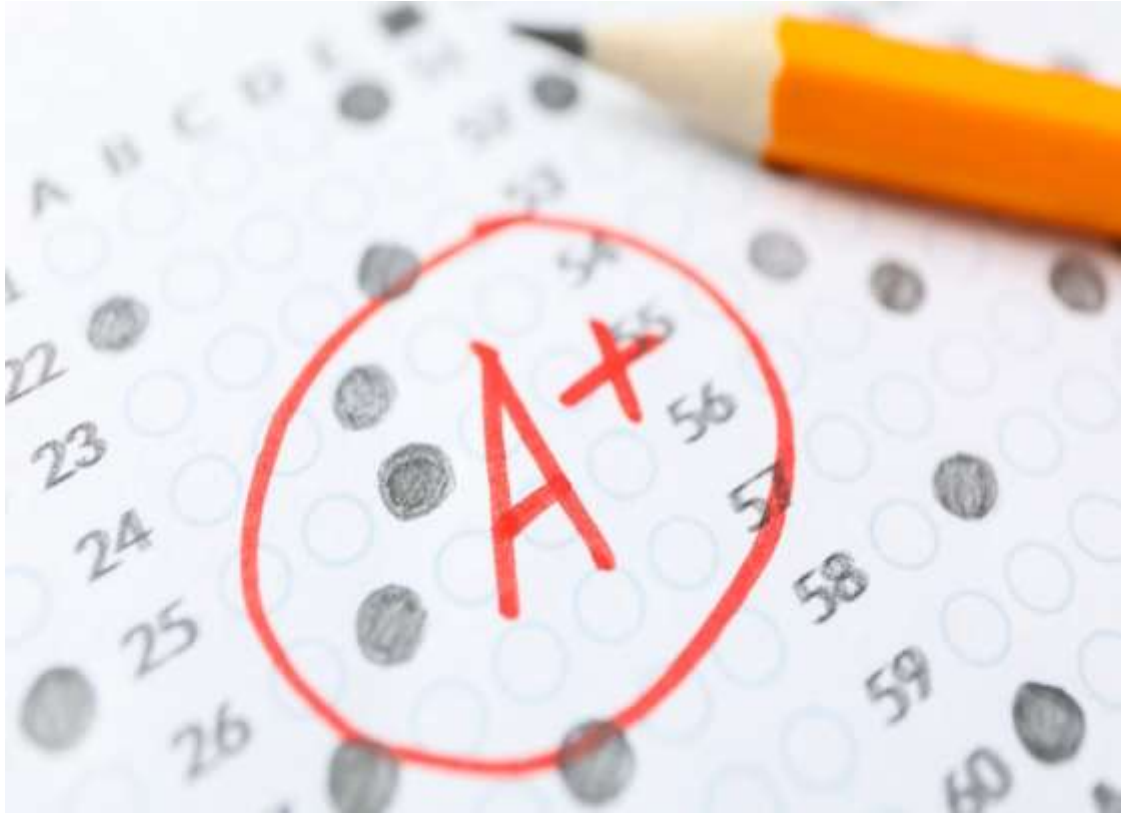
To find the best machine learning model for predicting student grades so that schools can identify students who need extra help and improve their learning outcomes. This goal aims to use data to support students and enhance their academic success.

CONTENT

- Introduction 01
- Literature Review 02
- Data Pre-Processing 03
- Exploratory Data Analysis 08
- Data Modeling & Evaluation 15
- Summary 24
- Future Scope 25

INTRODUCTION

Student grade prediction analyzes data to identify patterns in student performance, helping educators understand which students may excel or struggle. With advancements in machine learning, schools can gain insights into the factors influencing grades and predict future performance. These tools enable personalized strategies to address individual needs, enhance learning experiences, and improve academic success.



Literature Review

STUDENT GRADE PREDICTION USING GRADIENT BOOSTING CLASSIFIER

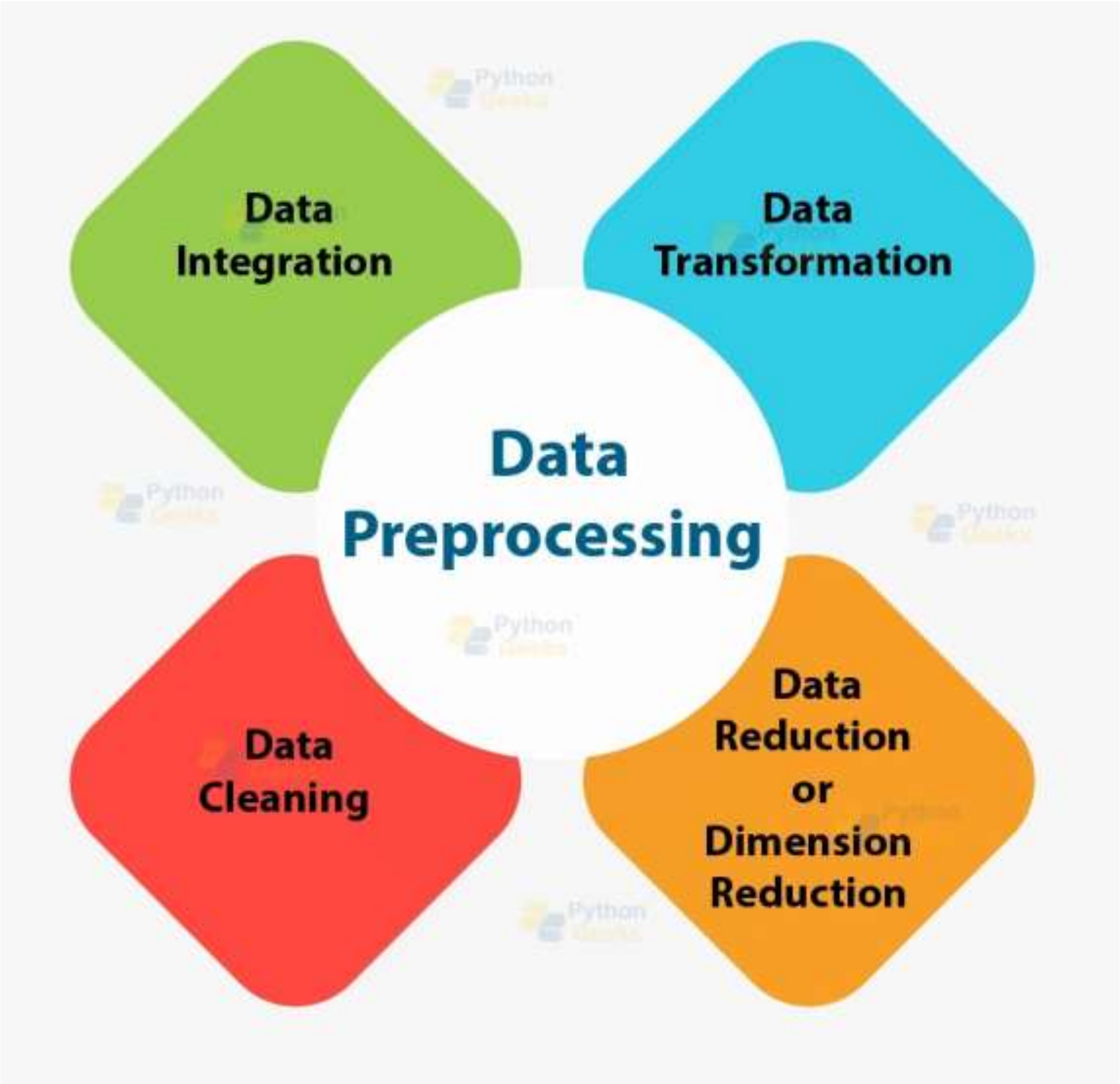
Annakula Srikanth, Alluri Abhi Karthikeya, and Dr. B. Venkateswara Rao

Technological Impact: The study highlights the use of the Gradient Boosting Classifier to predict student grades accurately by addressing imbalanced datasets and leveraging features like demographics, academic history, and attendance. The model emphasizes early detection of at-risk students for timely interventions and supports automation in education for improved outcomes.

Techniques Used: Gradient Boosting Classifier, feature selection, pseudo-residual computation, and data visualization techniques.

Published In: *International Journal of Advanced Research and Innovative Ideas in Education (IJARIIE)*, Vol-9, Issue-1, 2023

Data
Pre-Processing



Data

- **Data Set:** The dataset contains 395 records (rows) and 33 columns (attributes).

- **Source:** https://archive.ics.uci.edu/dataset/320/student+performance

- **Variables:**

Continuous Variable	Categorical Variable
age	school
Medu	sex
Fedu	address
traveltime	famsize
studytime	pstatus
failures	Mjob
famrel	Fjob
freetime	reason
goout	guardian
Dalc	schoolsup
Walc	famsup
health	paid
absences	activities
G1	nursery
G2	higher
G3	internet
	romantic

	school	sex	age	address	famsize	pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

5 rows × 33 columns

Data Cleaning

Data cleaning is a crucial step in preparing data for machine learning, ensuring accuracy and consistency for model training.

In our project, we renamed values in the "sex" column from 'F' and 'M' to 1 and 0, along with similar transformations in other columns, to make them machine-readable.

We also applied dummy variable encoding to categorical columns to convert them into numerical values.

Additionally, we checked for missing or null values across the dataset and confirmed there were none, ensuring the data was complete and ready for modeling


```
['F' 'M']
sex
F      208
M      187
```

```
[1 0]
sex
1      208
0      187
```

- To perform dummy variable encoding we divided the data into two sets

  - continuous data
  - categorical data

- Renaming the columns names:

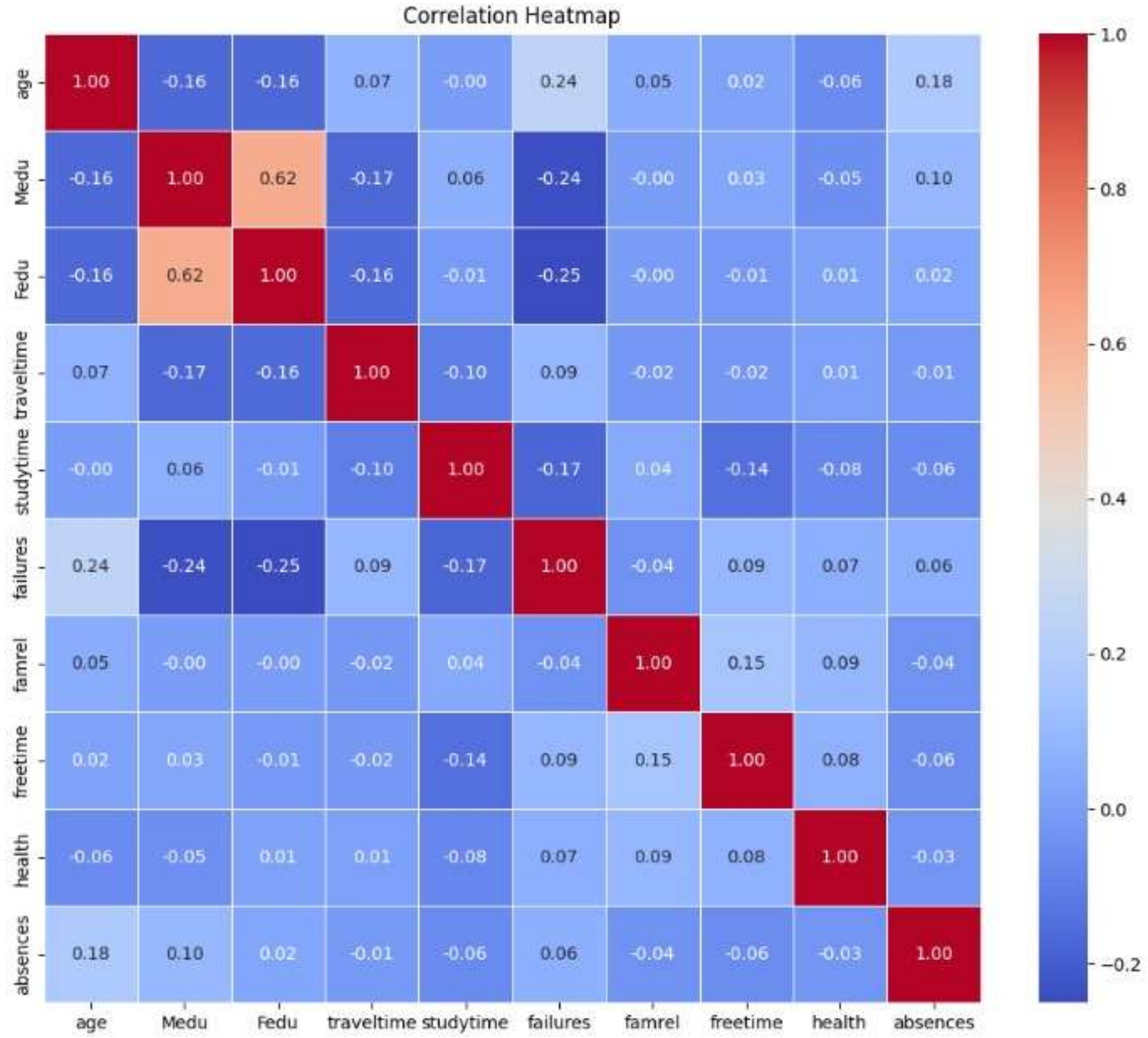
Column1	Column2
Original value names	Renamed value names
GP,MS	0,1
F,M	1,0
U,R	0,1
T,A	0,1
GT3,LE3	0,1
NO,YES	0,1
YES,NO	1,0
NO,YES	0,1
YES,NO	1,0
YES,NO	1,0
YES,NO	1,0
YES,NO	1,0
NO,YES	0,1

```python
X = pd.get_dummies(X, columns=['Fjob', 'Mjob', 'reason', 'guardian'], drop_first=True, dtype=int)
print(X)
```

```
     school  sex  age  address  famsize  pstatus  Medu  Fedu  traveltime  \
0         0    1   18        0        0        1     4     4           2
1         0    1   17        0        0        0     1     1           1
2         0    1   15        0        1        0     1     1           1
3         0    1   15        0        0        0     4     2           1
4         0    1   16        0        0        0     3     3           1
..      ...  ...  ...      ...      ...      ...   ...   ...         ...
390       1    0   20        0        1        1     2     2           1
391       1    0   17        0        1        0     3     1           2
392       1    0   21        1        0        0     1     1           1
393       1    0   18        1        1        0     3     2           3
394       1    0   19        0        1        0     1     1           1

     studytime  ...  Fjob_teacher  Mjob_health  Mjob_other  Mjob_services  \
0            2  ...             1            0           0              0
1            2  ...             0            0           0              0
2            2  ...             0            0           0              0
3            3  ...             0            1           0              0
4            2  ...             0            0           1              0
..         ...  ...           ...          ...         ...            ...
390          2  ...             0            0           0              1
391          1  ...             0            0           0              1
392          1  ...             0            0           1              0
393          1  ...             0            0           0              1
394          1  ...             0            0           1              0

     Mjob_teacher  reason_home  reason_other  reason_reputation  \
0               0            0             0                  0
1               0            0             0                  0
2               0            0             1                  0
3               0            1             0                  0
4               0            1             0                  0
..            ...          ...           ...                ...
390             0            0             0                  0
391             0            0             0                  0
392             0            0             0                  0
393             0            0             0                  0
394             0            0             0                  0

     guardian_mother  guardian_other
0                  1               0
1                  0               0
2                  1               0
3                  1               0
4                  0               0
..               ...             ...
390                0               1
391                1               0
392                0               1
393                1               0
394                0               0

[395 rows x 39 columns]
```

Exploratory Data Analysis

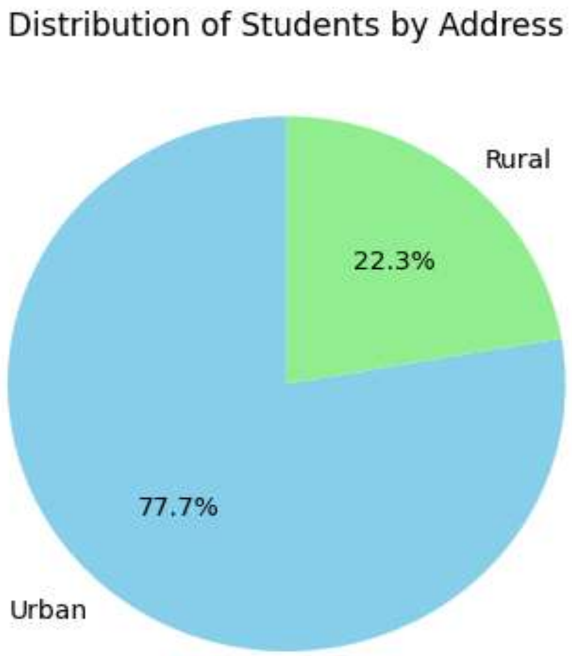
Correlation Matrix

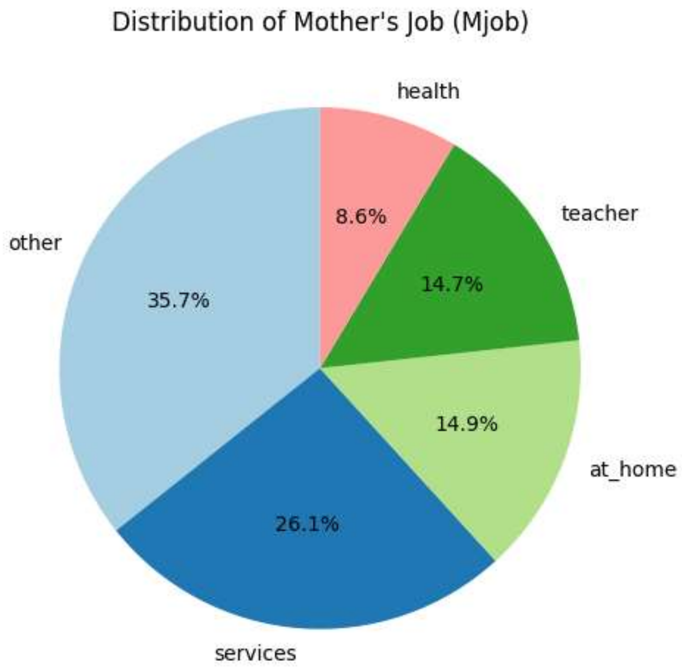


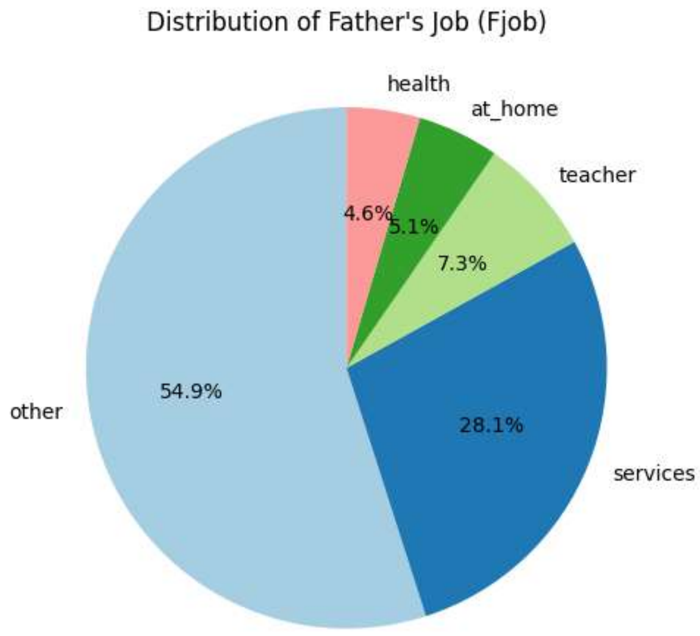
From the correlation heatmap, here are the pairs of terms that are most positively correlated:

- **Medu and Fedu**
- **Medu and absences**
- **Fedu and absences**
- **Study time and failures**
- **famrel and free time**

Pie-Chart



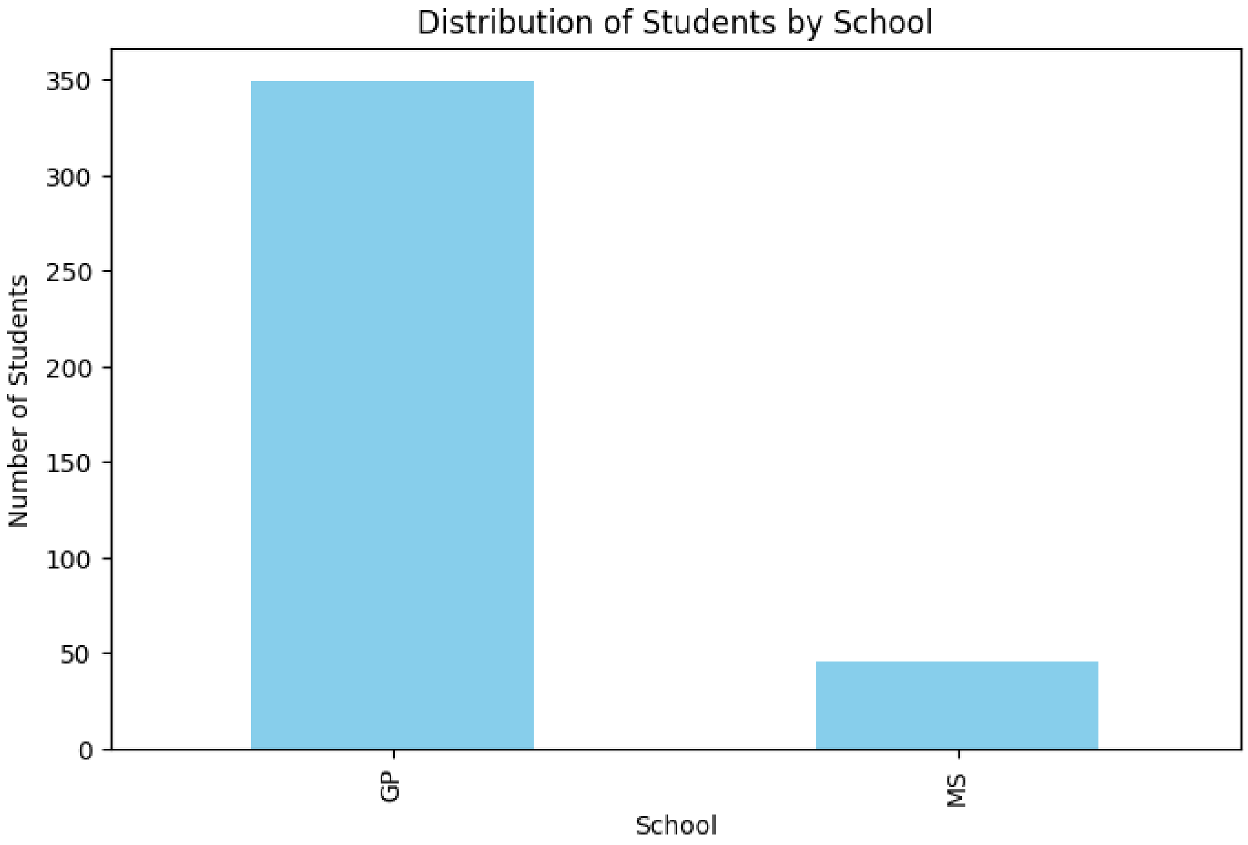


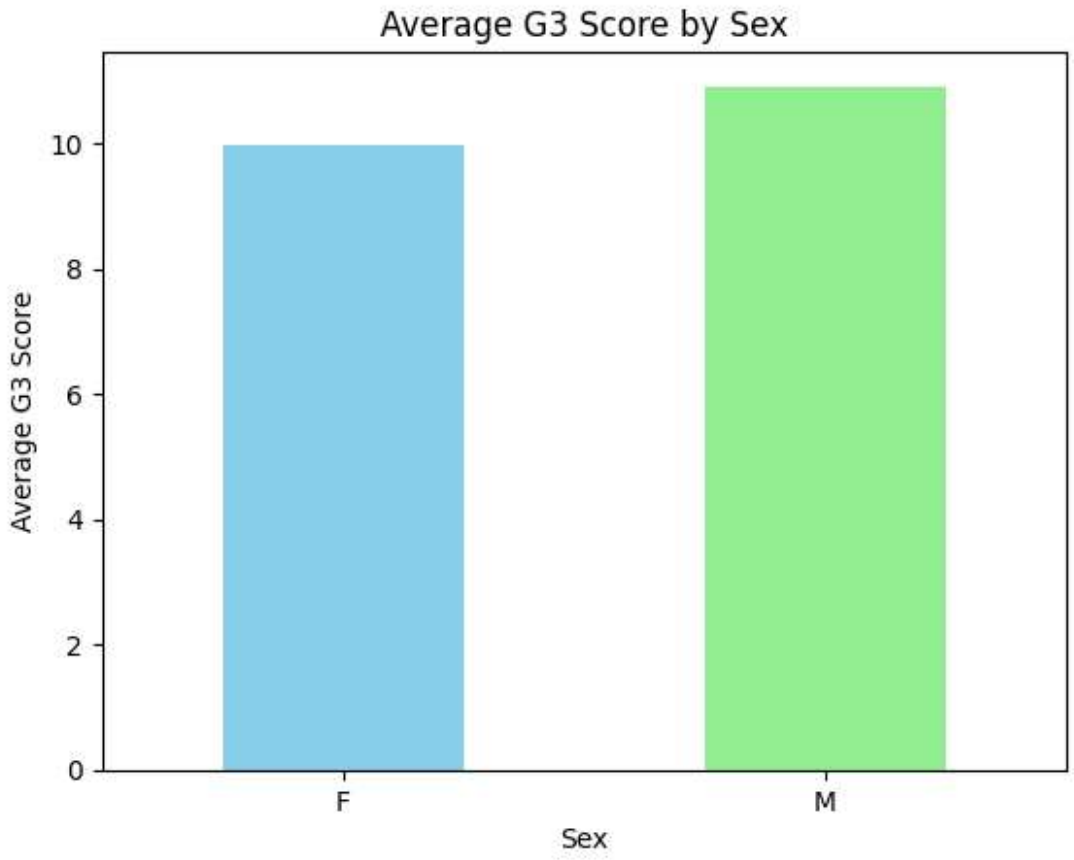


- This graph shows the distribution of students based on their address.
- Most students (77.7%) live in urban areas, while a smaller portion (22.3%) live in rural areas.

- The pie charts show the distribution of occupations for mothers and fathers. Most mothers have "other" jobs, while "services" is common for fathers. A significant number of mothers are "at home." "Other" jobs are also the most common for fathers.

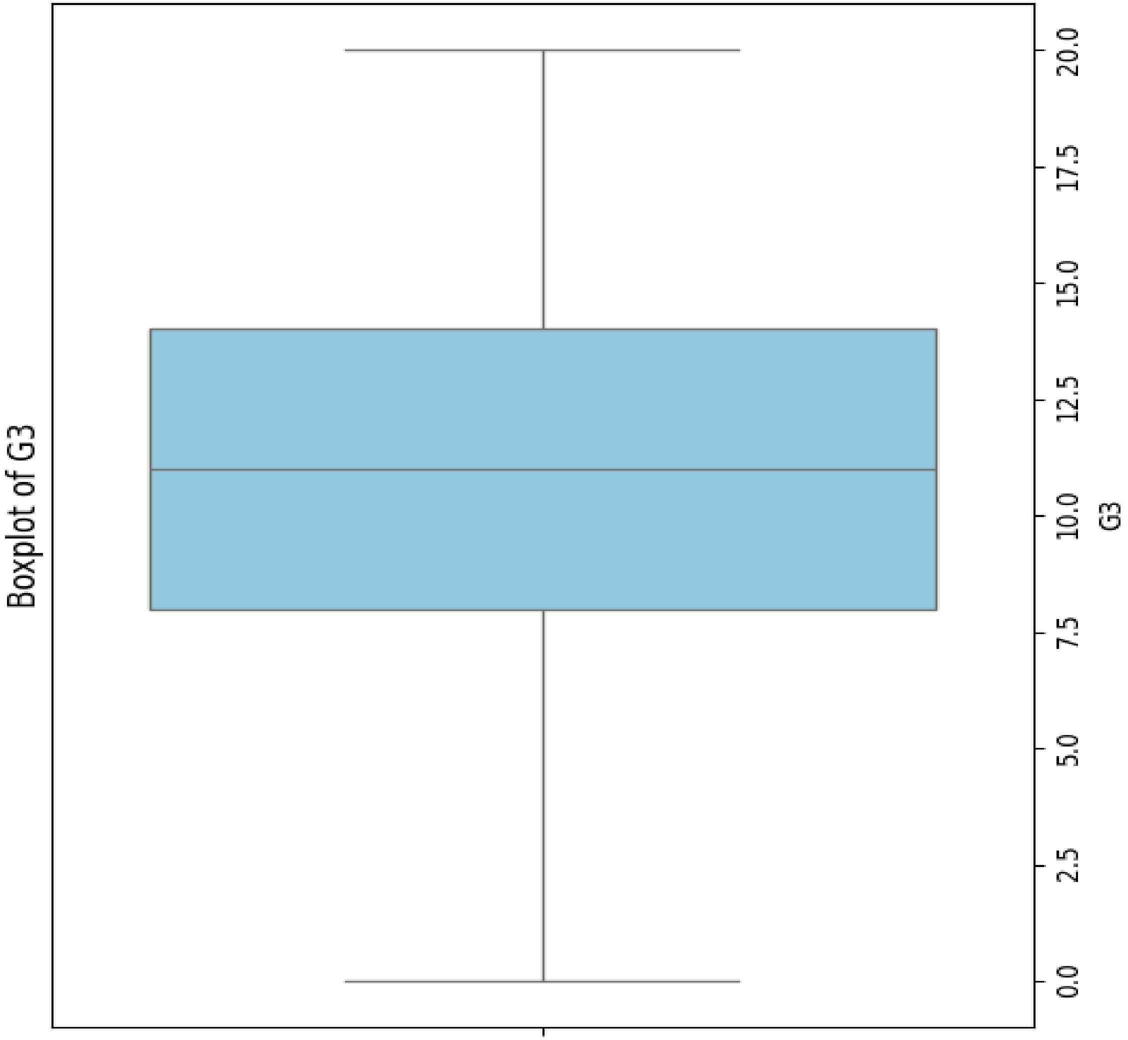
Bar-Plot





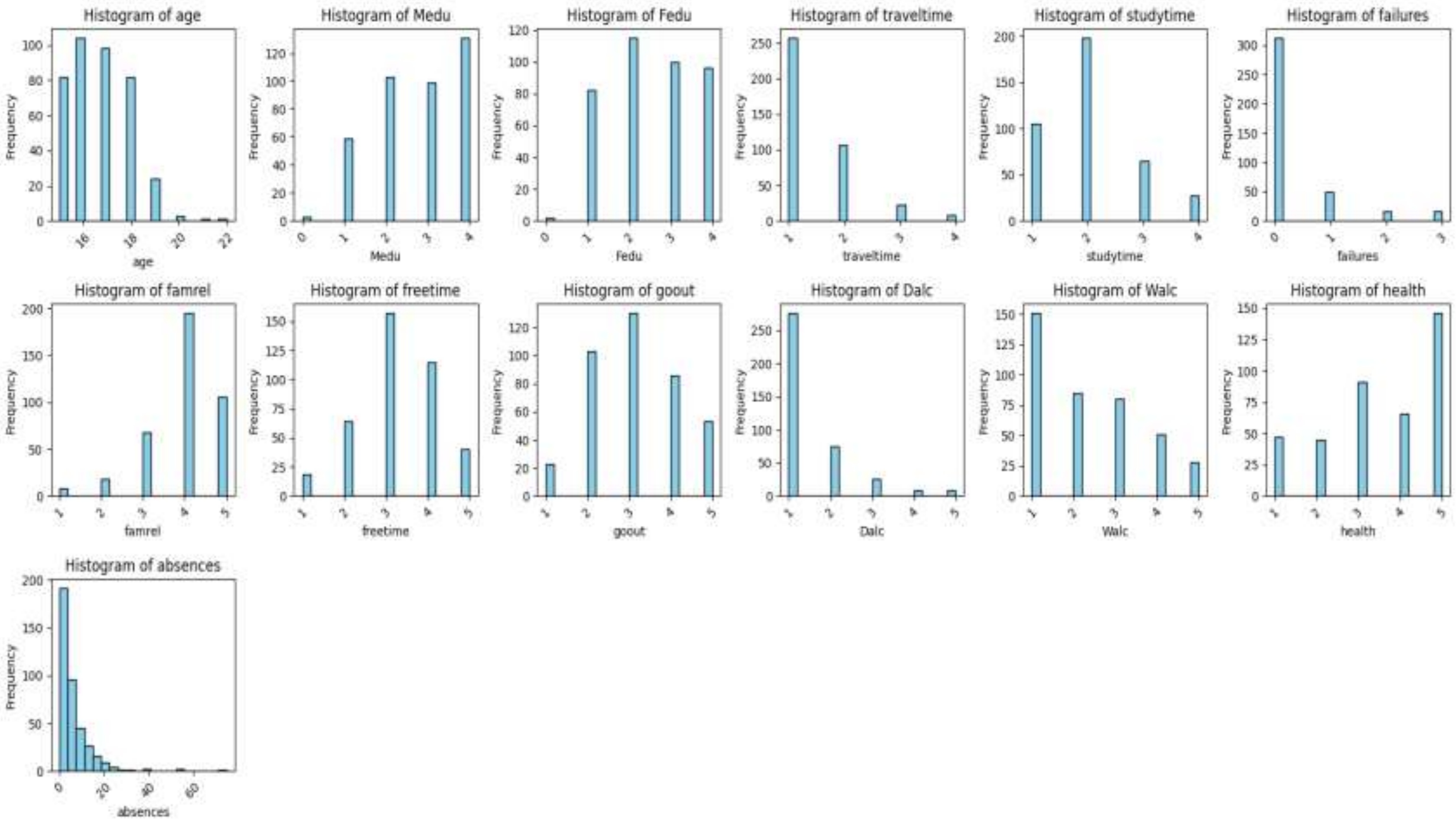
- This bar chart displays the distribution of students across two schools (GP and MS). The height of each bar represents the number of students enrolled in that respective school.
- This bar chart compares the average final grade (G3) for students of different sexes (Female and Male). The height of each bar represents the average G3 score for the corresponding sex.

Box Plot



- The boxplot shows the distribution of G3 scores. The median grade is around 11, and the majority of students scored between 9 and 13. The box is relatively narrow, indicating a small spread in the middle 50% of the data. There are some outliers on the lower end.

Histogram



- The histograms show the distribution of various factors. Most students are between 15-18, have parents with primary education, and have short travel times. They study for 2-3 hours, have good family relationships, and consume alcohol moderately. Absences are generally low.

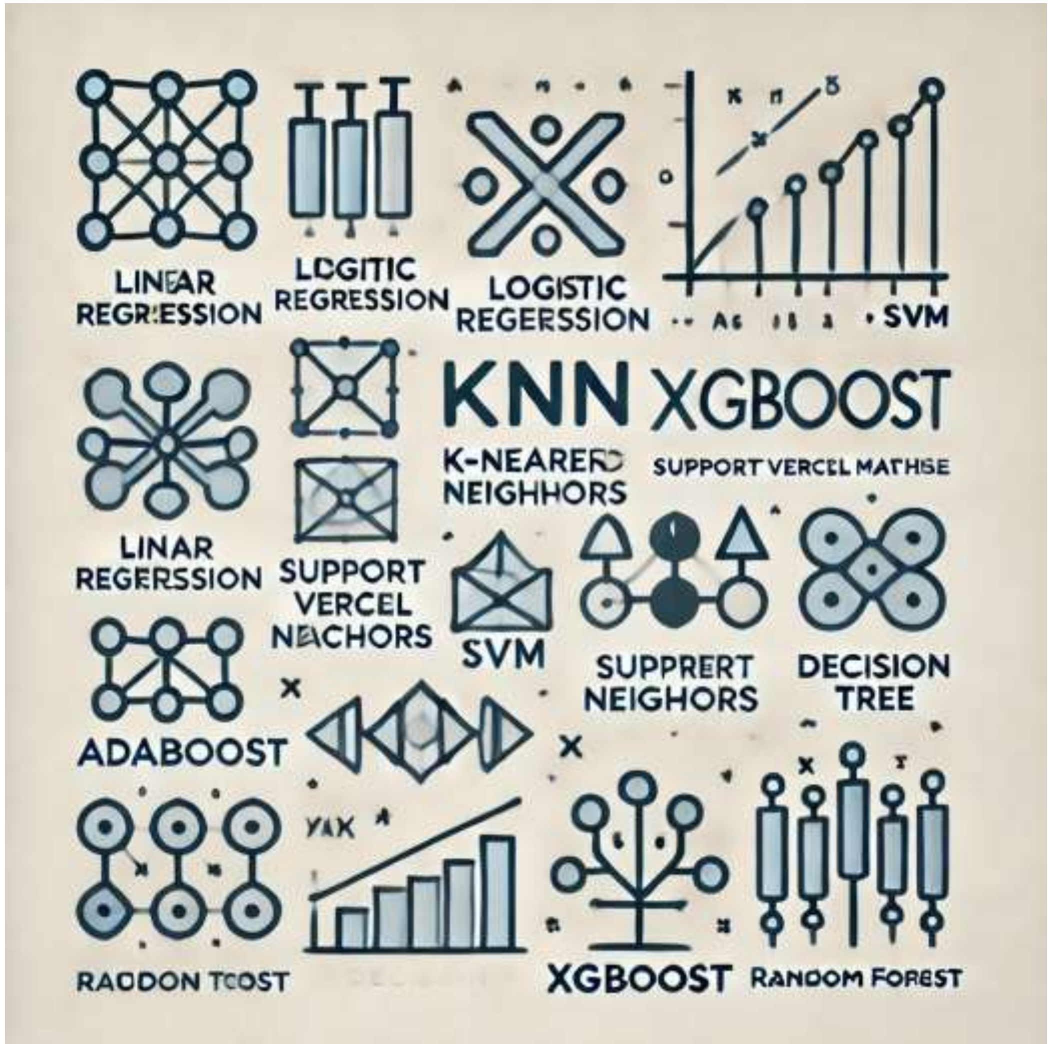
Multicollinearity Check

Variables with the greatest variance inflation factor (VIF > 3) were

	variables	VIF
0	school	1.6
1	sex	3.1
2	age	81.9
3	address	1.8
4	famsize	1.6
5	pstatus	1.3
6	Medu	21.3
7	Fedu	13.5
8	traveltime	6.7
9	studytime	9.4
10	failures	1.7
11	schoolsup	1.3
12	famsup	3.3
13	paid	2.4
14	activities	2.4
15	nursery	5.6
16	higher	22.8
17	internet	7.3
18	romantic	1.7
19	famrel	23.1
20	freetime	14.3
21	goout	12.7
22	Dalc	7.6
23	Walc	9.9
24	health	8.6
25	absences	1.8
26	Fjob_health	2.1
27	Fjob_other	12.6
28	Fjob_services	6.8
29	Fjob_teacher	2.8
30	Mjob_health	2.5
31	Mjob_other	4.2
32	Mjob_services	3.9
33	Mjob_teacher	3.7
34	reason_home	2.0
35	reason_other	1.4
36	reason_reputation	2.1
37	guardian_mother	4.8
38	guardian_other	1.8

	variables	VIF
0	school	1.4
1	sex	2.2
2	address	1.5
3	famsize	1.5
4	pstatus	1.2
5	failures	1.4
6	schoolsup	1.3
7	famsup	2.9
8	paid	2.3
9	activities	2.1
10	romantic	1.6
11	absences	1.7
12	Fjob_health	1.2
13	Fjob_services	1.6
14	Fjob_teacher	1.3
15	Mjob_health	1.5
16	Mjob_other	2.3
17	Mjob_services	2.3
18	Mjob_teacher	1.9
19	reason_home	1.8
20	reason_other	1.3
21	reason_reputation	1.9
22	guardian_other	1.3

Machine Learning Algorithms



ML Algorithms

- *Linear Regression*
- *K-Nearest Neighbors(KNN)*
- *Decision Tree*
- *Random Forest*
- *XG Boost*
- *Ada Boost*
- *Support Vector Machine(SVM)*
- *ANN*

Linear Regression

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.149	0.124	3.336	3.401
65-35	0.161	0.133	3.417	3.482
70-30	0.190	0.159	3.288	3.341
75-25	0.202	0.220	3.436	3.350
80-20	0.141	0.158	3.395	3.308

KNN

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.175	0.180	3.392	3.415
65-35	0.167	0.165	3.529	3.601
70-30	0.153	0.140	3.477	3.549
75-25	0.162	0.165	3.570	3.632
80-20	0.148	0.082	3.429	3.613

Decision Tree Regressor

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	-0.198	-0.198	3.740	3.803
65-35	-0.434	-0.183	4.381	3.812
70-30	-0.532	-0.035	4.277	3.470
75-25	0.200	-0.169	3.151	3.944
80-20	-0.166	-0.171	3.721	3.677

Random Forest

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.299	0.264	3.051	3.134
65-35	0.320	0.299	3.135	3.173
70-30	0.326	0.246	3.058	3.167
75-25	0.359	0.353	3.007	3.007
80-20	0.291	0.210	3.024	3.094

XG Boost

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.103	0.024	3.375	3.465
65-35	0.196	0.101	3.359	3.405
70-30	0.186	0.110	3.328	3.400
75-25	0.349	0.204	3.016	3.271
80-20	0.153	-0.085	3.409	3.532

Ada Boost

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.213	0.170	3.267	3.491
65-35	0.193	0.158	3.481	3.619
70-30	0.143	0.101	3.483	3.634
75-25	0.142	0.186	3.583	3.596
80-20	0.127	0.104	3.483	3.56

SVM

Train-Test Ratio	Model-1 (r2-Score)	Model-2 (r2-Score)	Model-1 (MAE)	Model-2 (MAE)
60-40	0.120	0.121	3.348	3.325
65-35	0.127	0.085	3.500	3.539
70-30	0.172	0.119	3.295	3.402
75-25	0.147	0.158	3.549	3.496
80-20	0.144	0.126	3.334	3.372

ANN

Train-Test Ratio	Model-1 (MAE)	Model-2 (MAE)	Architecture	Optimizer	Epochs
60-40	3.404	3.483	50-20-15-10-5-1	Adam	250
60-40	4.765	4.514	29-26-23-17-10-1	SGD	200
60-40	3.741	3.516	27-25-17-15-5-1	Adam	150
65-35	3.481	3.477	32-24-18-17-10-1	Adam	250
65-35	3.480	3.512	33-29-27-23-17-1	Adam	150
65-35	3.561	3.494	34-20-15-10-5-1	Adam	250
70-30	3.445	3.420	28-20-15-10-5-1	Adam	250
70-30	3.416	3.423	28-27-25-19-15-1	Adam	200
70-30	3.416	3.487	30-27-26-23-20-1	Adam	250
75-25	3.488	3.513	30-28-26-23-5-1	Adam	200
75-25	3.443	3.516	29-27-18-16-5-1	Adam	150
75-25	3.417	3.426	29-27-25-24-15-1	Adam	250
80-20	3.624	3.575	29-26-23-18-15-1	Adam	200
80-20	3.611	3.643	28-27-25-20-15-1	Adam	150
80-20	3.674	3.609	28-27-23-24-15-1	Adam	150

Algorithms Comparision

Model-1

Algorithms	MAE
Linear Regression	3.288
K-Nearest Neighbors(KNN)	3.392
Decision Tree	3.151
Random Forest	3.007
XG Boost	3.166
Ada Boost	3.267
Support Vector Machine(SVM)	3.295
ANN	3.404

Algorithms Comparision

Model-2

Algorithms	MAE
Linear Regression	3.341
K-Nearest Neighbors(KNN)	3.415
Decision Tree	3.470
Random Forest	3.007
XG Boost	3.271
Ada Boost	3.560
Support Vector Machine(SVM)	3.402
ANN	3.420

SUMMARY

In this project, a 75% training and 25% testing split was found to yield the best model performance. Random Forest was selected as the most effective model for predicting student grades, with both Model 1 and Model 2 showing similar results: an R^2 value of 0.359 and 0.353, and a Mean Absolute Error (MAE) of 3.007.

These findings suggest that the models explain about 35% of the variance in the data, indicating room for further improvement.

Overall, the project demonstrates how machine learning can offer valuable insights into student performance and potentially guide educational improvements.

Future Scope

To improve model performance, we can add relevant features example social media usage, study materials.

Reduce multicollinearity with PCA and remove outliers.

Tune hyperparameters example tree count, max depth in Random Forest.

Using Neural Networks (MLP) can capture complex relationships, especially with larger datasets.

Expanding the dataset with diverse data can also enhance accuracy.

Work Distribution

NAME	WORK DONE
VAISHNAVI SHIVALINGALA	Collecting Data and Performing Data pre-processing
SIDHHARTHA.S	Exploratory Data Analysis
NAGA SRAVANTHI.T	ML Algorithms

THANK YOU

Done By:
Naga Sravanthi T
Vaishnavi Shivalingala
S.Siddhartha

APPENDIX

Loading the Dataset

```
data= pd.read_excel('Predict student performance.xlsx')
data.head()
```

	school	sex	age	address	famsize	pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

5 rows × 33 columns

Null Values

```
data.isna().sum()
```

	0
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
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

Checking for the data type

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      395 non-null    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   pstatus     395 non-null    object
 6   Medu        395 non-null    int64
 7   Fedu        395 non-null    int64
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64
 13  studytime   395 non-null    int64
 14  failures    395 non-null    int64
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 23  famrel      395 non-null    int64
 24  freetime    395 non-null    int64
 25  goout       395 non-null    int64
 26  Dalc        395 non-null    int64
 27  Walc        395 non-null    int64
 28  health      395 non-null    int64
 29  absences    395 non-null    int64
 30  G1          395 non-null    int64
 31  G2          395 non-null    int64
 32  G3          395 non-null    int64
dtypes: int64(16), object(17)
memory usage: 102.0+ KB
```

Dividing the data set

```
X=data.drop(['G1','G2','G3'],axis=1)
print(X)
y = data[['G3']]
print(y)
```

```
     school  sex  age  address  famsize  pstatus  Medu  Fedu      Mjob  \
0         0    1   18        0        0        1     4     4   at_home
1         0    1   17        0        0        0     1     1   at_home
2         0    1   15        0        1        0     1     1   at_home
3         0    1   15        0        0        0     4     2    health
4         0    1   16        0        0        0     3     3     other
..      ...  ...  ...      ...      ...      ...   ...   ...       ...
390       1    0   20        0        1        1     2     2  services
391       1    0   17        0        1        0     3     1  services
392       1    0   21        1        0        0     1     1     other
393       1    0   18        1        1        0     3     2  services
394       1    0   19        0        1        0     1     1     other
```

```
         Fjob  ... higher internet  romantic  famrel  freetime  goout  Dalc  \
0     teacher  ...      1        0         0       4         3      4     1
1       other  ...      1        1         0       5         3      3     1
2       other  ...      1        1         0       4         3      2     2
3    services  ...      1        1         1       3         2      2     1
4       other  ...      1        0         0       4         3      2     1
..        ...  ...    ...      ...       ...     ...       ...    ...   ...
390  services  ...      1        0         0       5         5      4     4
391  services  ...      1        1         0       2         4      5     3
392     other  ...      1        0         0       5         5      3     3
393     other  ...      1        1         0       4         4      1     3
394   at_home  ...      1        1         0       3         2      3     3

     Walc  health  absences
0       1       3         6
1       1       3         4
2       3       3        10
3       1       5         2
4       2       5         4
..    ...     ...       ...
390     5       4        11
391     4       2         3
392     3       3         3
393     4       5         0
394     3       5         5
```

```
[395 rows x 30 columns]
     G3
0     6
1     6
2    10
3    15
4    10
..   ..
390   9
391  16
392   7
393  10
394   9

[395 rows x 1 columns]
```

Linear Regression

MODEL 1:

```
[ ] lm2 = LinearRegression()
    lm2.fit(X_train1, y_train1)
    y_pred1 = lm2.predict(X_test1)
    print(np.sqrt(metrics.mean_squared_error(y_test1, y_pred1)))
```

```
4.322443692042204
```

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.14904009255355366
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
3.3362728448111754
```

KNN

MODEL 1:

```
from sklearn.metrics import r2_score
r2_score(y_test1,y_pred1)
```

```
0.17535613790007187
```

```
from sklearn import metrics
metrics.mean_absolute_error(y_test1,y_pred1)
```

```
3.392706449668475
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test1,y_pred1)
```

```
18.10572921151583
```

```
mse = mean_squared_error(y_test1, y_pred1)
rmse = np.sqrt(mse)
rmse
```

```
4.255082750254785
```

Decision Tree Regressor

MODEL 1:

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
-0.1988980213645195
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
3.740506329113924
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test1,y_pred1)
```

```
26.32278481012658
```

```
[ ] mse = mean_squared_error(y_test1, y_pred1)
    rmse = np.sqrt(mse)
    rmse
```

```
5.130573536177664
```

Random Forest

MODEL 1:

```
[ ]  from sklearn.metrics import r2_score
     r2_score(y_test1,y_pred1)

     0.29994663942127864
```

```
[ ]  from sklearn import metrics
     metrics.mean_absolute_error(y_test1,y_pred1)

     3.0512658227848104
```

```
[ ]  from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test1,y_pred1)

     15.370243037974687
```

```
[ ]  mse = mean_squared_error(y_test1, y_pred1)
     rmse = np.sqrt(mse)
     rmse

     3.920490152771039 4
```

Ada Boost

MODEL 1:

```
[ ]  print('r2_score')
     from sklearn.metrics import r2_score
     r2_score(y_test1,y_pred1)
```

```
r2_score
0.21350511046776033
```

```
[ ]
     print('mean_absolute_error')
     from sklearn import metrics
     metrics.mean_absolute_error(y_test1,y_pred1)
```

```
mean_absolute_error
3.2670376727153336
```

```
print('mean_squared_error')
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test1,y_pred1)
mse = mean_squared_error(y_test1, y_pred1)
rmse = np.sqrt(mse)
rmse
```

```
mean_squared_error
4.155494841159424
```

XG Boost

MODEL 1:

```
print('r2_score')
from sklearn.metrics import r2_score
r2_score(y_test1,y_pred1)
```

```
r2_score
0.10385710000991821
```

```
print('mean_absolute_error')
from sklearn import metrics
metrics.mean_absolute_error(y_test1,y_pred1)
```

```
mean_absolute_error
3.375464732016007
```

```
print('mean_squared_error')
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test1,y_pred1)
mse = mean_squared_error(y_test1, y_pred1)
rmse = np.sqrt(mse)
rmse
```

```
mean_squared_error
4.435713232637776
```

SVM

MODEL 1:

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.12743524098622694
```

```
from sklearn import metrics
metrics.mean_absolute_error(y_test1,y_pred1)
```

```
3.3481318304434216
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test1,y_pred1)
```

```
19.157871624709344
```

```
[ ] mse = mean_squared_error(y_test1, y_pred1)
    rmse = np.sqrt(mse)
    rmse
```

```
4.376970599022724
```

Linear Regression

MODEL 2:

```
[ ] lm2 = LinearRegression()
    lm2.fit(X_train_nomulti1, y_train_nomulti1)
    y_pred_nomulti1 = lm2.predict(X_test_nomulti1)
    print(np.sqrt(metrics.mean_squared_error(y_test_nomulti1, y_pred_nomulti1)))

    4.384405016624924
```

```
from sklearn.metrics import r2_score
r2_score(y_test_nomulti1,y_pred_nomulti1)

0.12446856810181994
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)

    3.4017216651776176
```

KNN

MODEL 2:

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
0.18011089165781902
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
3.41500904159132
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
```

```
18.001334711099627
```

```
[ ] mse = mean_squared_error(y_test_nomulti1, y_pred_nomulti1)
    rmse = np.sqrt(mse)
    rmse
```

```
4.242797981415051
```

Decision Tree Regressor

MODEL 2:

```
[ ]  from sklearn.tree import DecisionTreeRegressor
```

```
[ ]  clf = DecisionTreeRegressor()
     clf = clf.fit(X_train_nomulti1,y_train_nomulti1)
```

```
[ ]  y_pred_nomulti1 = clf.predict(X_test_nomulti1)
```

Evaluation Metric

```
[ ]  from sklearn.metrics import r2_score
     r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
-0.19832148949562578
```

```
from sklearn import metrics
metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
3.8037974683544302
```

```
[ ]  from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
```

```
26.310126582278482
```

Random Forest

MODEL 2:

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
0.26443903991307427
```

```
from sklearn import metrics
metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
3.134565400843882
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
```

```
16.149841372714484
```

```
[ ] mse = mean_squared_error(y_test_nomulti1, y_pred_nomulti1)
    rmse = np.sqrt(mse)
    rmse
```

```
4.018686523320087
```

XG Boost

MODEL 2:

```
[ ] print('r2_score')
    from sklearn.metrics import r2_score
    r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
r2_score
0.02418208122253418
```

```
print('mean_absolute_error')
from sklearn import metrics
metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
mean_absolute_error
3.4651776094791256
```

```
[ ] print('mean_squared_error')
    from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
    mse = mean_squared_error(y_test_nomulti1, y_pred_nomulti1)
    rmse = np.sqrt(mse)
    rmse
```

```
mean_squared_error
4.62870171707907
```

Ada Boost

MODEL 2:

```
[ ] print('r2_score')
    from sklearn.metrics import r2_score
    r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
r2_score
0.17033888635291783
```

```
print('mean_absolute_error')
from sklearn import metrics
metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
mean_absolute_error
3.4913440279360306
```

```
print('mean_squared_error')
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
mse = mean_squared_error(y_test_nomulti1, y_pred_nomulti1)
rmse = np.sqrt(mse)
rmse
```

```
mean_squared_error
4.268007388137377
```

SVM

MODEL 2:

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test_nomulti1,y_pred_nomulti1)
```

```
0.12163172269563127
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test_nomulti1,y_pred_nomulti1)
```

```
3.3252752247868367
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test_nomulti1,y_pred_nomulti1)
```

```
19.285292606629987
```

```
[ ] mse = mean_squared_error(y_test_nomulti1, y_pred_nomulti1)
    rmse = np.sqrt(mse)
    rmse
```

```
4.391502317730231
```