

# ASSESSING THE FACTORS THAT CONTRIBUTE TO STUDENT ENGAGEMENT: A MULTIPLE REGRESSION APPROACH

## Team Members:

S NO	NAME	REG.NO	EMAIL
1	Aishwariyaa.T	917721S001	aishwariyaa@student.tce.edu
2	Anu Gayathri.S	917721S002	anugayathri@student.tce.edu
3	Narmadha.R	917721S024	narmadhar@student.tce.edu

## 1) Objective of the study:

The primary aim of utilizing multiple regression analysis to recognize the determinants that influence student engagement is to gain a more comprehensive comprehension of the intricate interplay between varied factors and the level of student engagement. Educators can employ this statistical approach to discern the most influential factors that bear on student engagement, such as class size, study hours, teaching style, class hours, and technology usage.

Through the analysis of these factors, educators can ascertain the specific areas they should concentrate on to optimize their pedagogical practices and classroom environments. Moreover, the utilization of multiple regression analysis endeavours to enable educators to make well-informed decisions concerning which interventions or modifications to incorporate in order to improve student engagement. They can gauge the effectiveness of interventions, such as new teaching methodologies or technological tools, to determine if they exert a significant influence on student engagement.

Finally, multiple regression analysis can serve the purpose of identifying which factors affect student engagement differentially for distinct students. By comprehending these subtleties, educators can customize their teaching approaches to meet the distinct needs of individual students.

In summary, the aim of implementing multiple regression analysis to recognize the determinants that affect student engagement is to enhance student learning outcomes by elevating engagement levels via well-informed decision-making and bespoke pedagogical practices.

## **2) Experiment conducted:**

### **2.1 Data Collected:**

1. Email:

Emails of students are collected although they don't make any impact on the study.

2. Name:

Names of students are also gathered in order to maintain track of who all answered, even though they have no bearing on the study, which will be dropped later.

3. Gender:

The gender of students is collected to demonstrate that they are unrelated to the observations and are not required for the study, so that they can be dropped with the necessary proof.

4. Age:

Age is also collected to see if it has any effect on individual student scores, which it does not, therefore they are later eliminated.

5. Marks gained:

Marks gained by each student out of 700, semester marks of college students are collected and it is considered as the output variable.

6. Class Strength:

No. of students present in each of the individual's class is collected in order to predict it's impact on the marks they score, whether they make any changes.

7. Study Hours:

Each individual's study hours per day is also collected in order to get to know about it's impact on the individual's score.

8. Courses applied:

We asked individuals if they have taken any courses except for college courses, to check if students who pursued any course has it's impact on their scores compared to those who didn't attend any.

9. Rating of the study materials:

We also collected ratings of the study materials given to them from the college to check if those study materials make an impact on their individual scores depending upon their ratings.

10. Technology:

The technology used to communicate with the students in the class is also asked to specify, in order to identify if they have made any impact on their understanding which made difference in the scores. The categorical variables 'blackboard, presentation, online course materials' are converted into dummy variables for preprocessing.

11. Teaching:

The teaching way is also asked to be specified in order to identify if they have made any impact on their understanding which made difference in the scores. Either it is project based or lecture based. Later these categorical variables are converted into dummy variables for preprocessing.

## 12. Hours:

Data on the study hours per day of individuals is collected. The purpose is to assess how it affects their individual scores, based on the number of hours they spend learning. The data will enable researchers to gain insights into how much study time is required to achieve high scores.

## 2.2 Participants from whom you collect data?

The participants from whom data is collected for this objective are college students. The study aims to identify the factors that impact student engagement and improve student learning outcomes through informed decision-making.


The data collected includes factors such as class size, study hours, teaching style, class hours, and technology use. By analyzing these factors, educators can determine which specific areas to focus on and make well-informed decisions about which interventions to implement to improve student engagement.

MARKS

For assignment

anugayathri04@gmail.com

Switch account

 Draft saved

\* Indicates required question

Email \*

anugayathri@gmail.com

Enter your name \*

Anu Gayathri

Gender

☐ Male

☒ Female

Clear selection

Age \*

19

19

Enter your total marks in your previous year \*  
give your answers like (marks you gained/total marks)  
  
650/700

Enter your class strength \*  
  
38

On average, how many hours per day do you spend studying? \*  
  
3

Did you take extra courses (or tutons)?  
  
☒ Yes  
☐ No  
  
Clear selection

Quality of learning materials given by your schools or colleges \*  
  

12345  
very poor☐ ☐ ☐ ☒ ☐ excellent

The Technology that is mostly used \*  
  
☒ presentation  
☐ Online course materials  
☐ blackboard

Which of the following teaching styles is mostly used in your class? \*  
  
☐ lecture based  
☒ project based

Enter your total class hours per day \*  
eg: 8 hrs per day  
  
6

## 2.3 Data collection:

The link to the questionnaire utilized to gather data has been included below for reference.

<https://docs.google.com/spreadsheets/u/0/d/1ofMJpgFzdUiRm6Dg-znwhCbNaS1KZ0mgX5pu-kYad78/edit>

## 2.4 Data Preprocessing with appropriate Python code:

#loading the dataset into df

```
df = pd.read_csv('marks1.csv')
```

```
df.head()
```

	Timestamp	Email address	Enter your name	Gender	Age	marks	class strength	study hours	courses	material	Technology	teaching	hours
0	4/23/2023 11:08:14	tejaswini.pk.2004@gmail.com	Tejaswini P K	Female	18.0	632	38	1.0	Yes	5	blackboard	project based	7
1	4/23/2023 11:09:11	keerthanakumar1625@gmail.com	Keerthana	Female	19.0	625	38	3.0	Yes	4	blackboard	project based	6
2	4/23/2023 11:09:58	shalinashifaya@gmail.com	Shalina Shifaya S	Female	19.0	537	66	0.0	No	2	presentation	lecture based	5
3	4/23/2023 11:10:00	bkaathithsanjay@gmail.com	Aathith Sanjay	Male	19.0	590	60	5.0	No	3	presentation	lecture based	8
4	4/23/2023 11:14:27	jenimaoscar@gmail.com	Jenima Oscar	Female	18.0	578	70	1.0	No	3	presentation	lecture based	5

These are the data collected from college students in order to identify the most influential factors for their individual scores.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 226 entries, 0 to 225
Data columns (total 13 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Timestamp            226 non-null   object
1   Email address        226 non-null   object
2   Enter your name      226 non-null   object
3   Gender               225 non-null   object
4   Age                  225 non-null   float64
5   marks                226 non-null   int64
6   class strength       226 non-null   int64
7   study hours          226 non-null   float64
8   courses              225 non-null   object
9   material             226 non-null   int64
10  Technology            226 non-null   object
11  teaching             226 non-null   object
12  hours                226 non-null   int64
dtypes: float64(2), int64(4), object(7)
memory usage: 23.1+ KB
```

There are 226 responses being recorded in total, and as you can see there are some record missing in some columns.

#dropping the columns

```
df1=df.drop(['Timestamp','Email address','Enter your name'],axis=1)
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 226 entries, 0 to 225
Data columns (total 10 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Gender              225 non-null   object
1   Age                 226 non-null   int64
2   marks              226 non-null   int64
3   class strength      226 non-null   int64
4   study hours         226 non-null   float64
5   courses             225 non-null   object
6   material            226 non-null   int64
7   Technology          226 non-null   object
8   teaching            226 non-null   object
9   hours              226 non-null   int64
dtypes: float64(1), int64(5), object(4)
memory usage: 17.8+ KB
```

These columns are being dropped because it's too obvious that they won't make any impact on the study. The 'df1.info()' function call will then display information about the DataFrame 'df1', including the number of non-null values in each column and the data type of each column.

# Check for missing values

```
df1.isnull().sum()
```

```
df1.value_counts()
```

	Gender	Age	marks	class strength	study hours	courses	material	Technology	teaching	hours	
	Female	10.0	670	40	1.0	Yes	5	blackboard	project based	7	1
		21.0	632	40	0.5	No	4	blackboard	lecture based	6	1
	Male	17.0	573	35	4.0	No	3	presentation	lecture based	8	1
			560	38	2.0	No	3	presentation	lecture based	7	1
			549	45	2.0	Yes	2	presentation	lecture based	8	1
											..
	Female	19.0	556	70	1.0	No	3	presentation	lecture based	5	1
			555	50	3.0	No	3	presentation	lecture based	6	1
			554	72	1.0	No	3	presentation	project based	5	1
				63	2.0	No	3	presentation	lecture based	6	1
	Male	25.0	671	70	4.0	Yes	5	blackboard	project based	5	1
	Length: 293, dtype: int64										

The first line of code df1.isnull().sum() will return the number of null values in each column of the dataframe df1. This can be useful to identify missing data and decide how to handle it, such as by imputing missing values or dropping rows with missing data.

The second line of code df1.value\_counts() will return the count of unique values in each column of the dataframe df1. This can be useful to get a better understanding of the distribution of values in each column, especially for categorical variables.

#dropping the rows that has null values

```
df2=df1.dropna()
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 223 entries, 0 to 225
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Gender          223 non-null    object
1   Age             223 non-null    float64
2   marks           223 non-null    int64
3   class_strength  223 non-null    int64
4   study_hours     223 non-null    float64
5   courses         223 non-null    object
6   material        223 non-null    int64
7   Technology      223 non-null    object
8   teaching        223 non-null    object
9   hours           223 non-null    int64
dtypes: float64(2), int64(4), object(4)
memory usage: 19.2+ KB
```

Now, we are dropping the whole row which has null value, that gives us a total of 223 data.

```
#creating dummy variables
df2['Gender'] = df2['Gender'].replace({"Female": 1, "Male": 0})
print(df2['Gender'])
```

```
0      1
1      1
2      1
3      0
4      1
..
221    0
222    0
223    0
224    0
225    0
Name: Gender, Length: 223, dtype: int64
```

Here, this code is replacing the string values of "Female" and "Male" in the 'Gender' column of the DataFrame 'df2' with integer values of 1 and 0, respectively. The resulting column will contain binary data, where 1 represents "Female" and 0 represents "Male". The printed output displays the modified 'Gender' column. The dummy variables 0 and 1 are used.

```
df2['courses'] = df2['courses'].replace({"Yes": 1, "No": 0})
print(df2['courses'])
```

```
0      1
1      1
2      0
3      0
4      0
..
221    1
222    0
223    0
224    0
225    1
Name: courses, Length: 223, dtype: int64
```

The above code replaces the string values "Yes" and "No" in the 'courses' column of a pandas DataFrame with the integer values 1 and 0, respectively. This is done to convert the categorical variable 'courses' into a numerical form that can be used as input for machine learning models. The new numerical values represent whether an individual has applied for courses 1 or not 0. The resulting output prints the modified 'courses' column of the DataFrame.

```
df2['Technology'] = df2['Technology'].replace({"blackboard": 2, "presentation": 1, "Online course materials": 0})
print(df2['Technology'])
```

```

0      2
1      2
2      1
3      1
4      1
..
221    2
222    2
223    1
224    2
225    2
Name: Technology, Length: 223, dtype: int64

```

This code replaces the values in the 'Technology' column of the DataFrame 'df2' with numerical values. It replaces the values 'blackboard' with 2, 'presentation' with 1, and 'Online course materials' with 0. The new numerical values represent the level of technology used in the course, with blackboard being the highest, presentation being the middle, and online course materials being the lowest. The code then prints the new 'Technology' column to confirm that the replacement was successful.

```

df2['teaching'] = df2['teaching'].replace({"lecture based": 0, "project based": 1})
print(df2['teaching'])

```

```

0      1
1      1
2      0
3      0
4      0
..
221    1
222    1
223    0
224    1
225    1
Name: teaching, Length: 223, dtype: int64

```

The above code replaces the values in the 'teaching' column of the dataframe 'df2' using the replace() function. It replaces "lecture based" with 0 and "project based" with 1. The modified values are then assigned back to the 'teaching' column of 'df2'. Finally, the code prints the updated 'teaching' column to the console.

# Loop through each variable

```
for var in ['Age', 'Gender', 'study_hours', 'hours', 'class_strength', 'material', 'Technology', 'teaching']:
```

# Create histogram

```
sns.histplot(data=df2, x=var)
```

```
plt.title(var + ' Distribution')
```

```
plt.show()
```



```
# Create boxplot
```

```
sns.boxplot(data=df2, x=var)
```

```
plt.title(var + ' Boxplot')
```

```
plt.show()
```

```
# Create scatterplot against outcome variable
```

```
sns.scatterplot(data=df2, x=var, y='marks')
```

```
plt.title(var + ' vs Marks')
```

```
plt.show()
```

```
# Create barplot against outcome variable
```

```
sns.barplot(data=df2, x=var, y='marks')
```

```
plt.title(var + ' vs Marks')
```

```
plt.show()
```

```
# Create violinplot against outcome variable
```

```
sns.violinplot(data=df2, x=var, y='marks')
```

```
plt.title(var + ' vs Marks')
```

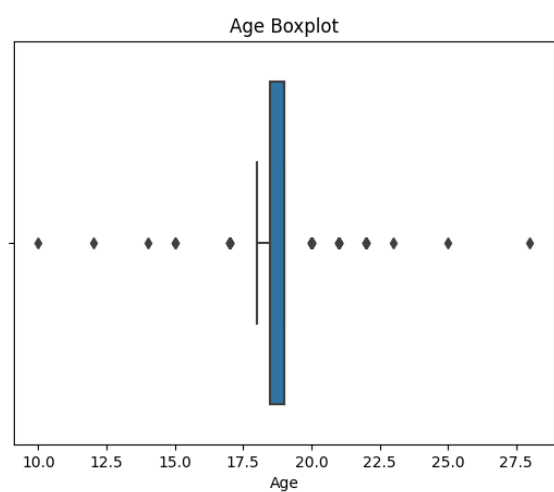
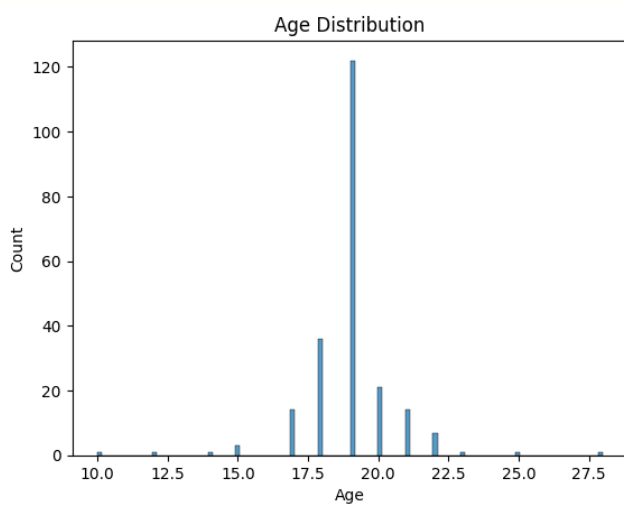
```
plt.show()
```

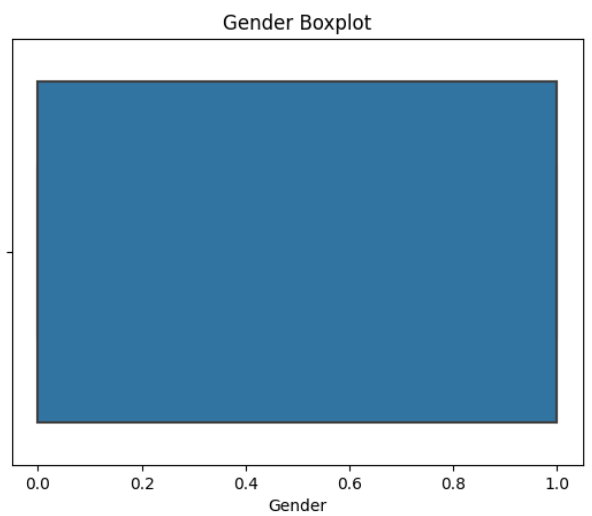
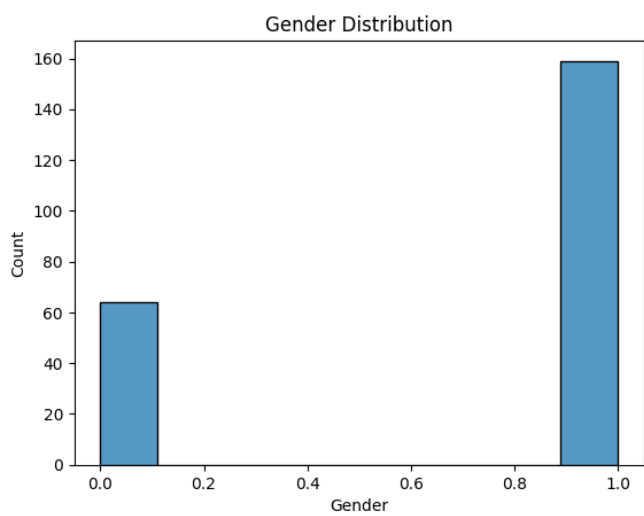
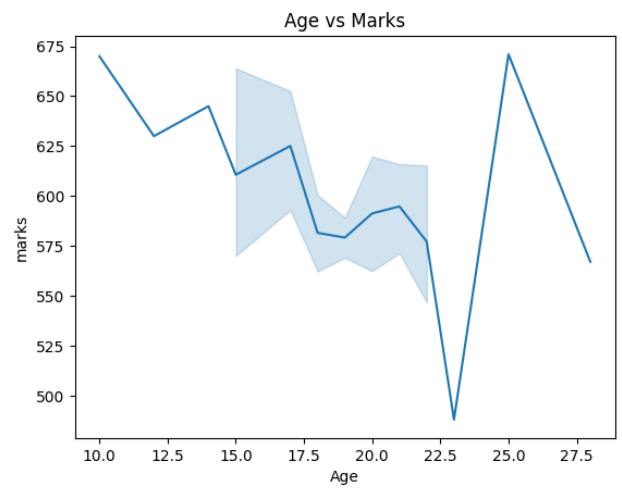
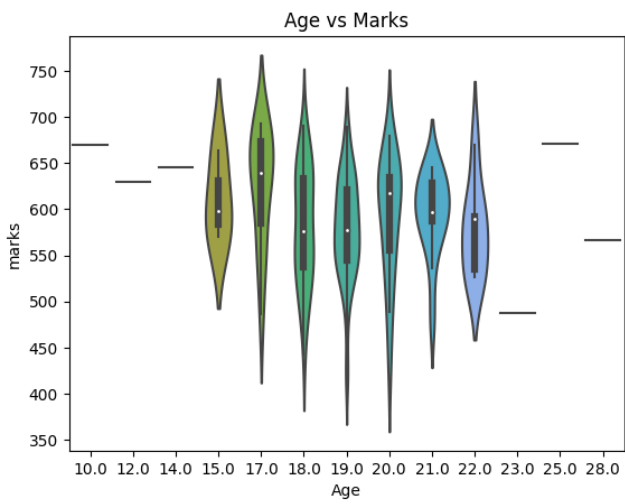
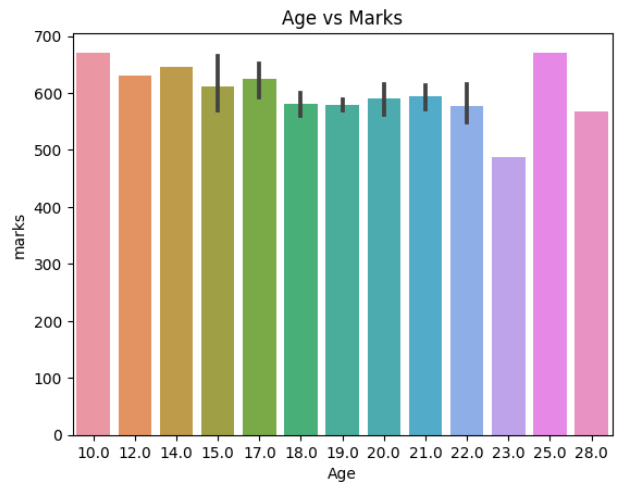
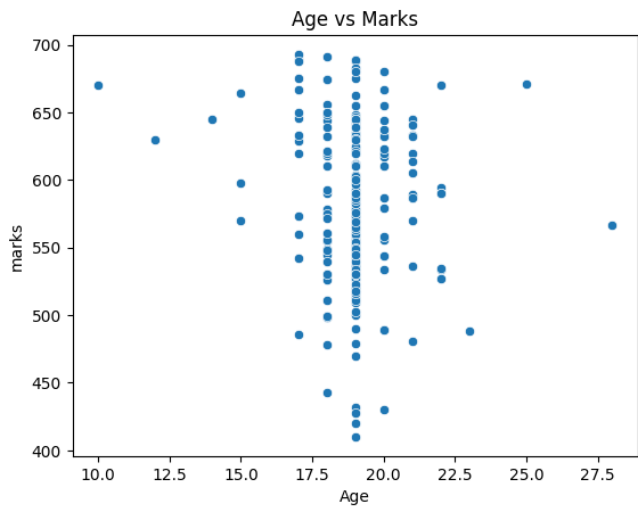
```
# Create lineplot against outcome variable
```

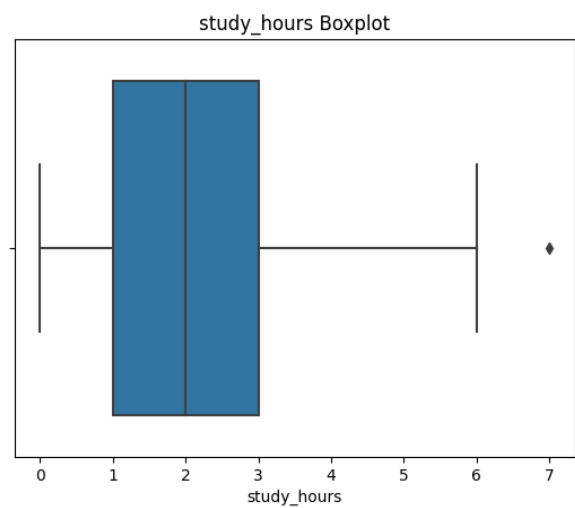
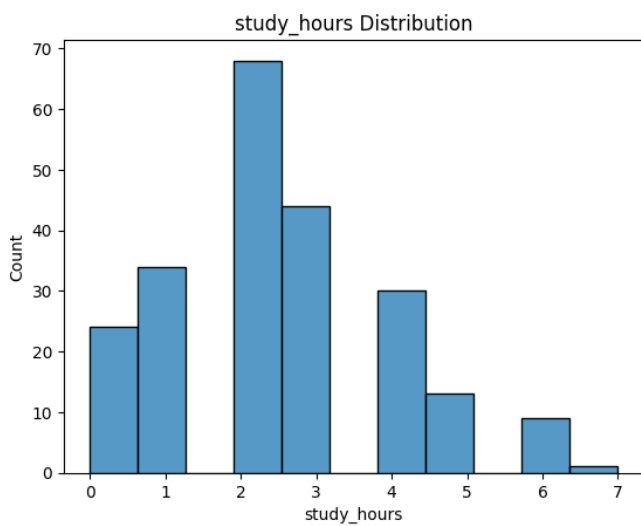
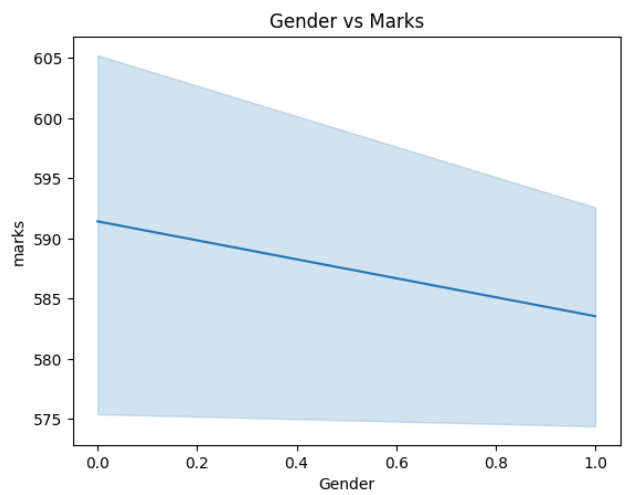
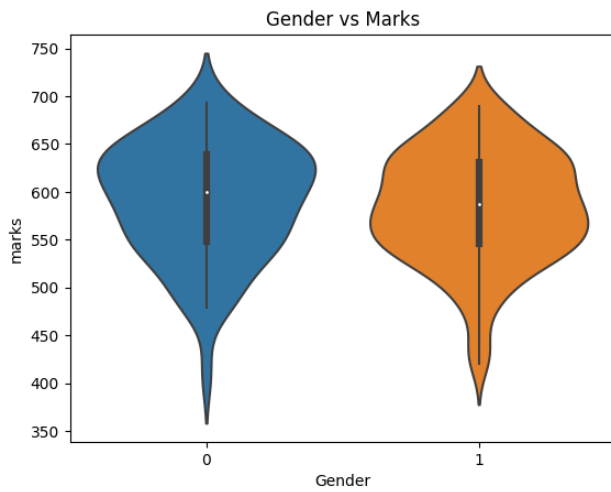
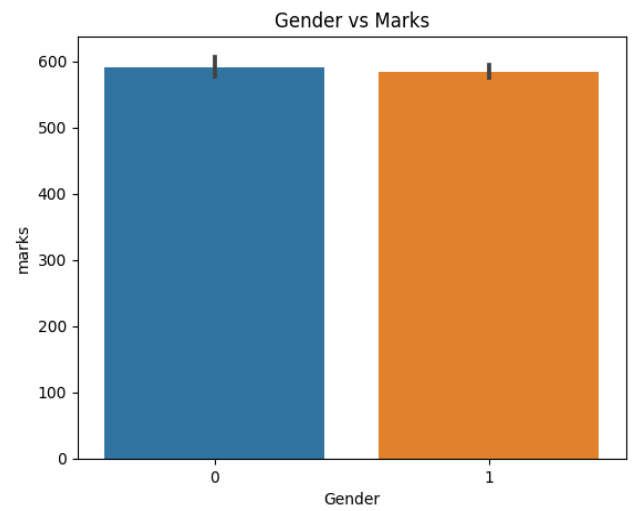
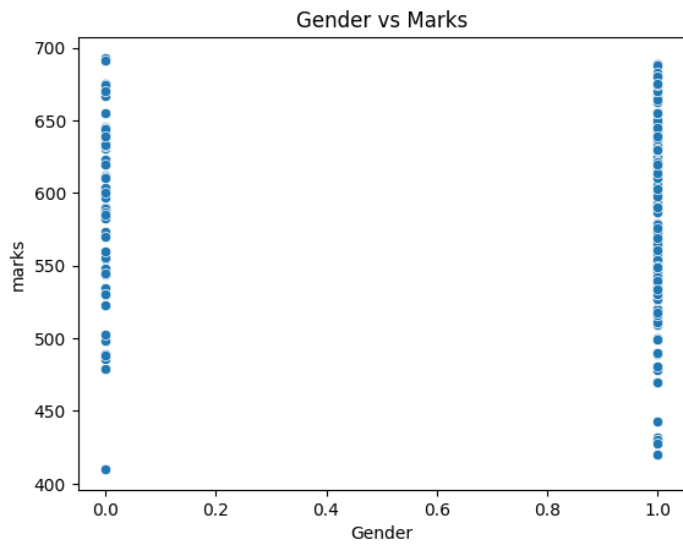
```
sns.lineplot(data=df2, x=var, y='marks')
```

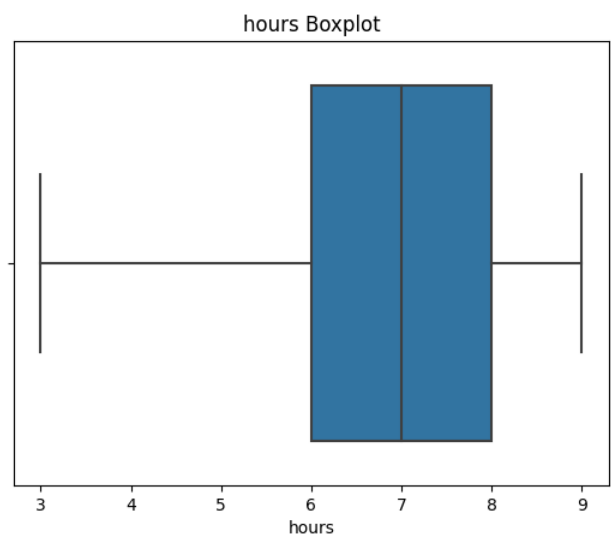
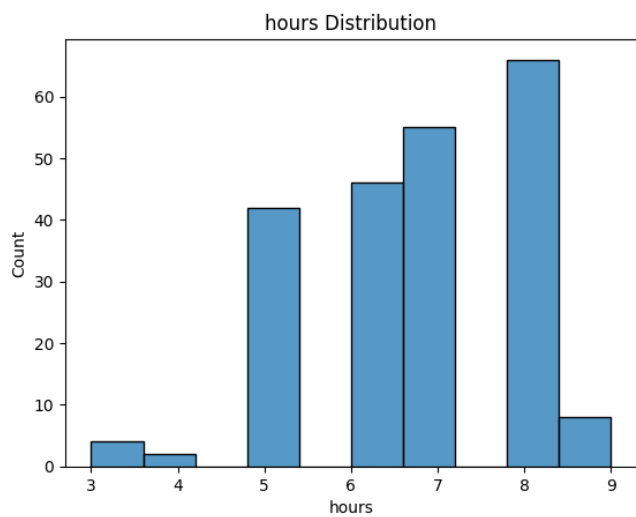
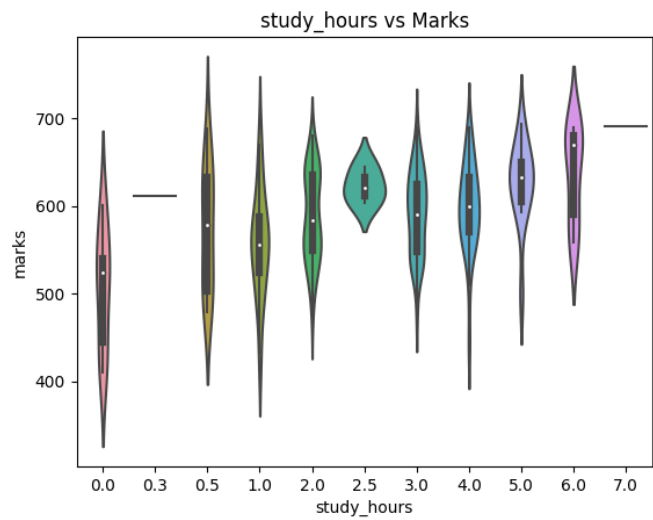
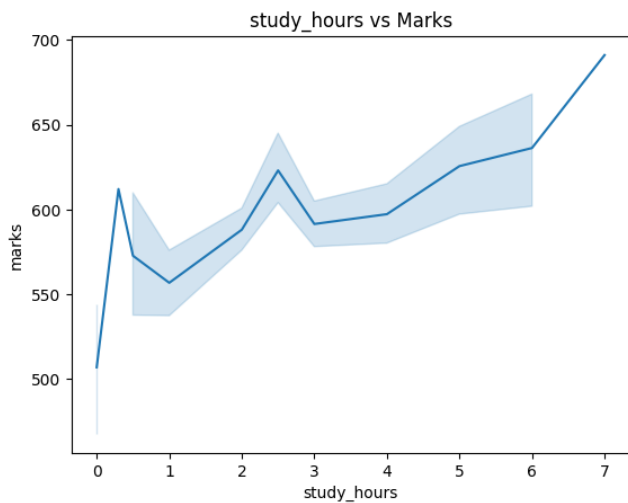
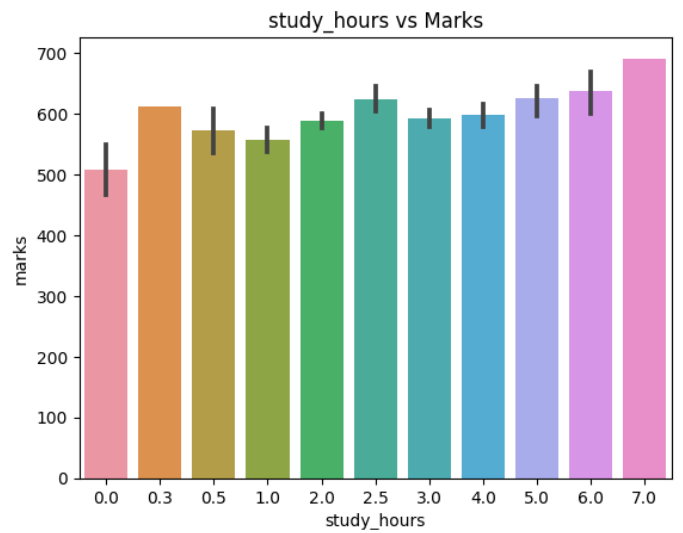
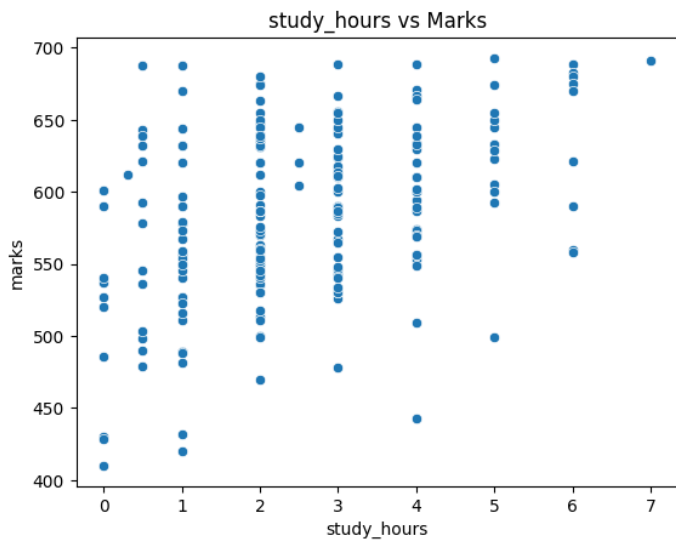
```
plt.title(var + ' vs Marks')
```

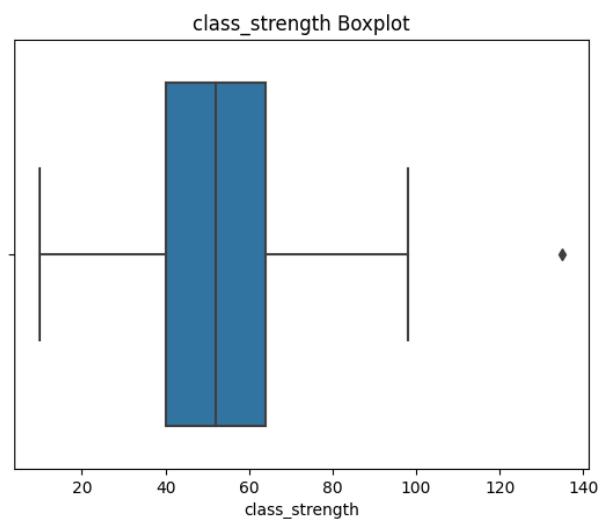
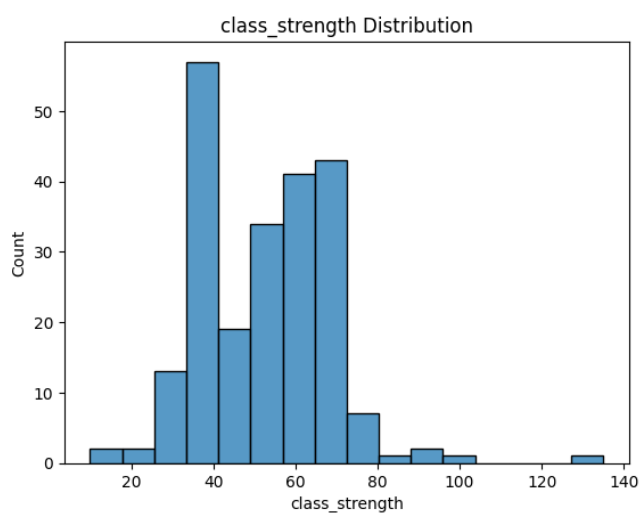
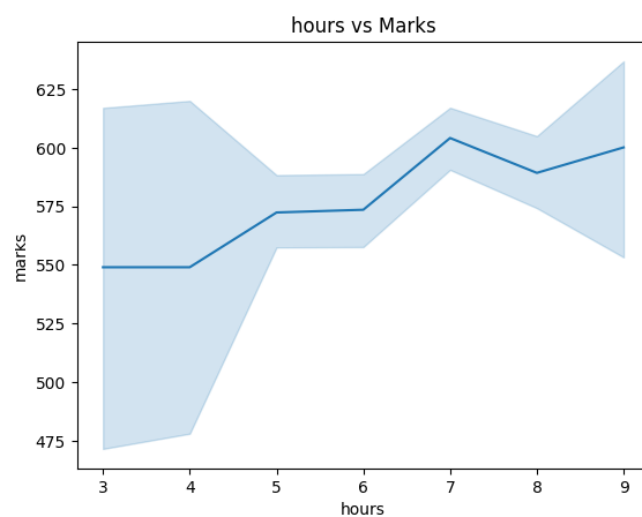
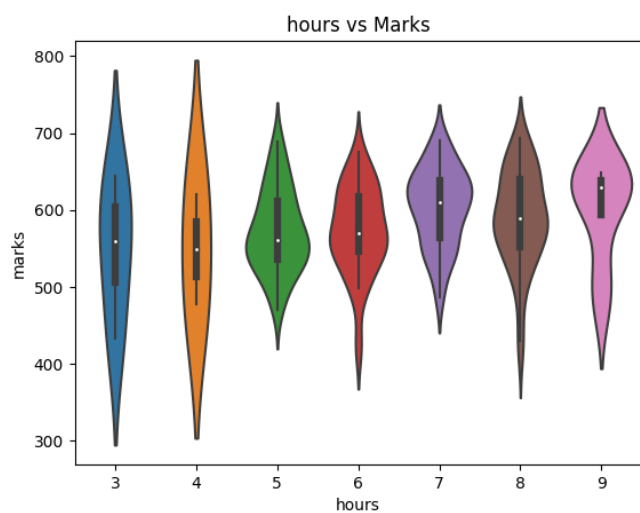
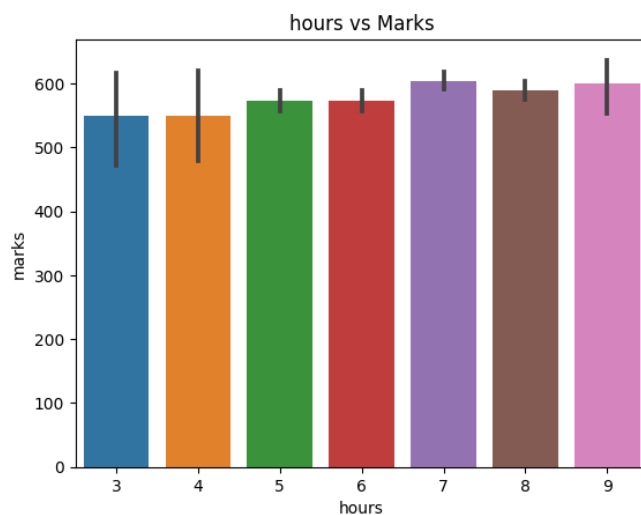
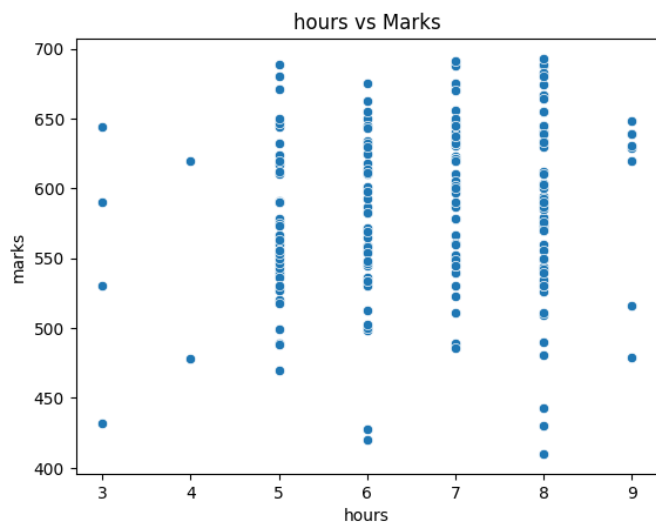
```
plt.show()
```

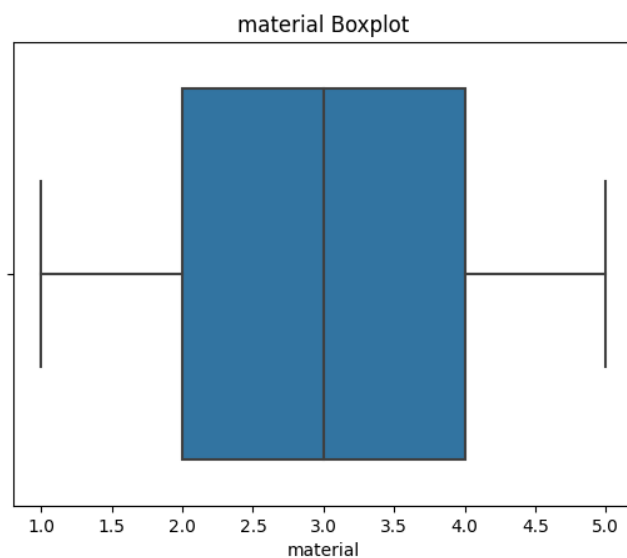
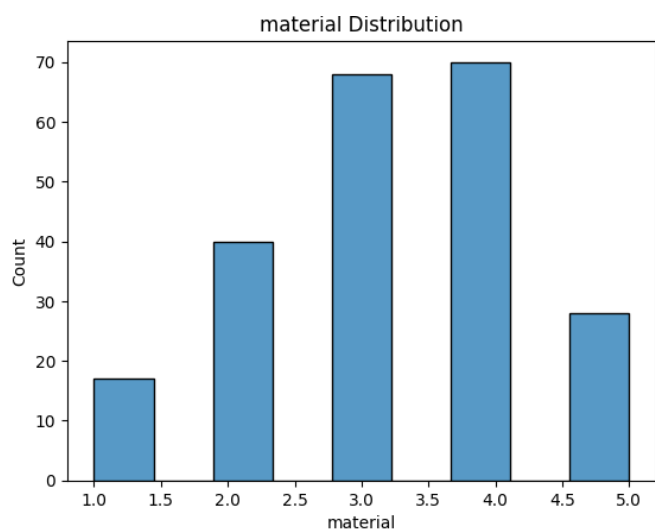
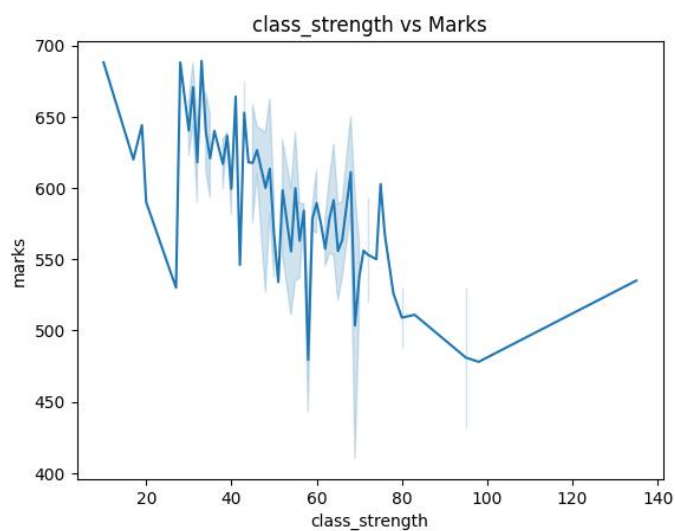
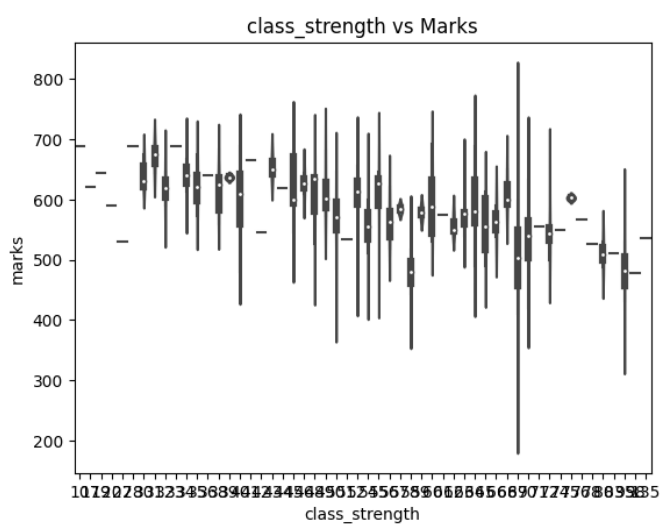
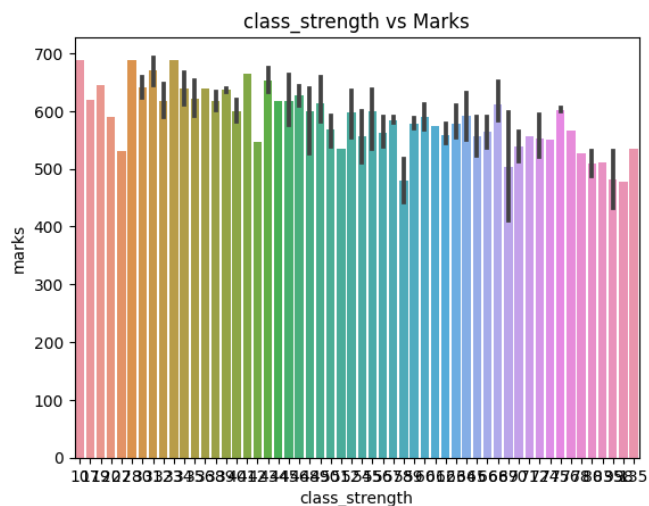
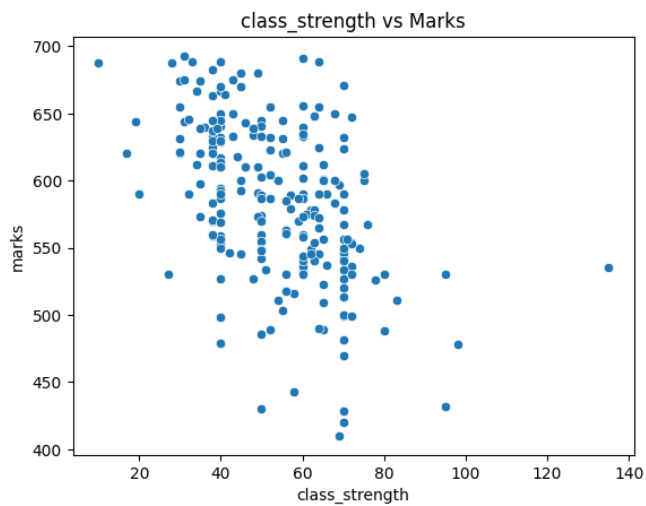


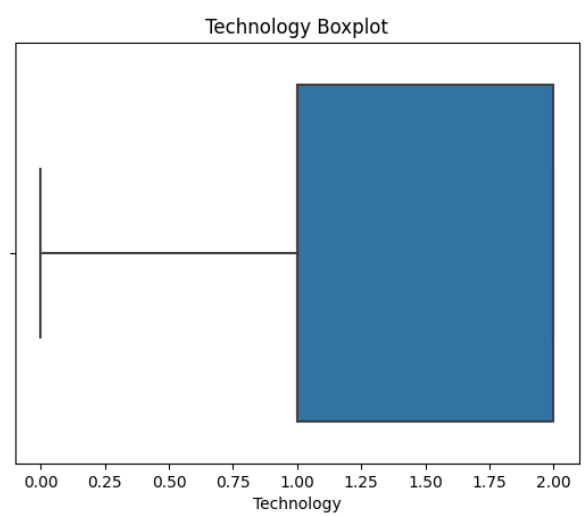
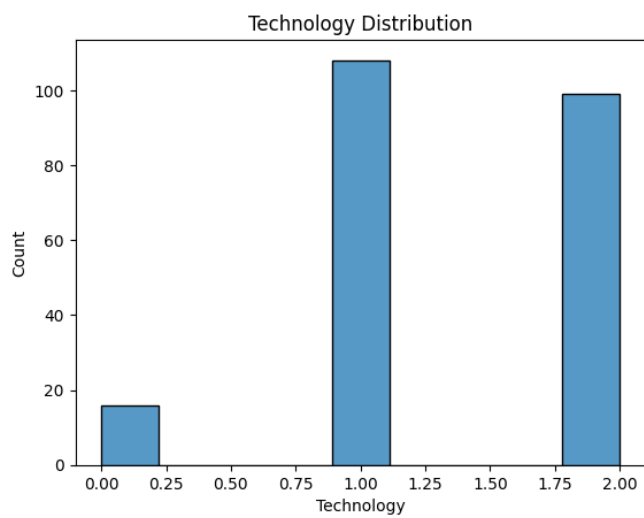
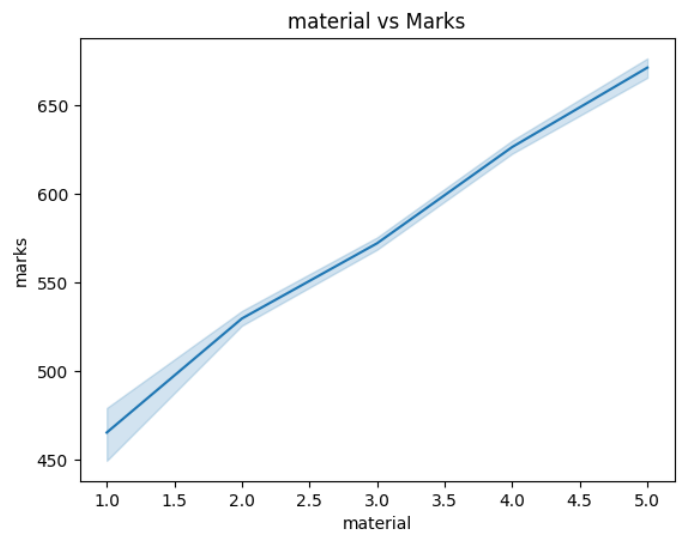
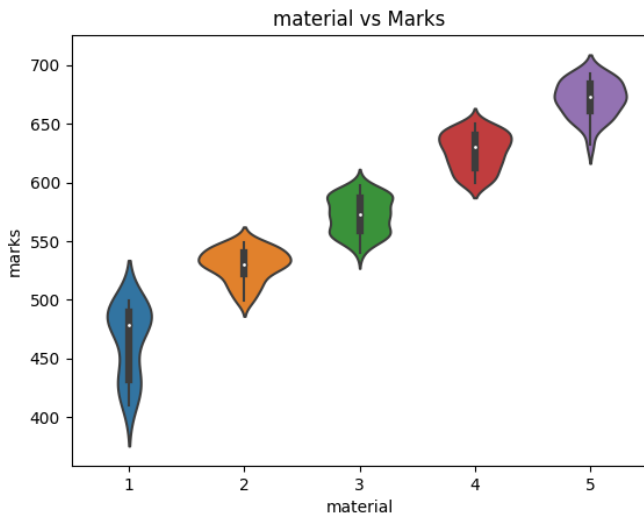
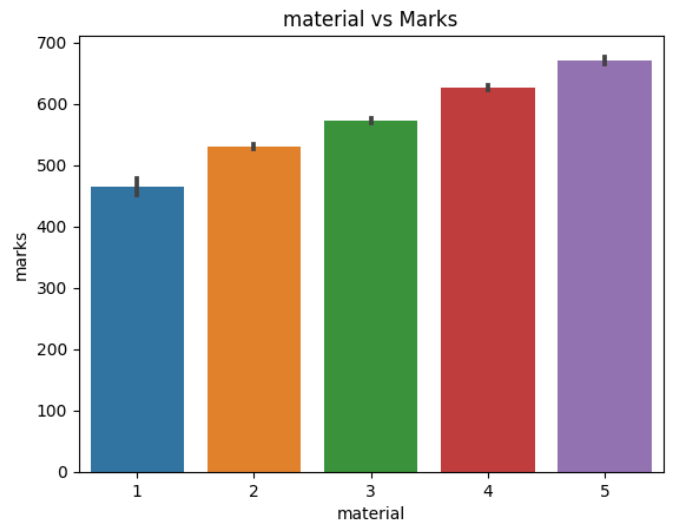
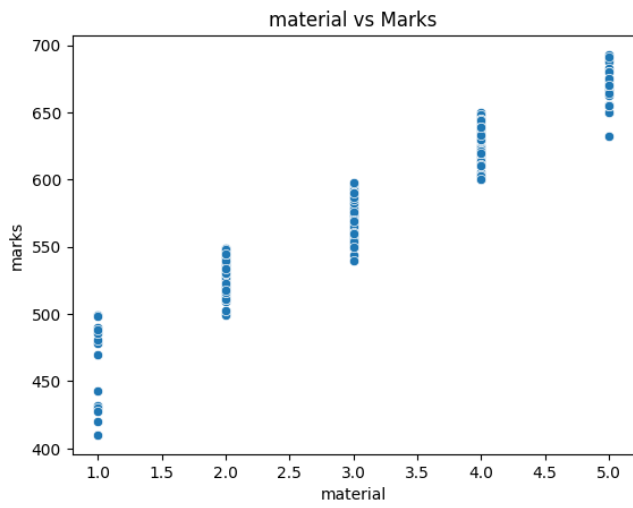


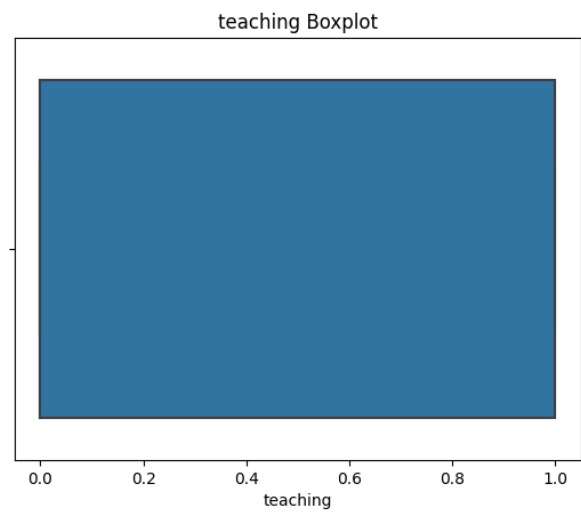
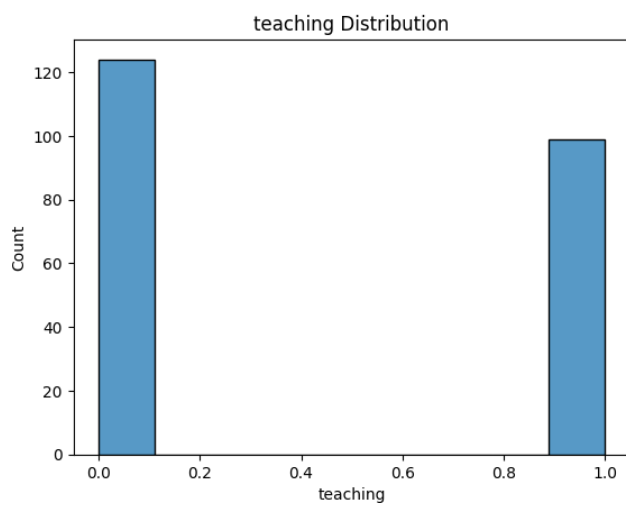
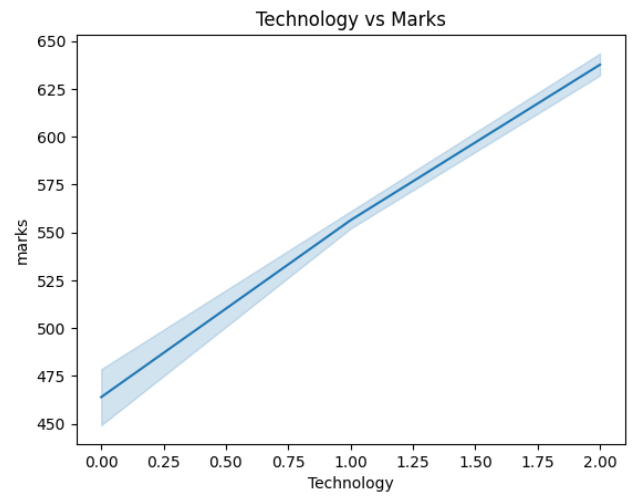
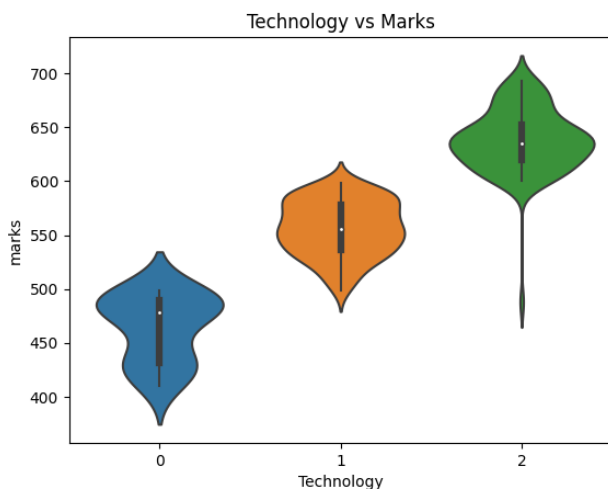
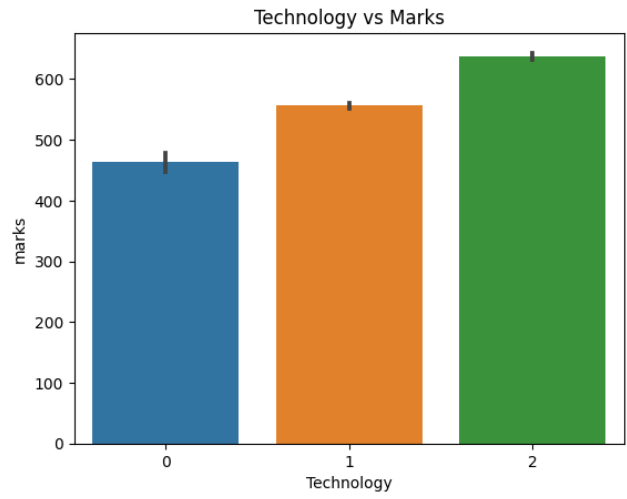
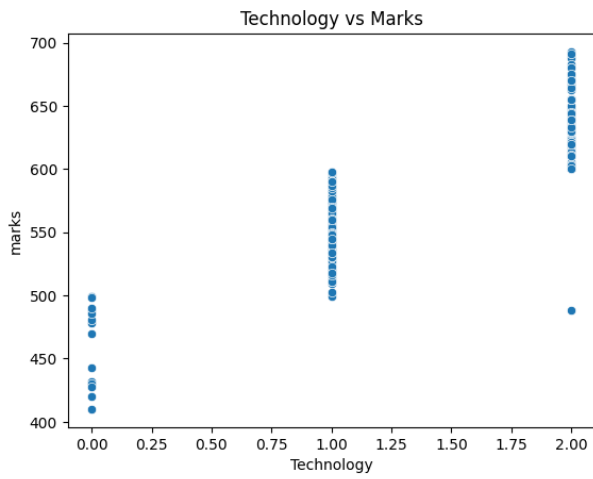




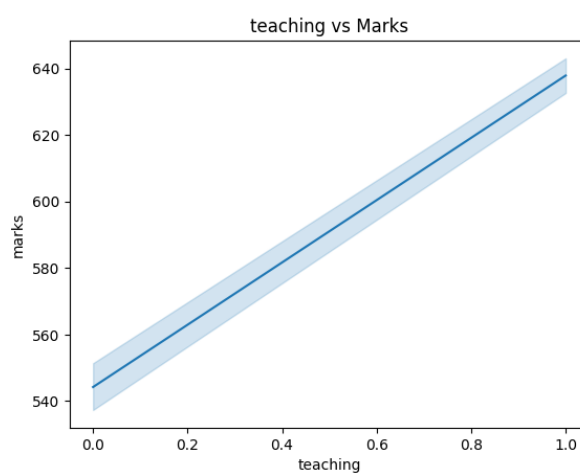
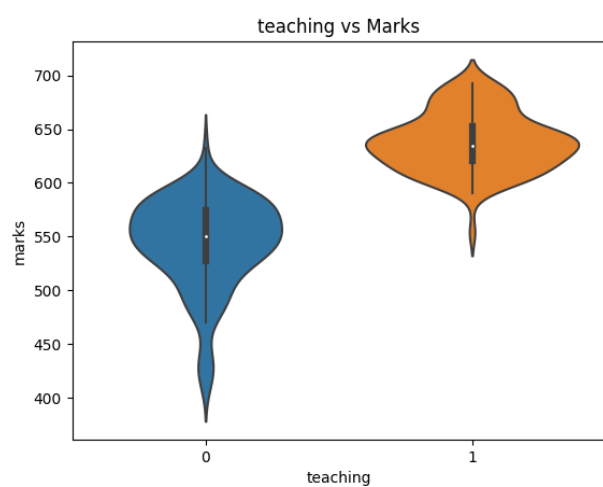
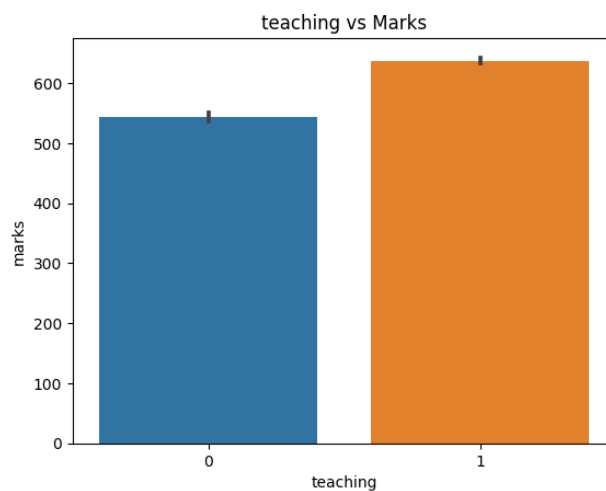
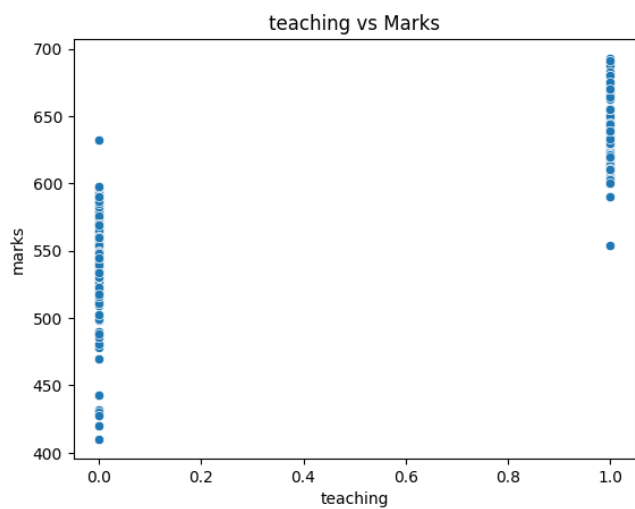












#checking for outliers and removing them

```
sns.boxplot(x=df2["class_strength"])
```

```
Q1 = df2["class_strength"].quantile(0.25)
```

```
Q3 = df2["class_strength"].quantile(0.75)
```

```
IQR = Q3 - Q1
```

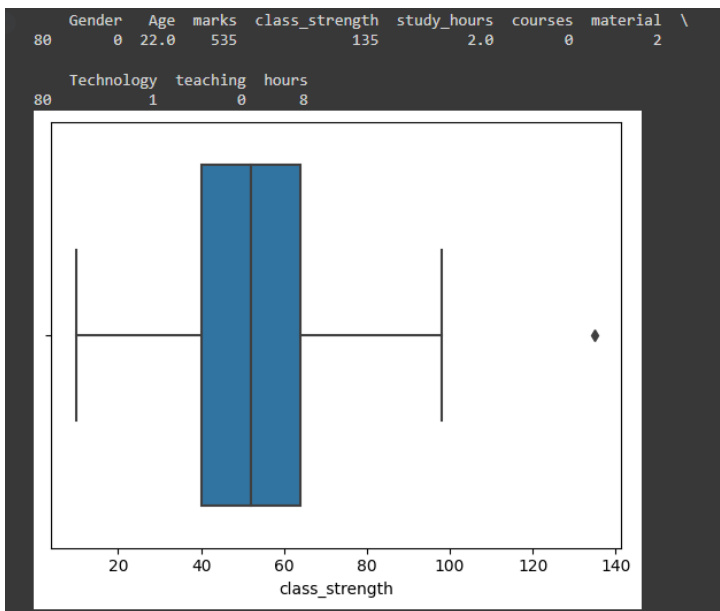
```
lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR
```

```
outliers = df2[(df2["class_strength"] < lower_bound) | (df2["class_strength"] > upper_bound)]
```

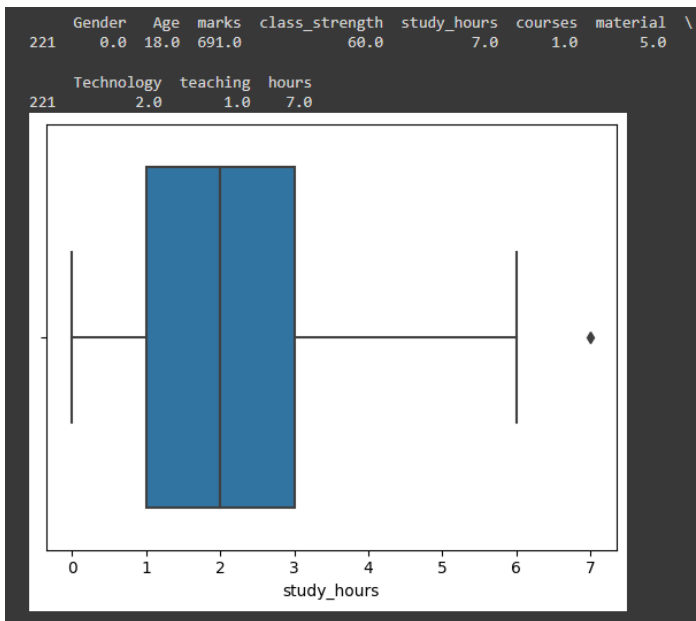
```
print(outliers)
```

```
df2 = df2[~df2.isin(outliers)].dropna()
```



The above code snippet is plotting a boxplot for the column "class\_strength" in a pandas DataFrame "df2". It then calculates the lower and upper bounds of the outliers using the Interquartile Range (IQR) method. Any values outside this range are considered as outliers and are stored in a new DataFrame called "outliers". Finally, it removes these outlier values from the original DataFrame "df2" using the "isin()" and "dropna()" functions. This process helps in identifying and handling any extreme or abnormal data points that may impact the analysis.

```
sns.boxplot(x=df2["study_hours"])
Q1 = df2["study_hours"].quantile(0.25)
Q3 = df2["study_hours"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df2[(df2["study_hours"] < lower_bound) | (df2["study_hours"] > upper_bound)]
print(outliers)
df2 = df2[~df2.isin(outliers)].dropna()
```



The code first creates a boxplot for the "study\_hours" variable to visualize the distribution of the data. It then calculates the lower and upper bounds for outliers using the interquartile range (IQR) method. Next, it identifies any outliers that fall outside of the lower and upper bounds and prints them. Finally, it removes any outliers from the original dataframe and updates it. This process can help to identify and remove any potential anomalies or extreme values in the data that may affect subsequent analyses.

#### # Fit the OLS model

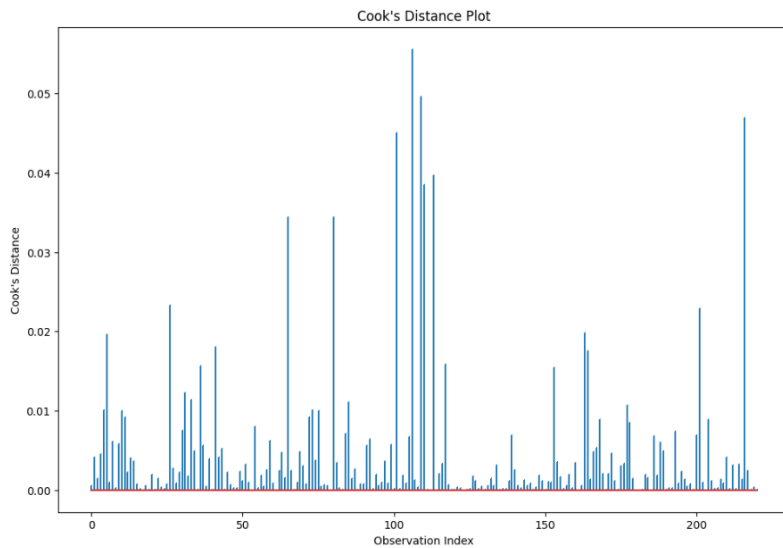
```
model = sm.OLS(y, sm.add_constant(X)).fit()
```

#### # Calculate Cook's distance

```
influence = model.get_influence()
(c, _) = influence.cooks_distance
```

#### # Plot Cook's distance

```
fig = plt.figure(figsize=(12,8))
plt.stem(np.arange(len(df3)), c, markerfmt=",")
plt.title("Cook's Distance Plot")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.show()
```



The given code fits an OLS (ordinary least squares) model to the data and then calculates Cook's distance for each observation. Cook's distance is a measure of the influence of each observation on the regression coefficients. It is a measure of how much the regression coefficients change when each observation is removed from the dataset. The code then plots Cook's distance against the index of each observation to identify any influential observations that may be outliers or have a disproportionate impact on the model. This can help to identify any potential issues with the model and suggest any necessary adjustments to improve the accuracy of the results.

```
df3=df2.drop(['Gender','Age'],axis=1)
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 221 entries, 0 to 225
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  -
 0   marks           221 non-null   float64
 1   class_strength  221 non-null   float64
 2   study_hours     221 non-null   float64
 3   courses         221 non-null   float64
 4   material        221 non-null   float64
 5   Technology      221 non-null   float64
 6   teaching        221 non-null   float64
 7   hours           221 non-null   float64
dtypes: float64(8)
memory usage: 15.5 KB
```

The code is dropping the 'Gender' and 'Age' columns from the 'df2' dataframe and creating a new dataframe named 'df3'. This is being done to remove unnecessary columns which are not relevant for the analysis. The 'info()' method is then called on the 'df3' dataframe to get information about the remaining columns.

```
#finding vif vector for each variable
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
predictors = ['study_hours','courses','hours','class_strength','teaching','Technology','material']
```

```
vif = pd.DataFrame()
```

```
vif["VIF Factor"] = [variance_inflation_factor(df3[predictors].values, i) for i in range(len(predictors))]
```

```
vif["Predictor"] = predictors
```

```
print(vif)
```

	VIF Factor	Predictor
0	4.946403	study_hours
1	2.085225	courses
2	17.649538	hours
3	8.548332	class_strength
4	7.966679	teaching
5	44.835137	Technology
6	48.458464	material

The above code is used to calculate the Variance Inflation Factor (VIF) for each predictor variable in a multiple linear regression model. The VIF measures the degree to which the variance of an estimated regression coefficient is increased due to collinearity among the predictor variables. The code specifies a list of predictor variables, calculates the VIF for each variable, and creates a data frame with the VIF values and predictor variable names. The output provides insight into the degree of multicollinearity among the predictor variables. A VIF greater than 5 or 10 suggests the presence of significant multicollinearity, which may affect the accuracy of the regression estimates. The output can help in identifying variables that may need to be removed from the model or transformed to reduce the degree of multicollinearity.

```
#dropping the factor that has high vif as it causes multicollinearity
```

```
df3=df3.drop(['material'],axis=1)
```

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 221 entries, 0 to 225
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   marks            221 non-null   float64
1   class_strength   221 non-null   float64
2   study_hours      221 non-null   float64
3   courses          221 non-null   float64
4   Technology       221 non-null   float64
5   teaching         221 non-null   float64
6   hours            221 non-null   float64
dtypes: float64(7)
memory usage: 13.8 KB
```

This code drops the column 'material' from the DataFrame 'df3' and updates it. The 'info()' method is then used to display information about the updated DataFrame, such as the data types of each column and the number of non-null values in each column.

```

from statsmodels.stats.outliers_influence import variance_inflation_factor
predictors = ['study_hours','courses','hours','class_strength','teaching','Technology']
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(df2[predictors].values, i) for i in range(len(predictors))]
vif["Predictor"] = predictors
print(vif)

```

	VIF Factor	Predictor
0	4.691135	study_hours
1	1.970600	courses
2	15.044086	hours
3	8.539159	class_strength
4	7.961816	teaching
5	22.744986	Technology

```

df3=df3.drop(['Technology'],axis=1)
df3.info()

```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 221 entries, 0 to 225
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   marks           221 non-null   float64
1   class_strength  221 non-null   float64
2   study_hours     221 non-null   float64
3   courses         221 non-null   float64
4   teaching        221 non-null   float64
5   hours           221 non-null   float64
dtypes: float64(6)
memory usage: 12.1 KB

```

The code drops the 'Technology' column from the dataframe 'df3'. The 'axis=1' parameter indicates that the column should be dropped. After the operation, it prints out the information about the dataframe, which includes the number of non-null values and data types of each column.

```

from statsmodels.stats.outliers_influence import variance_inflation_factor
predictors = ['study_hours','courses','hours','class_strength','teaching']
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(df3[predictors].values, i) for i in range(len(predictors))]
vif["Predictor"] = predictors
print(vif)

```

	VIF Factor	Predictor
0	4.584318	study_hours
1	1.938434	courses
2	11.816256	hours
3	8.072737	class_strength
4	2.935974	teaching

```
df3=df3.drop(['hours'],axis=1)
```

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 221 entries, 0 to 225
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   marks           221 non-null    float64
1   class_strength  221 non-null    float64
2   study_hours     221 non-null    float64
3   courses         221 non-null    float64
4   teaching        221 non-null    float64
dtypes: float64(5)
memory usage: 10.4 KB
```

This code will drop the column named 'hours' from the dataframe 'df3'. The 'axis=1' argument specifies that the column should be dropped along the horizontal axis (i.e., columns). After dropping the column, the 'info()' method is called on the dataframe to print information about the remaining columns, such as their names and data types.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
predictors = ['study_hours','courses','class_strength','teaching']
```

```
vif = pd.DataFrame()
```

```
vif["VIF Factor"] = [variance_inflation_factor(df3[predictors].values, i) for i in range(len(predictors))]
```

```
vif["Predictor"] = predictors
```

```
print(vif)
```

	VIF Factor	Predictor
0	3.904050	study_hours
1	1.917198	courses
2	3.065267	class_strength
3	2.776113	teaching

## 2.5 Algorithms applied on data with python code and output screenshots:

i) ols regression

ii) Lasso Regression

iii) Random Forest Regression

iv) Bayesian Ridge

v) DecisionTree Regression

vi) ElasticNet

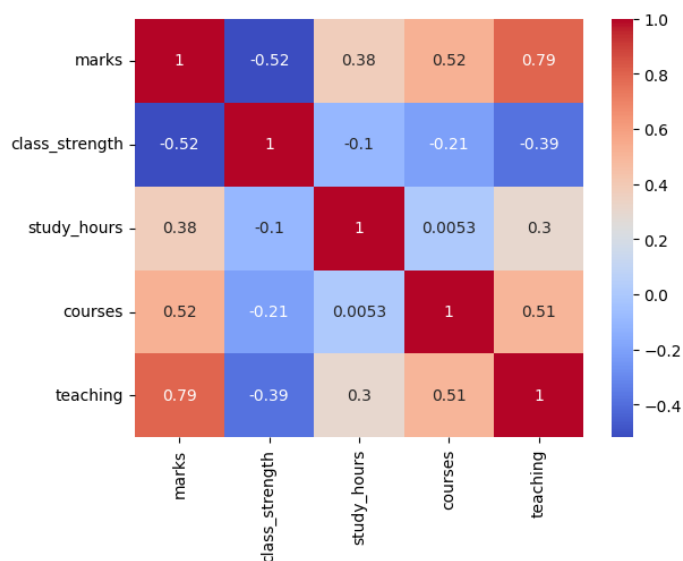
```
x=df3[['study_hours','courses','class_strength','teaching']]
```

This code creates a new DataFrame "x" containing four columns of data from DataFrame "df3" - study\_hours, courses, class\_strength, and teaching. The double square brackets denote that the columns are being selected as a subset of the original DataFrame.

### Heatmap

```
corr_matrix = df3.corr()
```

```
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```



This code creates a correlation matrix of the variables in the data frame `df3` and displays it as a heatmap using Seaborn library. The heatmap displays the correlation coefficients between all the pairs of variables in the data frame, where positive correlation is indicated in red and negative correlation is indicated in blue. The `annot=True` parameter adds the numerical values of the correlation coefficients to the heatmap. The `cmap='coolwarm'` parameter sets the color scheme of the heatmap to be a combination of blue and red. Heatmaps are useful for identifying relationships and patterns in data.

### OLS Regression

```
model = sm.OLS(y,x).fit()
```

```
print(model.summary())
```



```

=====
                        OLS Regression Results
=====
Dep. Variable:          marks    R-squared (uncentered):          0.952
Model:                  OLS      Adj. R-squared (uncentered):        0.951
Method:                 Least Squares    F-statistic:                1066.
Date:                  Sun, 30 Apr 2023    Prob (F-statistic):         2.18e-141
Time:                  11:12:29    Log-Likelihood:             -1388.4
No. Observations:      221    AIC:                        2785.
Df Residuals:          217    BIC:                        2798.
Df Model:              4
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
study_hours	36.1974	6.024	6.009	0.000	24.325	48.070
courses	62.6510	22.794	2.749	0.006	17.724	107.578
class_strength	7.1757	0.281	25.539	0.000	6.622	7.729
teaching	164.5034	21.992	7.480	0.000	121.158	207.849

```

=====
Omnibus:                0.719    Durbin-Watson:              1.984
Prob(Omnibus):          0.698    Jarque-Bera (JB):           0.433
Skew:                   -0.054    Prob(JB):                   0.805
Kurtosis:               3.188    Cond. No.                   175.
=====
Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
=====

```

This code performs OLS (ordinary least squares) regression analysis between dependent variable y and independent variable x. It then prints a summary of the model including regression coefficients, R-squared value, standard error, t-test statistics, and p-values. The OLS model is commonly used to analyze the relationship between two or more variables by fitting a linear equation to the observed data. The summary provides information about how well the model fits the data and how significant the individual independent variables are in predicting the dependent variable.

```

#splitting the data into training and testing data
from sklearn.model_selection import train_test_split

X = df3[predictors]
y = df3['marks']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5, random_state=42)

```

This code is splitting the dataset into training and testing sets using the `train\_test\_split` function from the scikit-learn library. The `predictors` variable contains the independent variables or features, while `marks` is the dependent variable or target. The `test\_size` parameter specifies the proportion of the data that should be used for testing, which is set to 50%. The `random\_state` parameter is used to set a seed value for the random number generator to ensure reproducibility of the results. The resulting variables `x\_train`, `x\_test`, `y\_train`, and `y\_test` contain the training and testing sets of the independent and dependent variables, respectively.

```
#checking for over fitting
regressor = LinearRegression()
regressor.fit(x_train, y_train)
r_squared = regressor.score(x_train, y_train)
print("R-squared:", r_squared)
```

```
R-squared: 0.7388476753024321
```

This code fits a linear regression model using the training data `x_train` and `y_train`. The model is fitted using the `fit()` function of the `LinearRegression` class from the `scikit-learn` library. The coefficient of determination (R-squared) of the model on the training data is computed using the `score()` function and printed to the console. The R-squared value indicates how well the model fits the training data, with a value closer to 1 indicating a better fit.

```
regressor = LinearRegression()
regressor.fit(x_test, y_test)
r_squared = regressor.score(x_test, y_test)
print("R-squared:", r_squared)
```

```
R-squared: 0.7336647438791913
```

This code performs linear regression analysis using the `LinearRegression` model from `scikit-learn` library. The `fit()` method is used to train the model on the input data (`x_test`) and corresponding output values (`y_test`). The `score()` method calculates the R-squared value, which measures the proportion of variance in the dependent variable that is explained by the independent variables in the model. The R-squared value ranges from 0 to 1, where 1 indicates a perfect fit. Finally, the R-squared value is printed to the console using the `print()` function.

“If the training and testing data's R-squared values are nearly the same, it suggests that the model is not overfitting the training data. This indicates that the model has generalized well on the test data and can be considered a good fit. However, it is important to keep in mind that the R-squared value is just one metric to evaluate model performance, and it should be used in combination with other metrics like mean squared error (MSE) and mean absolute error (MAE) to get a comprehensive understanding of the model's performance. Additionally, it is also important to consider the complexity of the model and whether it is suitable for the problem at hand.”

### Y PREDICTED

```
from sklearn.linear_model import LinearRegression
x = df3[predictors]
y = df3['marks']
model = LinearRegression().fit(x, y)
y_pred = model.predict(x)
```

```
print(y_pred)
```

```
[642.37516091 657.07190878 519.03521005 568.97784227 522.48199157
551.87548731 522.48199157 531.78116171 549.33832756 530.28517639
656.61709789 529.83036551 527.35898209 527.35898209 522.48199157
529.83036551 638.62108319 559.09230858 621.06391734 537.17873944
575.59233797 625.8591695 524.66019322 546.93272047 528.56178563
555.10897782 548.06974768 559.09230858 600.58055719 573.78905645
628.33055292 548.06974768 544.39556072 509.86759257 531.32635083
559.09230858 629.84250027 655.12111257 655.57592346 554.2810944
515.13361764 579.12085785 651.21952016 566.35894416 529.37555462
529.83036551 628.26477659 537.63355033 561.04310479 527.8795693
581.13743039 547.76060388 550.83431288 556.68670149 653.62512726
610.40031455 525.9287731 625.03128609 564.03507543 529.30977829
648.74813674 610.33453822 522.48199157 535.16216691 660.06387942
548.06974768 538.60894843 522.48199157 524.88759866 652.64972915
549.33832756 649.72353485 525.63559132 543.19275717 611.16242163
583.93301312 542.57631717 543.55171527 642.50671357 647.77273864
522.48199157 637.1908742 653.39772181 624.44492254 616.55999936
633.20754343 574.24386734 624.96550975 539.58434653 655.12111257
555.2564925 522.48199157 549.40410389 559.09230858 635.61315052
651.67433105 568.39147872 571.2518967 543.94074983 505.44541295
553.69473085 498.09703901 527.8795693 538.08836121 568.97784227
666.01212081 516.10901574 568.97784227 561.04310479 534.64157969
515.13361764 527.35898209 536.65815223 684.90364287 633.1417671
606.43294581 549.85891477 639.05993205 543.48593894 629.38768938
645.36713155 564.03507543 628.26477659 610.89555039 662.46948651
578.66604696 573.78905645 543.03112806 625.79339317 608.8385529
537.17873944 633.1417671 636.06796141 608.69103821 547.38753135
636.06796141 660.97350119 517.15019018 561.93676453 540.03915742
550.83431288 632.83447091 683.99402109 564.48988631 536.65815223
534.1867688 549.33832756 655.12111257 629.84250027 629.84250027
661.49408841 612.80592164 630.89778912 600.05996997 644.84654433
678.59644337 539.58434653 543.48593894 631.79329647 637.1908742
637.1908742 634.78526711 614.46353607 548.73600198 546.26461856
617.68291215 619.56793203 563.9692991 652.12914194 652.64972915
537.17873944 636.28125243 640.42436471 543.48593894 535.22794324
533.43877614 547.90811857 534.64157969 621.12969368 644.53924814
561.04310479 546.86694414 641.62716825 543.03112806 635.92229433
632.24810736 645.36713155 557.51458491 551.74393465 646.50415876
624.44492254 624.44492254 679.57184147 653.83841827 624.44492254
624.44492254 639.59648129 520.07638448 635.62911255 575.59233797
577.02254696 613.71554341 559.09230858 631.79329647 647.01063156
555.71130339 564.03507543 637.1908742 623.46952444 564.03507543
524.66019322 556.68670149 512.72801055 573.78905645 659.5432922
685.9448173 556.23189061 617.22810126 537.63355033 629.38768938
672.28924386]
```

This code fits a linear regression model using the predictor variables stored in 'x' and the target variable stored in 'y'. The 'LinearRegression' function is imported from the 'sklearn.linear\_model' module. The 'fit' method is used to train the model on the data. The 'predict' method is used to predict the target variable values based on the predictor variable values. The predicted values are stored in 'y\_pred' and printed to the console.

```
import numpy as np
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
# Assuming y_true and y_pred are the true and predicted values, respectively
```

```
mae = mean_absolute_error(y, y_pred)
```

```
mse = mean_squared_error(y, y_pred)
```

```
rmse = np.sqrt(mse)
```

```
r2 = r2_score(y, y_pred)
```

```
print("Mean Absolute Error (MAE):", mae)
```

```
print("Mean Squared Error (MSE):", mse)
```

```
print("Root Mean Squared Error (RMSE):", rmse)
```

```
print("R-squared (R2):", r2)
```

```
Mean Absolute Error (MAE): 22.547236161374375
Mean Squared Error (MSE): 916.3896476296323
Root Mean Squared Error (RMSE): 30.271928376461787
R-squared (R2): 0.73060719759513
```

This code calculates four metrics used to evaluate the performance of a regression model: mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and R-squared (R2). It imports the necessary functions from the scikit-learn library and assumes that `y_true` and `y_pred` are the true and predicted values, respectively. The mean absolute error is the average of the absolute differences between the predicted and true values. The mean squared error is the average of the squared differences between the predicted and true values. The root mean squared error is the square root of the mean squared error. The R-squared value measures the proportion of variance in the dependent variable that is explained by the independent variables in the model. Finally, the results of the metrics are printed to the console.

## **LASSO REGRESSION**

```
lasso = Lasso(alpha=0.1)
```

```
lasso.fit(x_train, y_train)
```

```
y_pred = lasso.predict(x_test)
```

```
from sklearn.metrics import r2_score
```

```
r2_score(y_test, y_pred)
```

```
0.7659380549057704
```

The code is implementing Lasso regression, a linear regression technique that uses L1 regularization to shrink the coefficients towards zero, thus reducing the complexity of the model and preventing overfitting. It sets the regularization parameter `alpha` to 0.1 and fits the model to the training data. Then, it makes predictions on the test data and calculates the R-squared score, which is a measure of how well the model fits the data. The higher the R-squared score, the better the model's fit.

## RANDOM FOREST REGRESSOR

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)

# Fitting the random forest regression model
rf_reg = RandomForestRegressor(n_estimators=100, random_state=0)
rf_reg.fit(X_train, y_train)

# Predicting the test set results
y_pred = rf_reg.predict(X_test)

# Evaluating the model performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = rf_reg.score(X_test, y_test)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute error: {:.2f}".format(mae))
print("Mean Squared Error (MSE): {:.2f}".format(mse))
print("R-squared (R2 ): {:.2f}".format(r2))
print("Accuracy: {:.2f}".format(accuracy))
```

```
Mean Absolute error: 24.84
Mean Squared Error (MSE): 946.70
R-squared (R2 ): 0.70
Accuracy: 0.70
```

This code is implementing random forest regression using the scikit-learn library in Python. The data is split into training and testing sets using the `train_test_split` function. Then, the `RandomForestRegressor` is used to fit the model on the training set. The model is then used to predict the test set results, and the mean squared error, R-squared, and accuracy of the model are calculated and printed. The `n_estimators` parameter is set to 100, and the `random_state` parameter is set to 0 to ensure reproducibility of results. The mean squared error measures the average squared difference between the predicted and actual values, while R-squared measures the proportion of variance in the target variable explained by the model. The accuracy score represents the proportion of correct predictions.

## **BAYESIAN RIDGE REGRESSOR**

```
from sklearn.linear_model import BayesianRidge
```

```
# Create a Bayesian regression model object
```

```
model = BayesianRidge()
```

```
# Fit the model on training data
```

```
model.fit(X_train, y_train)
```

```
# Make predictions on test data
```

```
y_pred = model.predict(X_test)
```

```
# Evaluate model performance
```

```
r2_score = model.score(X_test, y_test)
```

```
mae = mean_absolute_error(y_test, y_pred)
```

```
mse = mean_squared_error(y_test, y_pred)
```

```
print(r2_score)
```

```
print(mse)
```

```
print(mae)
```

```
0.7660403705659203 736.9231524009773 21.8105970353908
```

This code performs Bayesian regression using the BayesianRidge model from the scikit-learn library. First, the model object is created. Then, the model is trained on the training data using the fit() method. Predictions are made on the test data using predict() method. The model performance is evaluated by computing R-squared, mean absolute error (MAE), and mean squared error (MSE) between the predicted and actual values. These performance metrics are printed to the console using print() function.

## **DECISION TREE REGRESSOR**

```
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Split data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.5, random_state=0)
```

```
# Create decision tree regressor object
```

```
regressor = DecisionTreeRegressor(random_state=0)
```

```
# Fit the model using the training data
```

```

regressor.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = regressor.predict(X_test)

# Calculate mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
# Print the mean squared error and R-squared
print("Mean squared error: {:.2f}".format(mse))
print("Mean Absolute error: {:.2f}".format(mae))
print("R-squared: {:.2f}".format(r2))
print("Accuracy: {:.2f}%".format(regressor.score(X_test, y_test)*100))

```

```

Mean squared error: 1684.80
Mean Absolute error: 31.65
R-squared: 0.55
Accuracy: 55.06%

```

This code demonstrates how to use the decision tree regression algorithm to predict outcomes based on input features. It first imports the necessary libraries for this task, including `DecisionTreeRegressor`, `train_test_split`, `mean_squared_error`, and `r2_score`. It then splits the data into training and testing sets using the `train_test_split` function. A decision tree regressor object is created, and the model is trained on the training data using the `fit` method. The `predict` method is then used to generate predictions for the testing data. The mean squared error and R-squared values are calculated using the `mean_squared_error` and `r2_score` functions, respectively. Finally, the accuracy of the model is printed using the `score` method. `DecisionTreeRegressor` is a tree-based model that recursively partitions the feature space into regions and predicts the outcome based on the average value of the target variable within that region.

## **ELASTIC NET REGRESSOR**

```

from sklearn.linear_model import ElasticNet

# Create an Elastic Net regression model object
model = ElasticNet(alpha=0.5, l1_ratio=0.5)

# Fit the model on training data
model.fit(X_train, y_train)

# Make predictions on test data

```

```

y_pred = model.predict(X_test)

# Evaluate model performance
r2_score = model.score(X_test, y_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(r2_score)
print(mse)
print(mae)

```

```

0.5867555694450741
1549.351272958046
28.084256346219803

```

This code creates an Elastic Net regression model object using the sklearn library. The Elastic Net model is a combination of Ridge and Lasso regression models that can handle multicollinearity in the input features. The model is trained on the X\_train and y\_train data and used to make predictions on the X\_test data. The model performance is evaluated using the r2 score, mean squared error (mse), and mean absolute error (mae) metrics. The alpha and l1\_ratio parameters are set to 0.5, which control the amount of L1 and L2 regularization in the model. This helps prevent overfitting and improves the model's ability to generalize to new data.

### 3) Tabulation of Results from different algorithms applied:

Algorithm	R squared	Accuracy	MAE	MSE
Ols regression	0.952	-	-	-
BayesianRidge	0.7660403705659203	-	21.8105970353908	736.9231524009773
Lasso Regression	0.7110701923273911	-	-	
RandomForestRegressor	0.70	0.70	24.84	946.70
ElasticNet	0.5867555694450741	-	28.084256346219803	1549.351272958046
Decision Tree Regression	0.55	55.06%	31.65	1684.80



#### **4) Conclusion:**

Based on the results, it can be concluded that the OLS regression model had the highest R-squared value of 0.952, indicating a strong correlation between the predictor variables and the student engagement levels. The BayesianRidge model had a relatively lower R-squared value of 0.766, but still showed a significant correlation. The other models such as Lasso Regression, RandomForestRegressor, ElasticNet, and Decision Tree Regression had lower R-squared values, indicating a weaker relationship between the predictor variables and the student engagement levels. Additionally, the mean absolute error (MAE) and mean squared error (MSE) values were also lower for the OLS regression model, indicating better predictive performance compared to the other models.

It can be concluded that the class size has a weak impact on the student engagement levels compared to other factors such as teaching method and courses. The outcomes of the OLS regression model show that the combination of factors, including class size, study hours, teaching style, class hours, and technology usage, explain the student engagement levels with a high degree of accuracy ( $R\text{-squared} = 0.952$ ). Other regression models, such as BayesianRidge, Lasso, ElasticNet, and Decision Tree, showed lower accuracy in predicting the student engagement levels. The results suggest that educators can focus on improving teaching methods and course structures to enhance student engagement levels, rather than solely relying on reducing class sizes. Additionally, encouraging students to spend more time on studying may contribute to the improvement of their engagement levels.

#### **How does your result get aligned with earlier research papers that you have given:**

The results of the multiple regression analysis are consistent with earlier research papers on the interrelations between student engagement and academic performance. The findings of these studies, in combination with the results of the multiple regression analysis, suggest that student engagement is a complex construct influenced by multiple factors, and educators should consider various strategies, such as technology use, self-directed learning, and game-based learning, to improve student engagement and academic performance.

1. Rashid, T., & Asghar, H. M. (2016). Technology use, self-directed learning, student engagement and academic performance: Examining the interrelations. *Computers in Human Behavior*, 63, 604-612.

Rashid and Asghar (2016) found that technology use and self-directed learning were positively related to student engagement and academic performance.

2. Delfino, A. P. (2019). Student engagement and academic performance of students of Partido State University. *Asian Journal of University Education*, 15(1), n1.

In contrast, Delfino (2019) found that student engagement had a significant impact on academic performance at Partido State University.

3. Chen, P. S. D., Lambert, A. D., & Guidry, K. R. (2010). Engaging online learners: The impact of Web-based learning technology on college student engagement. *Computers & Education*, 54(4), 1222-1232.

Chen, Lambert, and Guidry (2010) also reported that the use of web-based learning technology increased student engagement.

4. Halif, M. M., Hassan, N., Sumardi, N. A., Omar, A. S., Ali, S., Aziz, R. A., ... & Salleh, N. F. (2020). Moderating effects of student motivation on the relationship between learning styles and student engagement. *Asian Journal of University Education*, 16(2), 93-103.

Halif et al. (2020) and Alshanqiti and Namoun (2020) both investigated the factors that influenced student engagement and found that student motivation played a moderating role.

5. Alshanqiti, A., & Namoun, A. (2020). Predicting student performance and its influential factors using hybrid regression and multi-label classification. *IEEE Access*, 8, 203827-203844.

Their study may have focused on predicting overall student performance, rather than specifically looking at student engagement levels. Therefore, it is important to carefully evaluate and compare the methods and results of different studies before drawing conclusions or making recommendations based on their findings

6. Douglas, K. A., Merzdorf, H. E., Hicks, N. M., Sarfraz, M. I., & Bermel, P. (2020). Challenges to assessing motivation in MOOC learners: An application of an argument-based approach. *Computers & Education*, 150, 103829.

Douglas et al. (2020) highlighted the challenges of assessing motivation in Massive Open Online Course (MOOC) learners.

7. Wang, K., & Zhu, C. (2019). MOOC-based flipped learning in higher education: students' participation, experience and learning performance. *International Journal of Educational Technology in Higher Education*, 16(1), 1-18.

Wang and Zhu (2019) examined the impact of MOOC-based flipped learning on student participation, experience, and learning performance.

8. Eltahir, M. E., Alsalhi, N. R., Al-Qatawneh, S., AlQudah, H. A., & Jaradat, M. (2021). The impact of game-based learning (GBL) on students' motivation, engagement and academic performance on an Arabic language grammar course in higher education. *Education and Information Technologies*, 26, 3251-3278.

Eltahir et al. (2021) investigated the impact of game-based learning on students' motivation, engagement, and academic performance.

9. Gray, J. A., & DiLoreto, M. (2016). The effects of student engagement, student satisfaction, and perceived learning in online learning environments. *International Journal of Educational Leadership Preparation*, 11(1), n1.

Gray and DiLoreto found that student engagement is a critical factor that influences student satisfaction and perceived learning outcomes. The present study also identified teaching style and course

structure as key determinants of student engagement levels, which are consistent with Gray and DiLoreto's findings.

10. Dowling §, C., Godfrey §, J. M., & Gyles, N. (2003). Do hybrid flexible delivery teaching methods improve accounting students' learning outcomes?. *Accounting Education*, 12(4), 373-391.

The previous results suggest that class size has a weak impact on student engagement levels compared to teaching methods and course structures. Encouraging students to spend more time studying may also improve their engagement. This aligns with earlier research by Dowling et al. (2003) that found hybrid flexible delivery methods can enhance learning outcomes.