University of HUDDERSFIELD

Inspiring tomorrow's professionals

Prediction of students' performance using random forest regression

Module: Data Mining

Module leader : Dr. Isa Inuwa-Dutse

Module Tutor: Muhammad Usman, Dr Yuchu Qin

Prepared by

Dirdh Prafullkumar Patel

MSc. Artificial Intelligence

U2366489

Table of Contents

1 Introduction

In the era of Machine Learning (ML), the potency of predictive models has become paramount, with Supervised Learning standing out as a pivotal paradigm. This report delves into the application of the Random Forest Regressor, a robust algorithm within Supervised Learning, for predicting students' final grades. Extensive preprocessing, encompassing factor analysis, outlier imputation using IQR method, and exploratory data analysis, has been conducted. Additionally, a Random Forest Classifier has been employed to categorize students into groups based on academic performance. This study aims to unravel the nuances of student grade prediction through the lens of advanced ML techniques.

2 Questions

2.1 Question 1: Exploratory Data Analysis (EDA)

2.1.1 Preparing the Data (Merging)

```
In [6]: df1= pd.read_csv('student-mat.csv',sep=';')
        df1['subject']= "Maths"
        df2= pd.read_csv('student-por.csv',sep=';')
        df2['subject']="Portuguese"
        df=df1.merge(df2,how='outer',on=df1.columns.tolist())
        df.sample(5)
```

Out[6]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	subject
148	GP	M	16	U	GT3	T	4	4	teacher	teacher	...	3	2	2	1	5	0	7	6	0	Maths
365	MS	M	18	R	GT3	T	1	3	at_home	other	...	3	4	2	4	3	4	10	10	10	Maths
979	MS	F	17	R	GT3	T	0	0	at_home	other	...	4	3	1	1	5	0	10	11	11	Portuguese
846	MS	M	16	R	GT3	T	1	2	other	other	...	3	3	1	1	5	0	10	11	11	Portuguese
649	GP	F	18	U	LE3	T	2	2	at_home	services	...	3	1	1	1	5	16	9	8	10	Portuguese

5 rows × 34 columns

Figure 2.1.1. Python code to merge two dataframe

2.1.2 Size of the data

```
In [7]: df.size

Out[7]: 35496
```

Figure 2.1.2. A Python code snippet showing the size of the data.

This figure shows the size of data is 35496.

2.1.3 Numbers of rows and columns in the Data

```
In [8]: #rows
        df.shape[0]

Out[8]: 1044

In [9]: #columns
        df.shape[1]

Out[9]: 34
```

Figure 2.1.3 A Python code snippet showing the number of rows and columns in the data.

2.1.4 Details about Columns

In [63]:
```
#sample data of with all columns names
df.T.iloc[:,:10]
```

Out[63]:

	0	1	2	3	4	5	6	7	8	9
school	GP	GP	GP	GP	GP	GP	GP	GP	GP	GP
sex	F	F	F	F	F	M	M	F	M	M
age	18.0	17.0	15.0	15.0	16.0	16.0	16.0	17.0	15.0	15.0
address	U	U	U	U	U	U	U	U	U	U
famsize	GT3	GT3	LE3	GT3	GT3	LE3	LE3	GT3	LE3	GT3
Pstatus	A	T	T	T	T	T	T	A	A	T
Medu	4	1	1	4	3	4	2	4	3	3
Fedu	4	1	1	2	3	3	2	4	2	4
Mjob	at_home	at_home	at_home	health	other	services	other	other	services	other
Fjob	teacher	other	other	services	other	other	other	teacher	other	other
reason	course	course	other	home	home	reputation	home	home	home	home
guardian	mother	father	mother	mother	father	mother	mother	mother	mother	mother
traveltime	2	1	1	1	1	1	1	2	1	1
studytime	2	2	2	3	2	2	2	2	2	2
failures	0	0	3	0	0	0	0	0	0	0
schoolsup	yes	no	yes	no	no	no	no	yes	no	no
famsup	no	yes	no	yes	yes	yes	no	yes	yes	yes
paid	no	no	yes	yes	yes	yes	no	no	yes	yes
activities	no	no	no	yes	no	yes	no	no	no	yes
nursery	yes	no	yes	yes	yes	yes	yes	yes	yes	yes
higher	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
internet	no	yes	yes	yes	no	yes	yes	no	yes	yes
romantic	no	no	no	yes	no	no	no	no	no	no
famrel	4	5	4	3	4	5	4	4	4	5
freetime	3	3	3	2	3	4	4	1	2	5
goout	4	3	2	2	2	2	4	4	2	1
Dalc	1	1	2	1	1	1	1	1	1	1
Walc	1	1	3	1	2	2	1	1	1	1
health	3	3	3	5	5	5	3	1	1	5
absences	6.0	4.0	10.0	2.0	4.0	10.0	0.0	6.0	0.0	0.0
G1	5.0	5.0	7.0	15.0	6.0	15.0	12.0	6.0	16.0	14.0
G2	6.0	5.0	8.0	14.0	10.0	15.0	12.0	5.0	18.0	15.0
G3	6.0	6.0	10.0	15.0	10.0	15.0	11.0	6.0	19.0	15.0
subject	Maths	Maths	Maths	Maths	Maths	Maths	Maths	Maths	Maths	Maths

Figure 2.1.4 3 A Python code snippet showing all columns name and sample data

2.1.5 Datatypes of all Columns

```
In [70]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1044 entries, 0 to 1043
Data columns (total 34 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      1044 non-null   object
 1   sex         1044 non-null   object
 2   age         1044 non-null   int64
 3   address     1044 non-null   object
 4   famsize     1044 non-null   object
 5   Pstatus     1044 non-null   object
 6   Medu        1044 non-null   int64
 7   Fedu        1044 non-null   int64
 8   Mjob        1044 non-null   object
 9   Fjob        1044 non-null   object
 10  reason      1044 non-null   object
 11  guardian    1044 non-null   object
 12  traveltime  1044 non-null   int64
 13  studytime   1044 non-null   int64
 14  failures    1044 non-null   int64
 15  schoolsup   1044 non-null   object
 16  famsup      1044 non-null   object
 17  paid        1044 non-null   object
 18  activities  1044 non-null   object
 19  nursery     1044 non-null   object
 20  higher      1044 non-null   object
 21  internet    1044 non-null   object
 22  romantic    1044 non-null   object
 23  famrel      1044 non-null   int64
 24  freetime    1044 non-null   int64
 25  goout       1044 non-null   int64
 26  Dalc        1044 non-null   int64
 27  Walc        1044 non-null   int64
 28  health      1044 non-null   int64
 29  absences    1044 non-null   int64
 30  G1          1044 non-null   int64
 31  G2          1044 non-null   int64
 32  G3          1044 non-null   int64
 33  subject     1044 non-null   object
dtypes: int64(16), object(18)
memory usage: 277.4+ KB
```

Figure 2.1.5 3 A Python code snippet showing datatype of all columns

2.1.6 Missing values

```
In [30]: df.isna().sum()
```

```
Out[30]: 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
subject       0
dtype: int64
```

Figure2.1.6 A Python code snippet showing Missing value according the Columns

This figure shows the missing value according to the columns, by observing figure no columns have any missing values

2.1.7 Duplicated Rows

2.1.7.1 Duplicated Rows according identical attributes of Each student

```
duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus",
                                      "Medu","Fedu","Mjob","Fjob","reason","nursery","internet"],keep='first')
df[duplicated_rows]
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	subject
144	GP	M	17	U	GT3	T	2	1	other	other	...	4	5	1	2	5	0	5	0	0	Maths
148	GP	M	16	U	GT3	T	4	4	teacher	teacher	...	3	2	2	1	5	0	7	6	0	Maths
181	GP	M	16	U	GT3	T	3	3	services	other	...	2	3	1	2	3	2	12	13	12	Maths
341	GP	M	18	U	GT3	T	4	4	teacher	services	...	3	3	2	2	2	0	10	10	0	Maths
395	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	3	4	1	1	3	4	0	11	11	Portuguese
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1039	MS	F	19	R	GT3	T	2	3	services	other	...	4	2	1	2	5	4	10	11	10	Portuguese
1040	MS	F	18	U	LE3	T	3	1	teacher	services	...	3	4	1	1	1	4	15	15	16	Portuguese
1041	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	5	6	11	12	9	Portuguese
1042	MS	M	17	U	LE3	T	3	1	services	services	...	4	5	3	4	2	6	10	10	10	Portuguese
1043	MS	M	18	R	LE3	T	3	2	services	other	...	4	1	3	4	5	4	10	11	11	Portuguese

382 rows × 34 columns

Figure 2.1.7.1 A Python code snippet showing Duplicated Rows According the identical attributes of Each student

```python
duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus",
                                       "Medu","Fedu","Mjob","Fjob","reason","nursery","internet","G1","G2","G3"])
df[duplicated_rows]
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	subject
712	GP	F	17	U	LE3	T	2	2	services	other	...	4	4	2	3	5	6	12	12	12	Portuguese
774	GP	M	17	R	GT3	T	2	2	services	other	...	4	5	5	5	4	2	11	10	10	Portuguese
878	MS	F	16	R	GT3	T	2	2	other	other	...	4	5	1	2	1	1	9	10	11	Portuguese

3 rows × 34 columns

Figure 2.1.7.2 A Python code snippet showing Duplicated Rows According the identical attributes and grades of Each student

```python
duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus","Medu","Fedu",
                                       "Mjob","Fjob","reason","nursery","internet","G1","G2","G3","subject"],keep='first')
df[duplicated_rows]
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	subject
878	MS	F	16	R	GT3	T	2	2	other	other	...	4	5	1	2	1	1	9	10	11	Portuguese

1 rows × 34 columns

Figure 2.1.7.3 A Python code snippet showing Duplicated Rows According the identical attributes, grades and, Subject of Each student

This Figure shows that there is only one student having same marks, same identical Attributes and Same Subject.

```python
df=df.drop_duplicates(subset=["school","sex","age","address","famsize","Pstatus","Medu","Fedu","Mjob","Fjob","reason",
                               "nursery","internet","G1","G2","subject"],keep='last')
df=df.reset_index(drop=True)
df
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	subject
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	3	4	1	1	3	6	5	6	6	Maths
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	3	3	1	1	3	4	5	5	6	Maths
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	3	2	2	3	3	10	7	8	10	Maths
3	GP	F	15	U	GT3	T	4	2	health	services	...	2	2	1	1	5	2	15	14	15	Maths
4	GP	F	16	U	GT3	T	3	3	other	other	...	3	2	1	2	5	4	6	10	10	Maths
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1038	MS	F	19	R	GT3	T	2	3	services	other	...	4	2	1	2	5	4	10	11	10	Portuguese
1039	MS	F	18	U	LE3	T	3	1	teacher	services	...	3	4	1	1	1	4	15	15	16	Portuguese
1040	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	5	6	11	12	9	Portuguese
1041	MS	M	17	U	LE3	T	3	1	services	services	...	4	5	3	4	2	6	10	10	10	Portuguese
1042	MS	M	18	R	LE3	T	3	2	services	other	...	4	1	3	4	5	4	10	11	11	Portuguese

1043 rows × 34 columns

Figure 2.1.7.3 A Python code snippet showing deleted Duplicated Row According the identical attributes, marks and, Subject of Each student

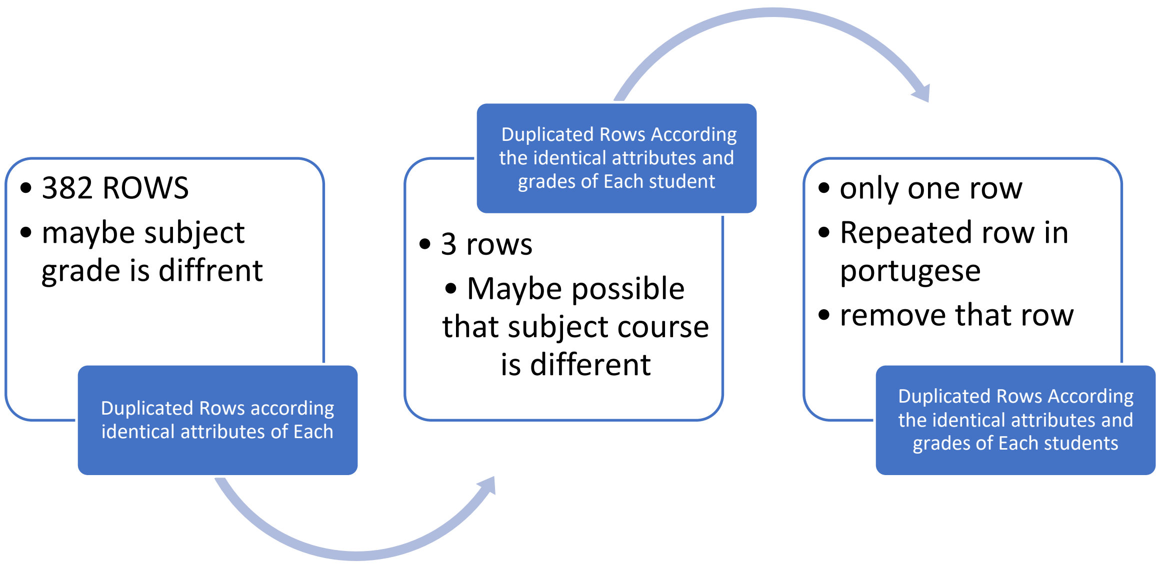

Figure 2.1.7.2 Flow chart to detect the repeated student

2.1.8 Unique values in the columns

```
df.shape
unique_counts = df.nunique()
unique_counts
```

```
school         2
sex            2
age            8
address        2
famsize        2
Pstatus        2
Medu           5
Fedu           5
Mjob           5
Fjob           5
reason         4
guardian       3
traveltime     4
studytime      4
failures       4
schoolsup      2
famsup         2
paid           2
activities     2
nursery        2
higher         2
internet       2
romantic       2
famrel         5
freetime       5
goout          5
Dalc           5
Walc           5
health         5
absences      35
G1            18
G2            17
G3            19
subject        2
dtype: int64
```

Figure 2.1.8 A Python code snippet showing Unique Values in each column

2.1.9 Outliers

```python
def boxanddistplot(df):
    unique_counts = df.nunique()
    selected_columns = unique_counts[unique_counts >= 6].index.tolist()
    for column_name in selected_columns:
        plt.figure(figsize=(15, 6))
        plt.subplot(1, 2, 1)
        sns.boxplot(x=column_name,data=df)
        plt.title(f"box Plot of {column_name}",fontsize=16,loc='center')
        plt.xlabel(f"{column_name}")
        plt.subplot(1,2,2)
        sns.histplot(df[column_name], kde=True,bins=df[column_name].nunique() ,kde_kws=dict(cut=3))
        plt.xlabel(f"{column_name}")
        plt.ylabel('no of students')
        plt.title(f" of {column_name}",fontsize=16,loc='center')
        plt.show()
```

Figure 2.1.9.1 A Python code snippet for plotting the Boxplot and Histogram

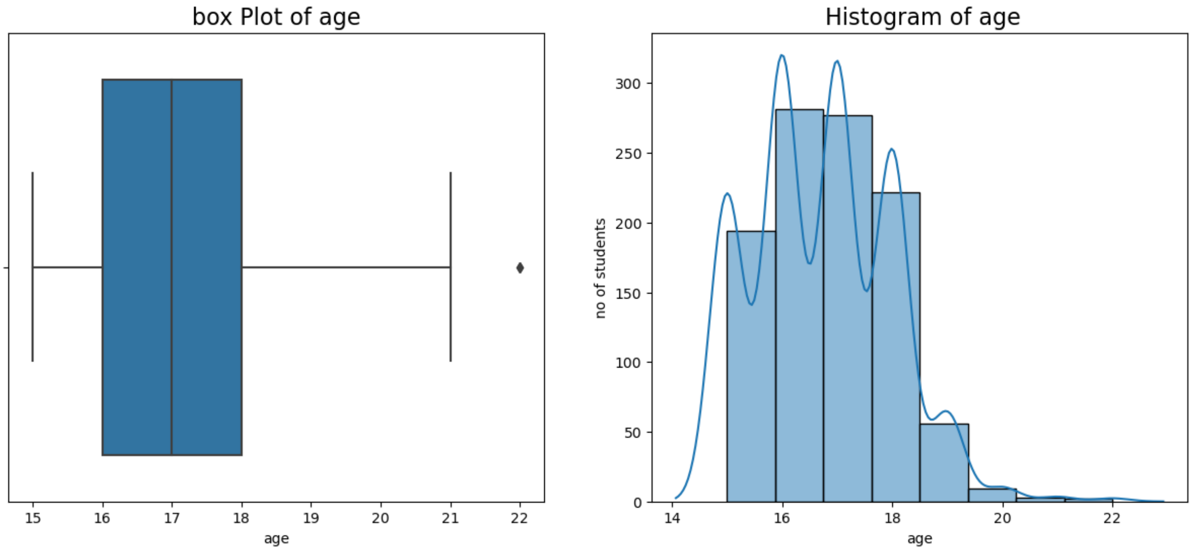

Figure 2.1.9.2 Boxplot and Histogram of Student's Age

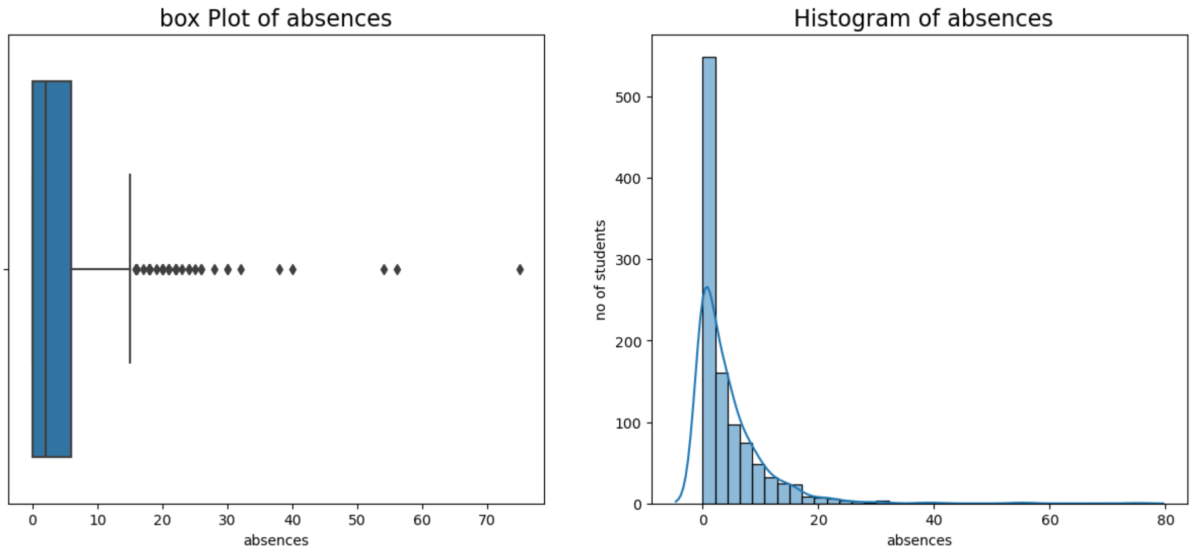

Figure 2.1.9.3 Boxplot and Histogram of Student's Absences during the term

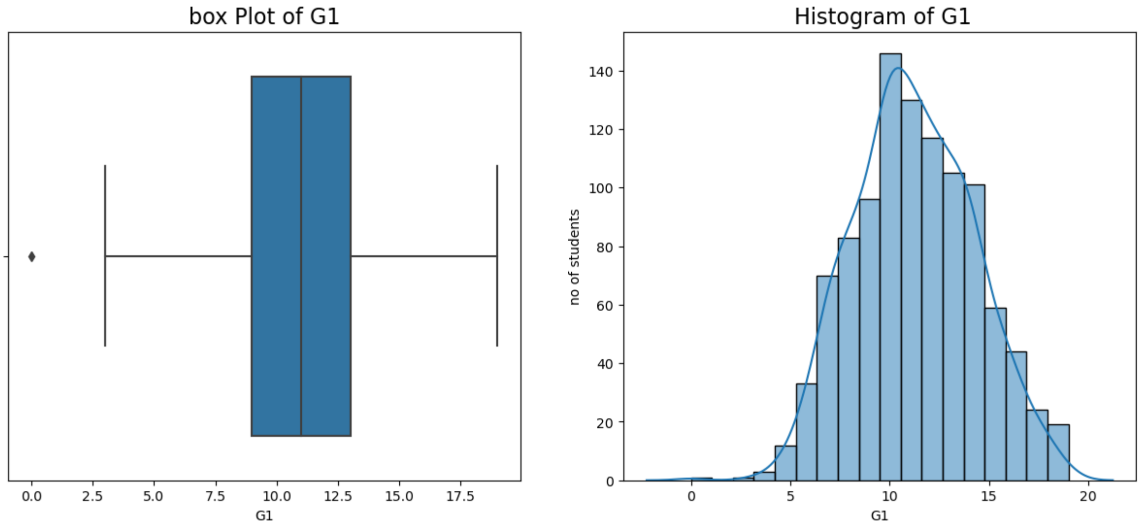

Figure 2.1.9.4 Boxplot and Histogram of Student's first period grade

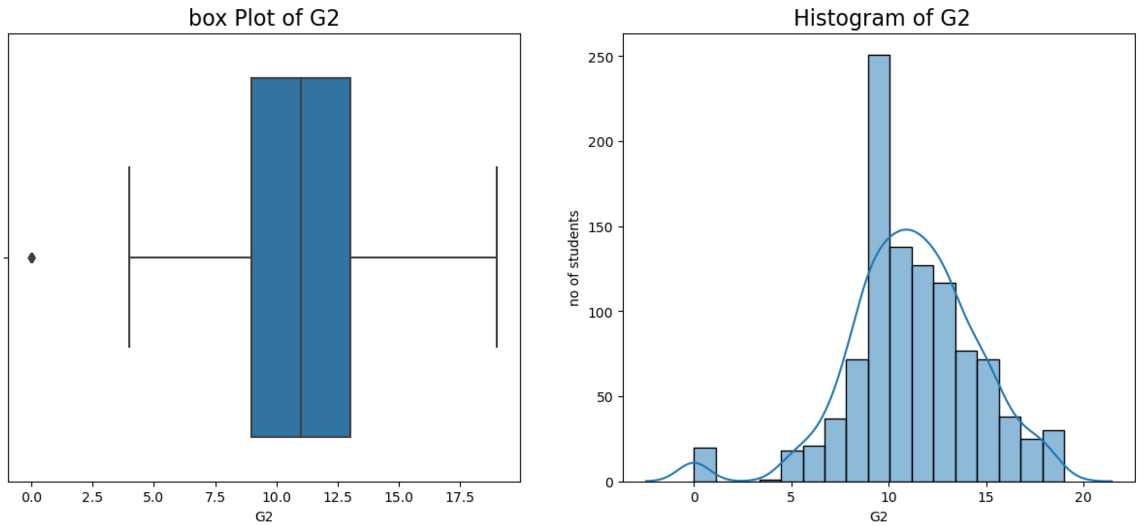

Figure 2.1.9.5 Boxplot and Histogram of Student's second period grade

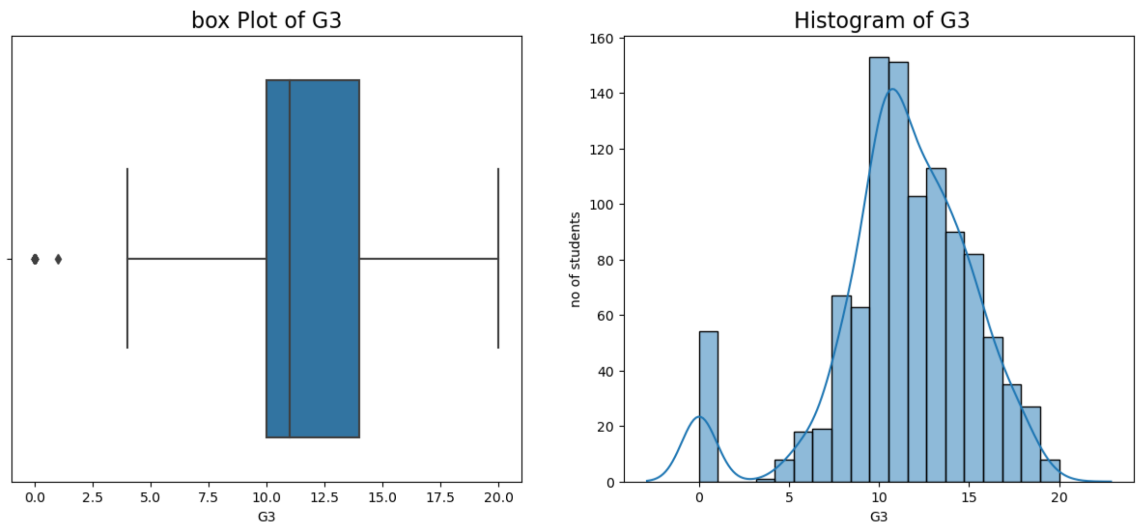

Figure 2.1.9.6 Boxplot and Histogram of Student's final grade

```python
unique_counts = df.nunique()
selected_columns = unique_counts[unique_counts >= 6].index.tolist()
for columns1 in selected_columns:
    percentile25 = df[columns1].quantile(0.25)
    percentile75 = df[columns1].quantile(0.75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    df[columns1] = np.where(
    df[columns1] > upper_limit,upper_limit,np.where(df[columns1] < lower_limit,lower_limit,df[columns1]))
```

```python
boxanddistplot(df)
df.describe()
```

Figure 2.1.9.7 A Python code snippet to handle outliers and for plotting the Boxplot and Histogram

To handle the outlier, I used IQR imputation technique.

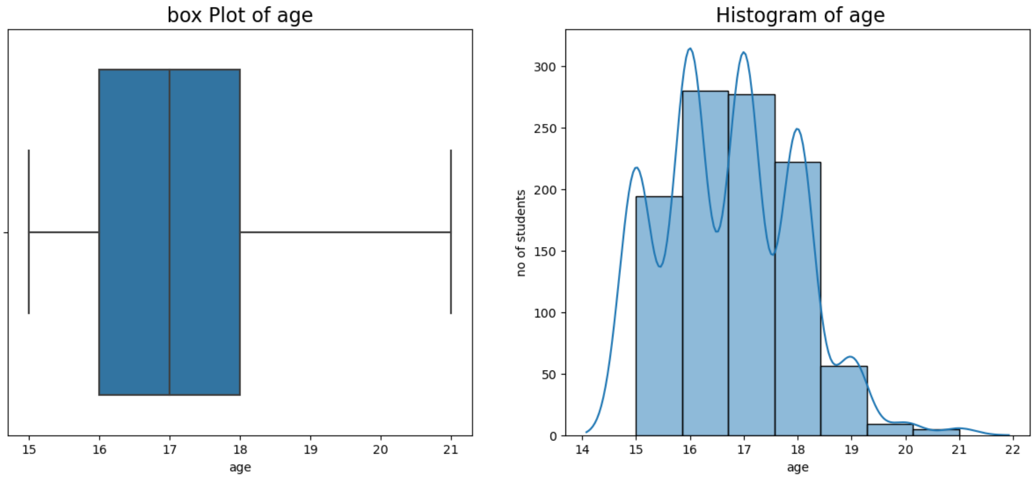

Figure 2.1.9.8 after imputation boxplot and histogram of student's age

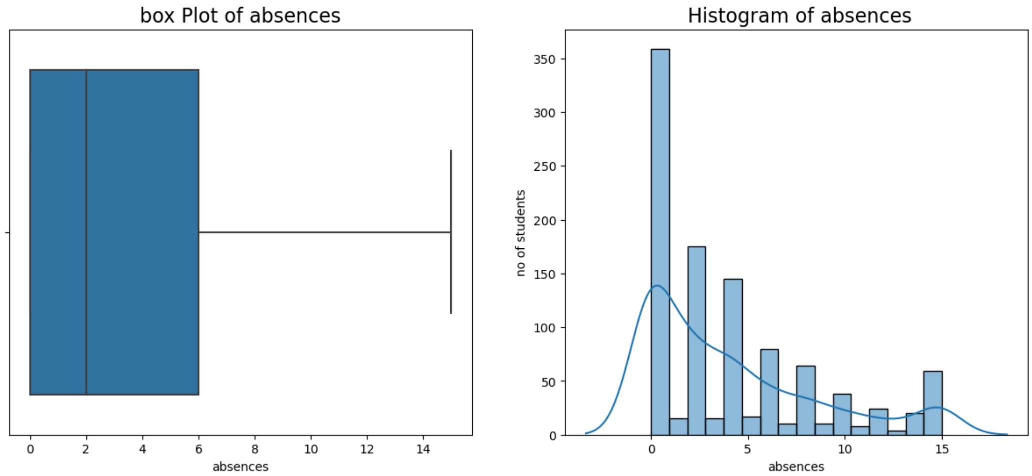

Figure 2.1.9.9 after imputation boxplot and histogram of student's absence in the class

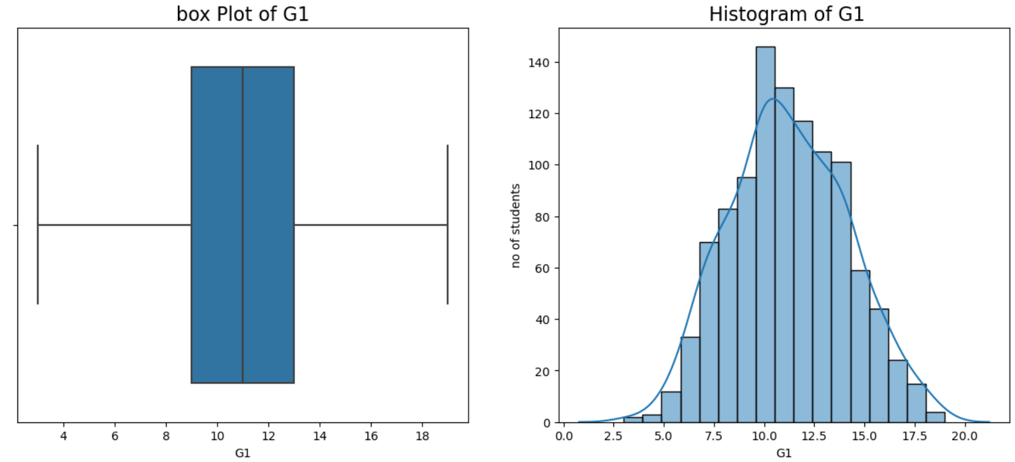

Figure 2.1.9.10 after imputation boxplot and histogram of student's first period grades

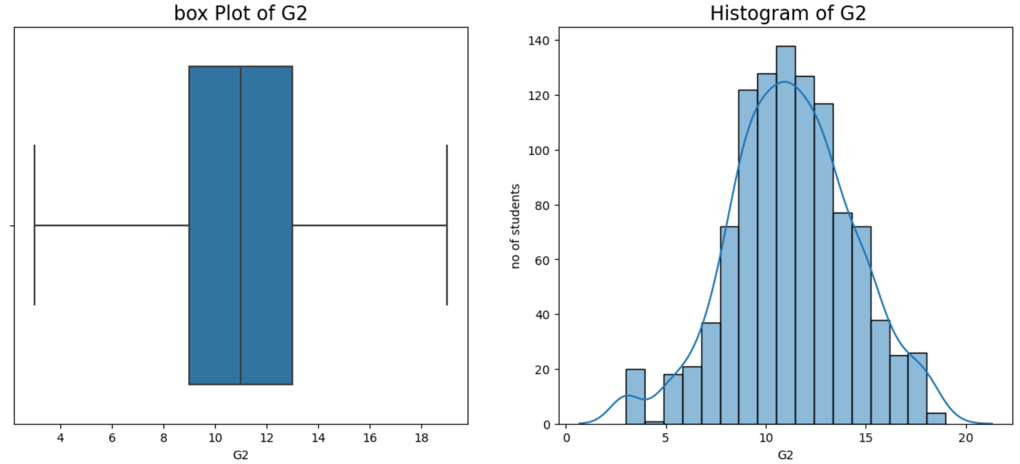

Figure 2.1.9.10 after imputation boxplot and histogram of student's Second period grades

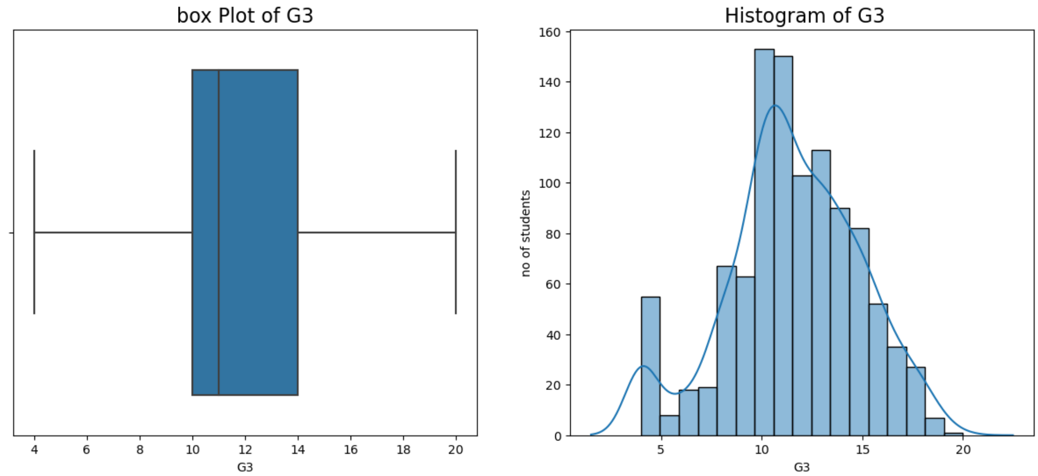

Figure 2.1.9.10 after imputation boxplot and histogram of student's final grades

Columns name	Type of distribution	Outliers	Need to handle outlier?
Age	Left skewed Normal	Yes	Yes (IQR)
Absences	Exponentially decreased	Yes	Yes (IQR)
G1	Near to Normal	Yes	Yes (IQR)
G2	Normal	Yes	Yes (IQR)
G3	Near to Normal	Yes	Yes (IQR)

Table 2.1.9 Summery of Outliers in Data

2.1.10 Association with Final grades

I have used the chi square association to finding the association with the Final grades.

```
def chi_square_association(df, categorical_column, output_column):

    contingency_table = pd.crosstab(df[categorical_column], df[output_column])
    associtated=[]
    BOLD = '\033[1m'
    END_BOLD = '\033[0m'
    chi2, p, _, _ = chi2_contingency(contingency_table)

    print(f"{BOLD}column:{categorical_column}{END_BOLD}")
    print(f"Chi-Square Statistic: {chi2}")
    print(f"{BOLD}P-value: {p}{END_BOLD}")


    alpha = 0.05
    print(f"Significance Level: {alpha}")
    print(f"Degrees of Freedom: {(contingency_table.shape[0] - 1) * (contingency_table.shape[1] - 1)}")

    if p < alpha:
        print(f"{BOLD}There is asignificant association between '{categorical_column}' and '{output_column}{END_BOLD}'.")
        print("   ")
        return categorical_column
    else:
        print(f"There is no significant association between the variables.")
        print("   ")
        return None
```

```
def check_association(df,column_name11):
    columns=[]
    for column_name in df.columns:
        result=chi_square_association(df, column_name, output_column=column_name11)
        if  result is not None:
            columns.append(result)
    print("Columns with significant association:")
    print(columns)
    for associatoncolums in columns:
        unique_values= df[associatoncolums].nunique()
        if unique_values >= 6:
            plt.figure(figsize=(7, 5))
            sns.scatterplot(x=associatoncolums, y=column_name11, data=df)
            plt.title(f"scatterplot  of final grade and {associatoncolums}",fontsize=16,loc='center')
            plt.xlabel(f"{associatoncolums}")
            plt.ylabel('final grades')
            plt.show()
    return columns
```

```
mycolumns=check_association(df,'G3')
```

Figure 2.1.10 A Python code to check association with Final grades

2.1.10.1 School

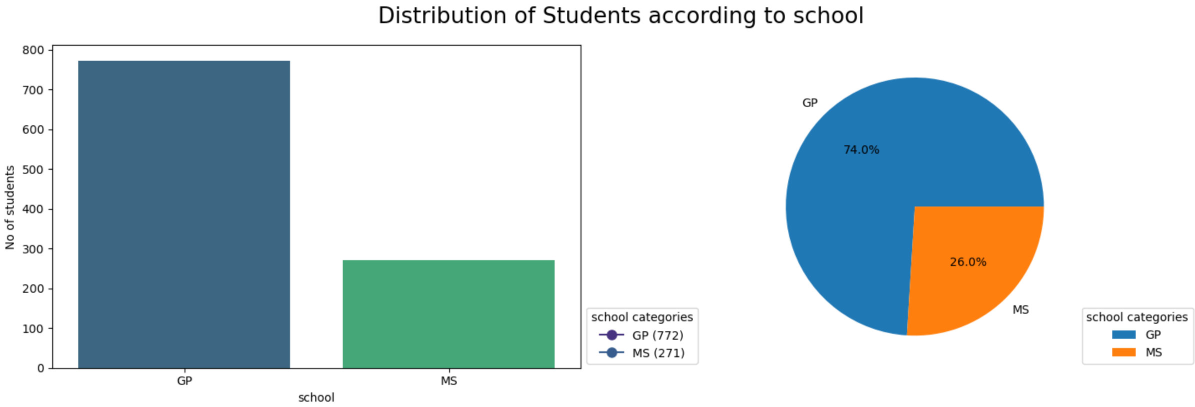

Figure 2.1.10.1.1 Distribution Of students According to School.

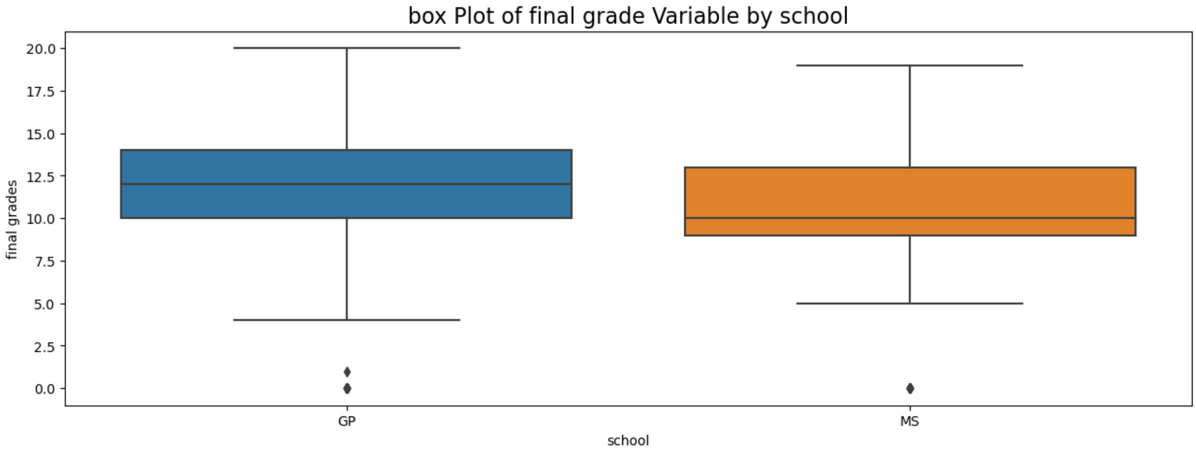

Figure 2.1.10.1.2 Box plot of Final grade according the schools

2.1.10.2 Mothers Education

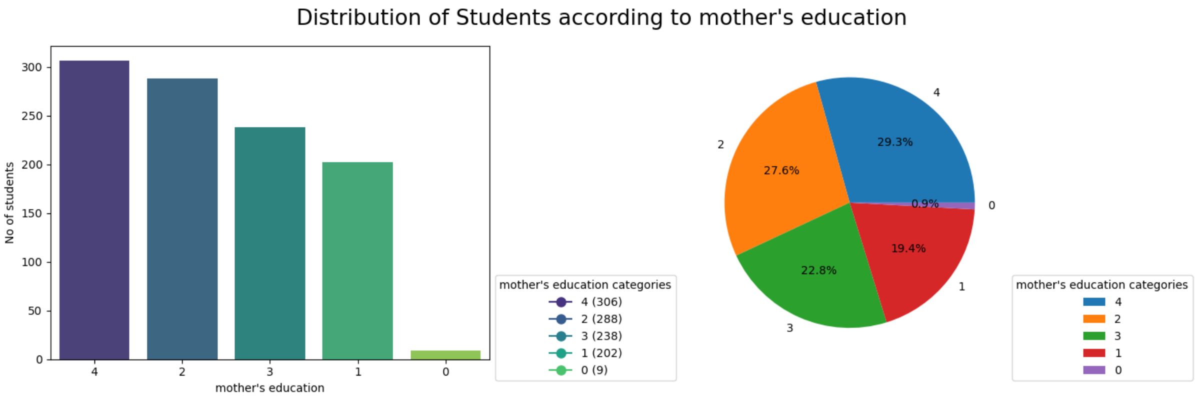

Figure 2.1.10.2.1 Distribution of Students according the mother education

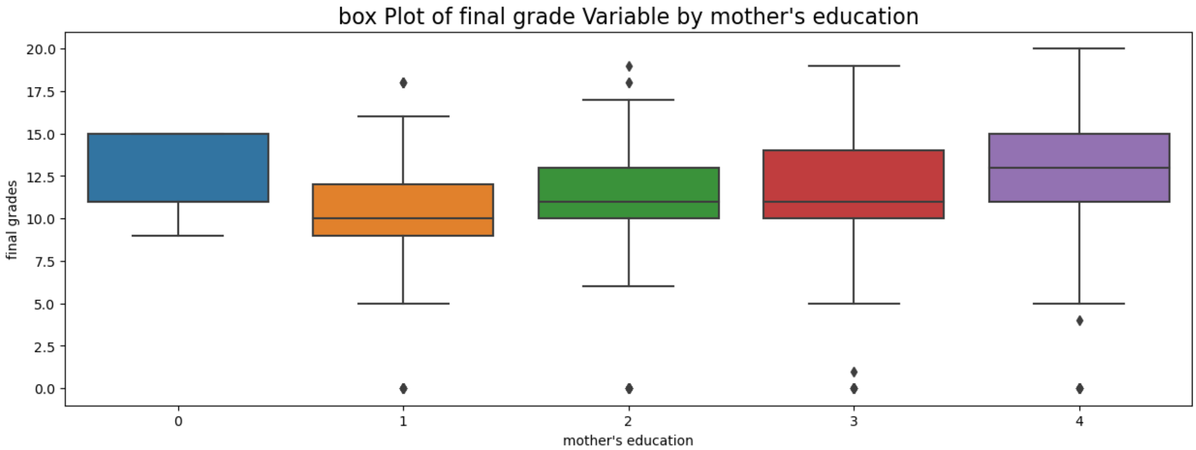

Figure 2.1.10.2.2 Box plot of final grades according the Mothers education

2.1.10.3 Father's Education

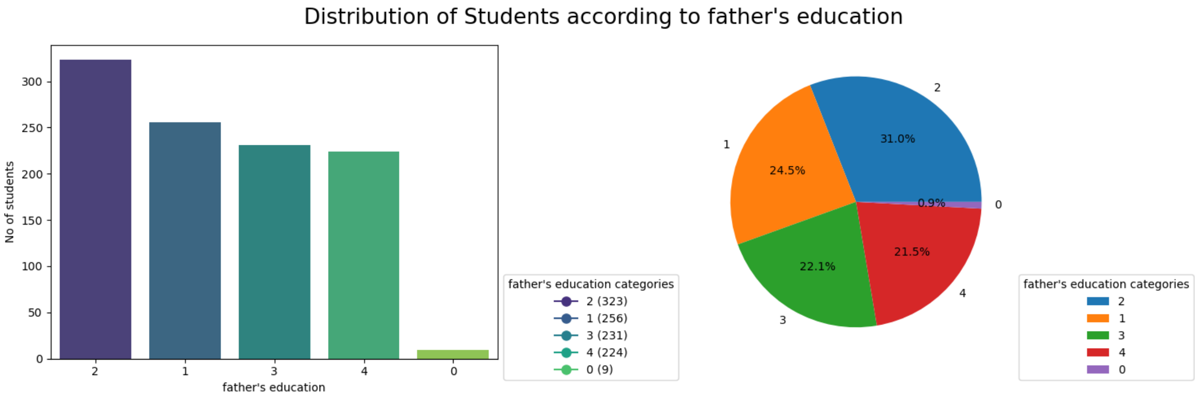

Figure 2.1.10.3.1 Distribution of Student According

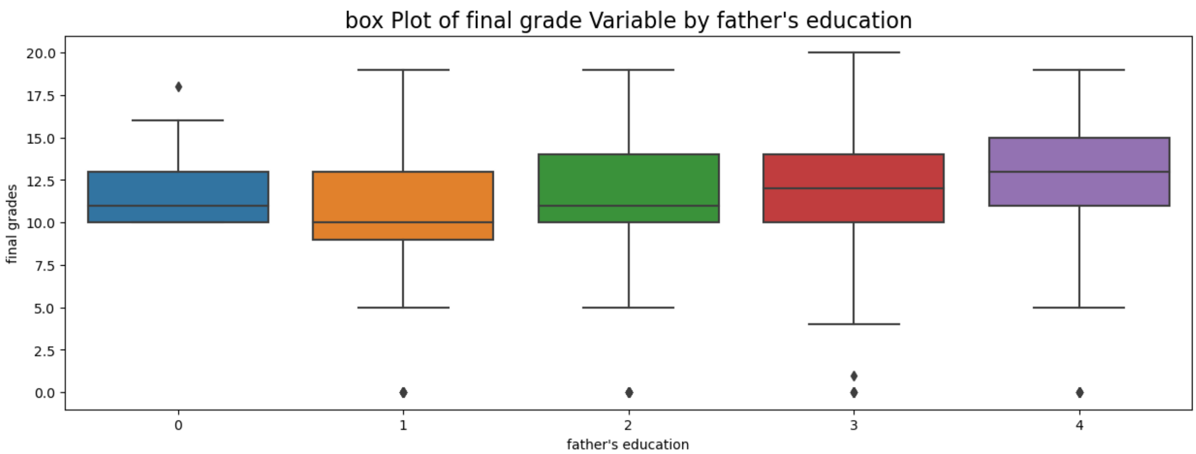

Figure 2.1.10.3.2 Box plot of final grade according Father's education

2.1.10.4 Daily Consumption of Alcohol

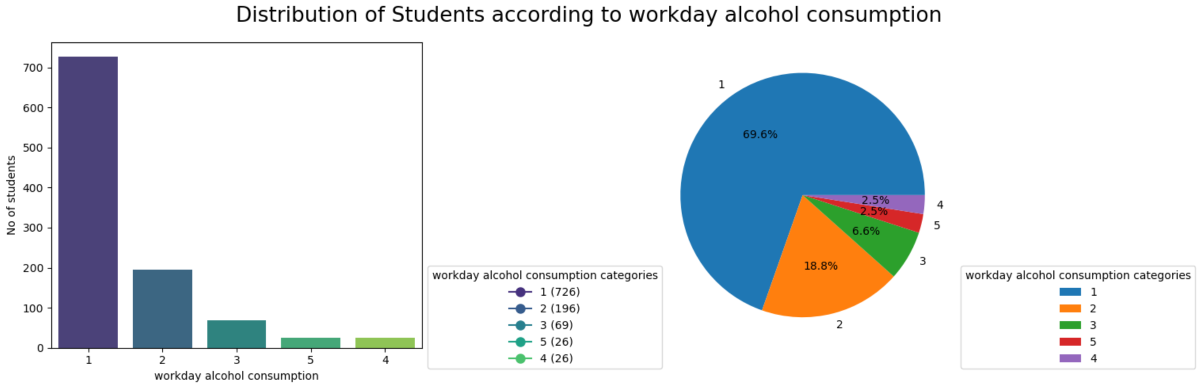

Figure 2.1.10.4.1 Distribution of students According to Daily Consumption Alcohol

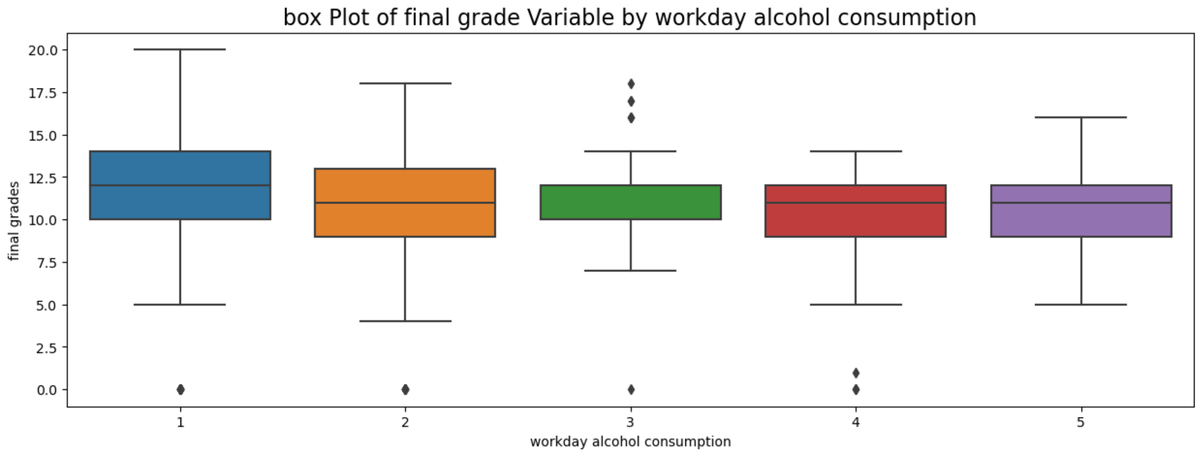

Figure 2.1.10.4.2 Box plot of final grade according Daily consumption alcohol

2.1.10.5 First period and second Period grades

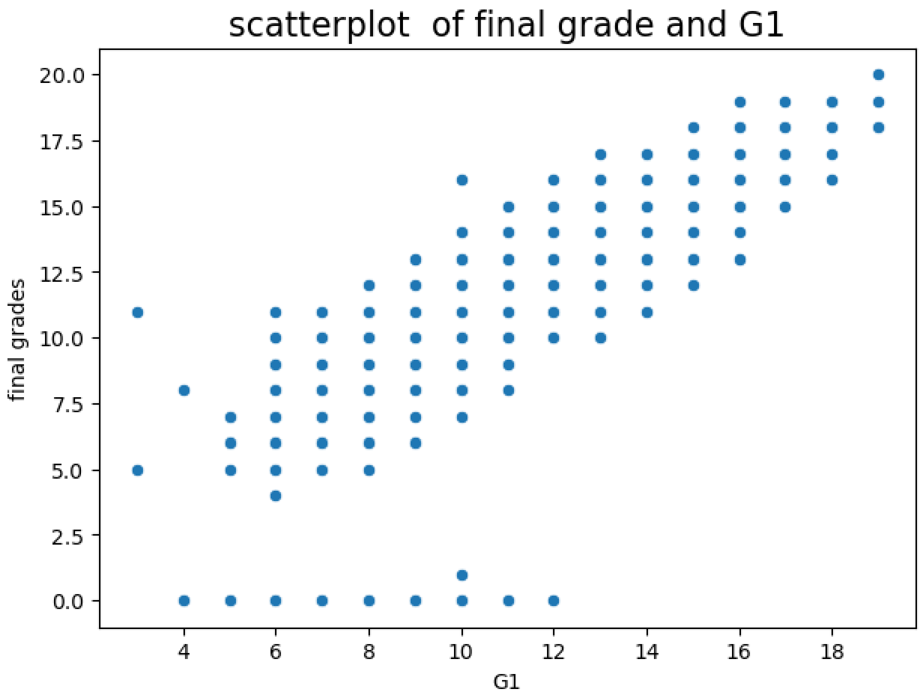

Figure 2.1.10.5.1 Scatterplot of final Grades and First period grade

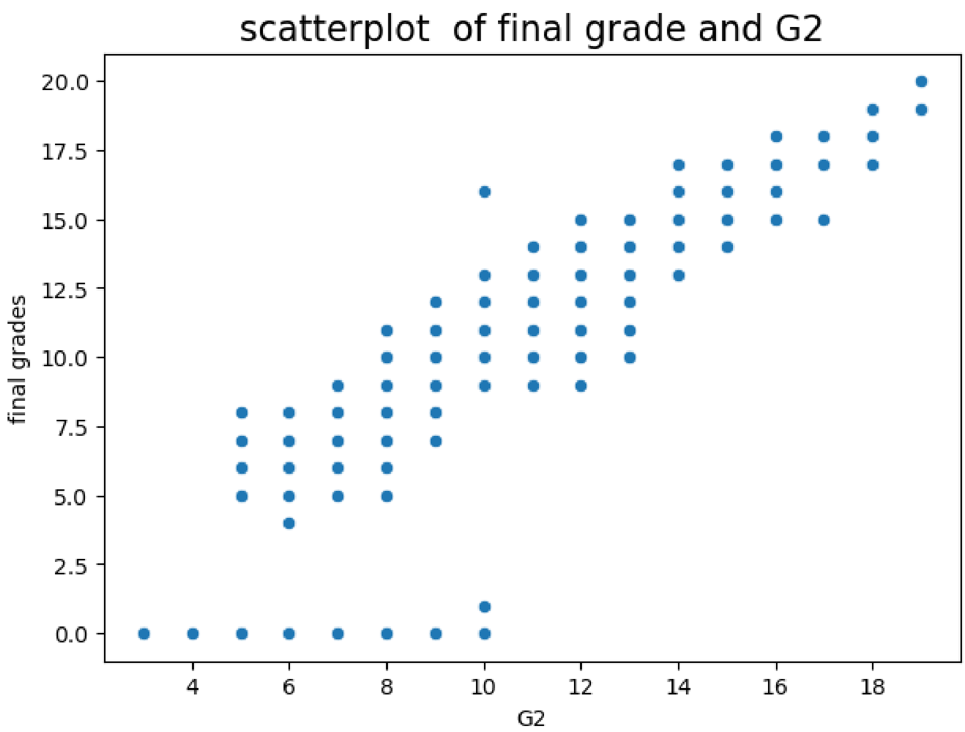

Figure 2.1.10.5.2 Scatterplot of final Grades and Second period grade

In the summary of EDA, the removed the duplicated rows and identify the outliers in the data and using IQR method handle that outlier without disturbing the distribution of each Features and **from the data two insightful finding is students' grades depends on the mother and father's education as well as higher consumption of alcohol is also the factor for the lower score in final grades**. However, Grade of first and second period is linearly co- related with the final grade even the data is majorly collected from school GP Gabriel Pereira whose students have the higher final grades then school MP Mousinho da Silveira.

2.2 Advanced question 1 Factor analysis

2.2.1 Preparing data For the Factor analysis

```
newdf=df[mycolumns]
```

```
newdf=newdf.drop(columns=['address','Mjob','romantic'])
```

```
newdf
```

	school	age	Medu	Fedu	studytime	failures	schoolsup	higher	goout	Dalc	absences	G1	G2	G3	subject
0	GP	18.0	4	4	2	0	yes	yes	4	1	6.0	5.0	6.0	6.0	Maths
1	GP	17.0	1	1	2	0	no	yes	3	1	4.0	5.0	5.0	6.0	Maths
2	GP	15.0	1	1	2	3	yes	yes	2	2	10.0	7.0	8.0	10.0	Maths
3	GP	15.0	4	2	3	0	no	yes	2	1	2.0	15.0	14.0	15.0	Maths
4	GP	16.0	3	3	2	0	no	yes	2	1	4.0	6.0	10.0	10.0	Maths
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1038	MS	19.0	2	3	3	1	no	yes	2	1	4.0	10.0	11.0	10.0	Portuguese
1039	MS	18.0	3	1	2	0	no	yes	4	1	4.0	15.0	15.0	16.0	Portuguese
1040	MS	18.0	1	1	2	0	no	yes	1	1	6.0	11.0	12.0	9.0	Portuguese
1041	MS	17.0	3	1	1	0	no	yes	5	3	6.0	10.0	10.0	10.0	Portuguese
1042	MS	18.0	3	2	1	0	no	yes	1	3	4.0	10.0	11.0	11.0	Portuguese

1043 rows × 15 columns

Figure 2.2.1.1 A Python code snippet to create the new data frame of Associated columns

```
object_cols = newdf.select_dtypes(include=['object']).columns.tolist()
```

```
preprocessor = ColumnTransformer(transformers=[('onehot', OneHotEncoder(drop='first'), object_cols)],remainder='passthrough')
```

```
df_transformed = pd.DataFrame(preprocessor.fit_transform(newdf), columns=preprocessor.get_feature_names_out(newdf.columns))
df_transformed
```

	onehot__school_MS	onehot__schoolsup_yes	onehot__higher_yes	onehot__subject_Portuguese	remainder__age	remainder__Medu	remainder__Fedu	rema
0	0.0	1.0	1.0	0.0	18.0	4.0	4.0	
1	0.0	0.0	1.0	0.0	17.0	1.0	1.0	
2	0.0	1.0	1.0	0.0	15.0	1.0	1.0	
3	0.0	0.0	1.0	0.0	15.0	4.0	2.0	
4	0.0	0.0	1.0	0.0	16.0	3.0	3.0	
...	...	...	...	...	...	...	...	
1038	1.0	0.0	1.0	1.0	19.0	2.0	3.0	
1039	1.0	0.0	1.0	1.0	18.0	3.0	1.0	
1040	1.0	0.0	1.0	1.0	18.0	1.0	1.0	
1041	1.0	0.0	1.0	1.0	17.0	3.0	1.0	
1042	1.0	0.0	1.0	1.0	18.0	3.0	2.0	

1043 rows × 15 columns

Figure 2.2.1.2 A Python code Snippet of Onehot Encoding of new data frame

```
In [38]: X=df_transformed.drop(columns=['remainder__G3'])
         Y=df_transformed['remainder__G3']
```

Figure 2.2.1.3 A Python code snippet to decide input columns and target column

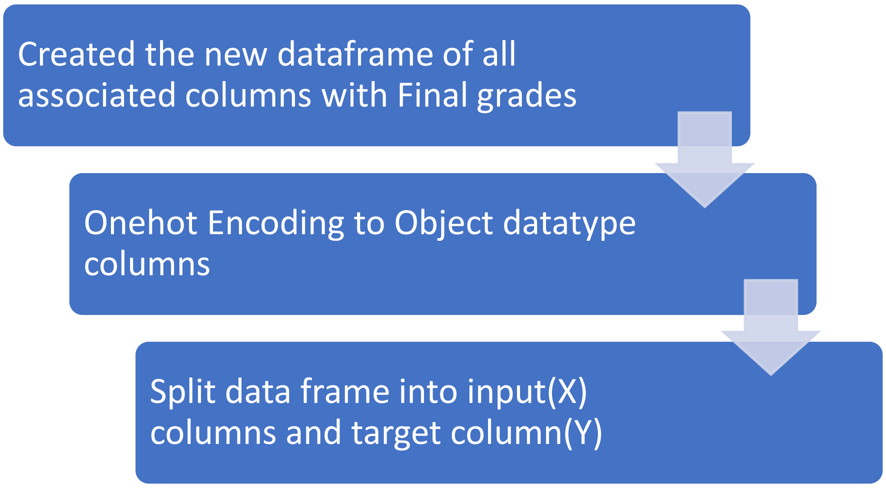

Figure 2.2.1.4 A flowchart for preparing the data for factor analysis

2.2.2 Factor analysis

```
from factor_analyzer import FactorAnalyzer
fa=FactorAnalyzer(n_factors=X.shape[1]-1)
fa.fit(X)
eigenvector, value= fa.get_eigenvalues()
eigenvector

array([2.97158398, 1.71681712, 1.4695741 , 1.19340853, 1.01173396,
       0.94088115, 0.86255194, 0.78135516, 0.72927126, 0.68956239,
       0.62330964, 0.54794442, 0.34947881, 0.11252754])
```

Figure 2.2.2.1 A Python code snippet to do factor analysis showing the eigenvector for each factor

```python
plt.scatter(range(1,X.shape[1]+1),eigenvector)
plt.plot(range(1,X.shape[1]+1),eigenvector)
plt.grid(True)
plt.xlabel(' no of Factors')
plt.ylabel('Eigen values')
plt.title("Scree plot of student's data")
plt.show()
```

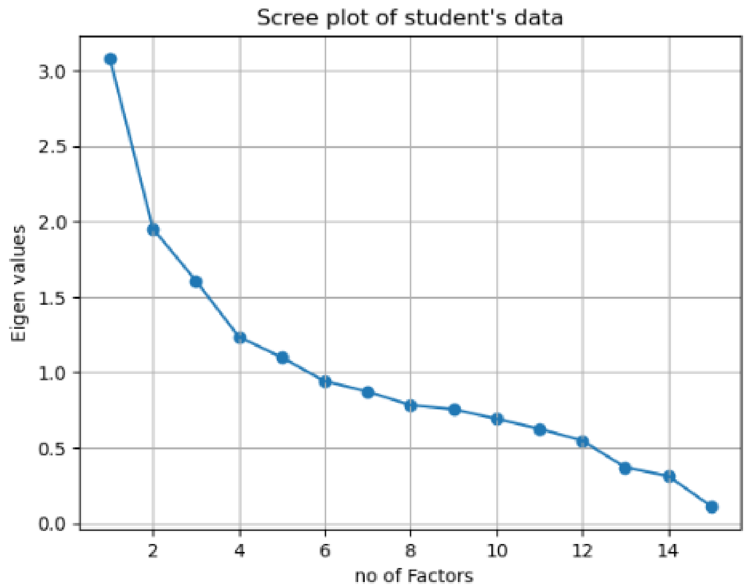

Figure 2.2.2.2 A Python code snippet to plot the scree plot for the data

```python
fa1=FactorAnalyzer(n_factors=6)
fa1.fit(X)
loadings=fa1.loadings_
```

Figure 2.2.1.3 A Python code factor analysis after deciding no of factors from the scree plot

```python
factor_loading=pd.DataFrame(loadings,index=X.columns)
```

```python
factor_loading
```

	0	1	2	3	4	5
onehot__school_MS	-0.235312	0.152353	-0.022699	0.652338	-0.002561	0.070122
onehot__schoolsup_yes	-0.165214	-0.187194	-0.042622	-0.094549	0.112595	-0.005317
onehot__higher_yes	0.049363	-0.051886	0.055405	-0.008234	0.524416	0.103189
onehot__subject_Portuguese	0.097627	-0.026053	-0.041374	0.410517	-0.083768	0.043294
remainder__age	-0.122742	1.110900	0.002232	0.023370	0.269000	0.021257
remainder__Medu	-0.007447	0.030232	0.781065	-0.089776	0.018615	-0.014666
remainder__Fedu	-0.074807	0.004707	0.839864	-0.022342	-0.021160	-0.028173
remainder__studytime	0.038683	0.179705	-0.055149	-0.068333	0.464952	-0.076644
remainder__failures	-0.219748	0.137917	-0.078872	-0.148474	-0.319814	-0.063066
remainder__goout	-0.038606	0.042975	0.019289	0.033080	0.039611	0.393819
remainder__Dalc	0.014212	-0.047716	-0.061741	0.002513	-0.006071	0.658581
remainder__absences	0.037726	0.073419	-0.000788	-0.355691	-0.096710	0.184856
remainder__G1	0.949365	-0.008917	-0.062416	-0.112239	0.106043	-0.035981
remainder__G2	0.916530	-0.026165	-0.040941	-0.061366	0.082350	-0.034805

Figure 2.2.2.3 A Python code for factors loading

```python
factoredselected_columns = []

# Set a threshold for loading values
loading_threshold = 0.5

for factor in factor_loading.columns:
    factoredselected_columns.extend(factor_loading.index[factor_loading[factor] > loading_threshold])

factoredselected_columns = list(set(factoredselected_columns))
factoredselected_columns
X=X[factoredselected_columns]
X
```

	remainder__Medu	remainder__G2	remainder__age	remainder__Fedu	remainder__G1	remainder__Dalc	onehot__higher_yes	onehot__school_MS
0	4.0	6.0	18.0	4.0	5.0	1.0	1.0	0.0
1	1.0	5.0	17.0	1.0	5.0	1.0	1.0	0.0
2	1.0	8.0	15.0	1.0	7.0	2.0	1.0	0.0
3	4.0	14.0	15.0	2.0	15.0	1.0	1.0	0.0
4	3.0	10.0	16.0	3.0	6.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...
1038	2.0	11.0	19.0	3.0	10.0	1.0	1.0	1.0
1039	3.0	15.0	18.0	1.0	15.0	1.0	1.0	1.0
1040	1.0	12.0	18.0	1.0	11.0	1.0	1.0	1.0
1041	3.0	10.0	17.0	1.0	10.0	3.0	1.0	1.0
1042	3.0	11.0	18.0	2.0	10.0	3.0	1.0	1.0

1043 rows × 8 columns

Figure 2.2.2.4 A Python code for selecting the columns according to the factor loadings

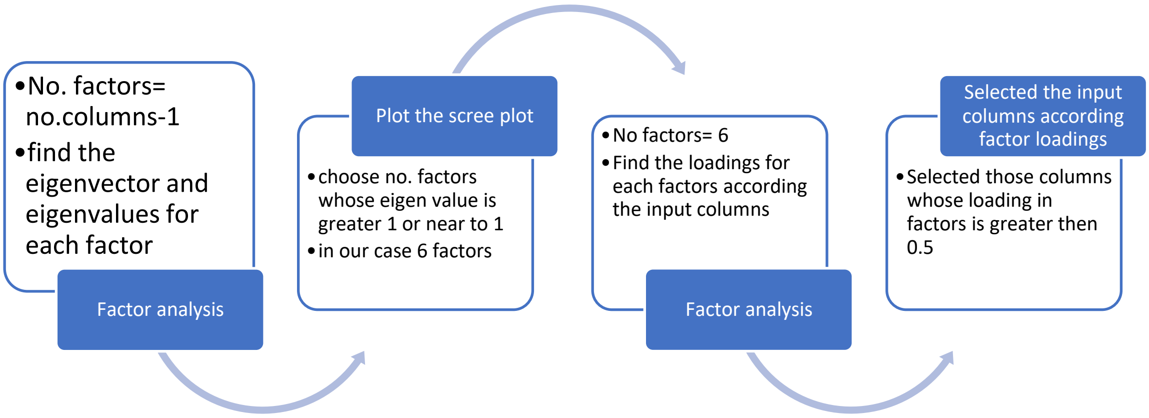

Figure 2.2.2.5 A Flowchart of factor analysis for the student dataset

2.3 Model building and training

2.3.1 Split data into training and testing

```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
columns_to_scale = X.columns
# Create a ColumnTransformer
minmax= ColumnTransformer(transformers=[('scaler', MinMaxScaler(), columns_to_scale)],remainder='passthrough')
# Fit and transform the feature data
X2= minmax.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X2, Y, test_size=0.3, random_state=42)
```

Figure 2.3.1 A python code snippet for normalization and split data for training and test

2.3.2 Train the model and its performance evolution

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score,r2_score


# Create the model
rf_model = RandomForestRegressor(n_estimators=300, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)
```

Figure 2.3 A Python code snippet for the train model using RF Regressor

```python
# Evaluate the model using different metrics
r_squared = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
explained_var = explained_variance_score(y_test, y_pred)
n = len(y_test)
k = X.shape[1]  # Number of predictors
adjusted_r_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

# Print the performance metrics
print(f'R-squared: {r_squared}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Explained Variance Score: {explained_var}')
print(f'Adjusted R-squared: {adjusted_r_squared}')
```

```
R-squared: 0.8543291182769109
Mean Squared Error (MSE): 1.5488108674272953
Mean Absolute Error (MAE): 0.8637805458356576
Explained Variance Score: 0.8543295750765345
Adjusted R-squared: 0.8504956740210402
```

Figure 2.3 A Python code snippet show the Performance of the model

The Random Forest Regressor performs well with an R-squared of 0.8543, indicating it explains 85.43% of variance. Mean Squared Error is 1.5488, and Mean Absolute Error is 0.8638, showcasing accurate predictions. The Adjusted R-squared of 0.8505 accounts for model complexity.

2.3.3 Feature Importance Graph

```python
feature_importances = rf_model.feature_importances_

# Create a DataFrame with feature names and their importance scores
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances,
})

# Sort the DataFrame by importance values in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importance graph
plt.figure(figsize=(10, 6))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Random Forest Regressor - Feature Importance')
plt.xticks(rotation=45)
plt.show()
```

Figure 2.3.3.1 A python code snippet for the features importance graph

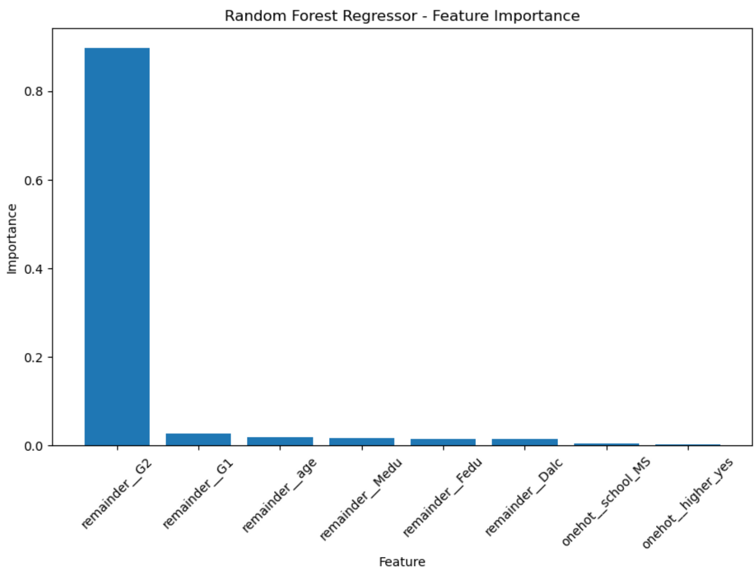

Figure 2.3.3.2 A feature importance Graph

This graph describes the domination of the second period grade to predict the final grade of the students where as other features shows very less importance for the prediction of the final grades.

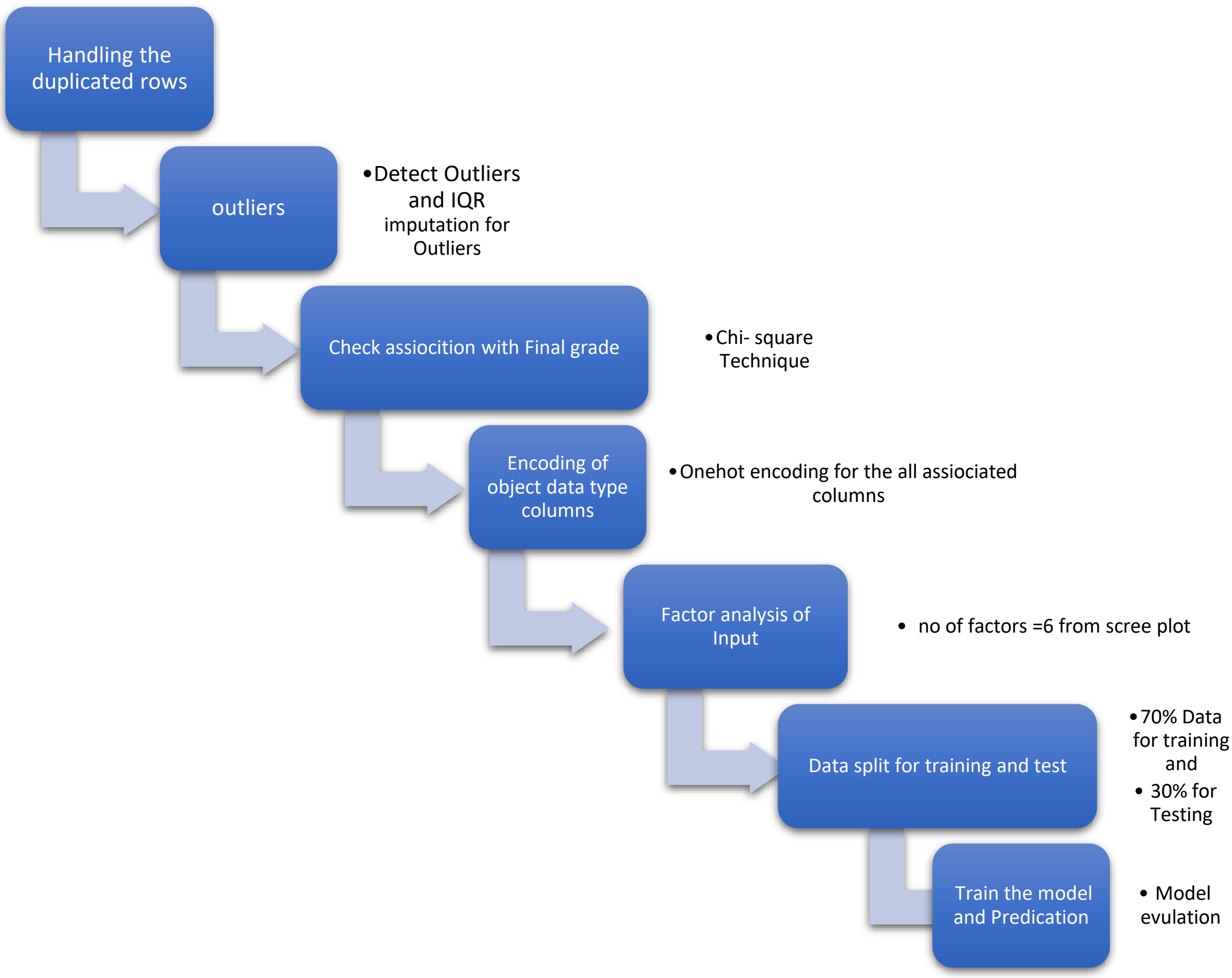

Figure 2.3.3.3 Data pipeline for the model

2.3.4 Optimization of the Model

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100,200,250,300,350,500],
    'max_depth': [None, 10, 20,12,13],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=10, scoring='r2')
grid_search.fit(X_train, y_train)

best_rf = grid_search.best_estimator_
best_rf.fit(X_train, y_train)

y_pred = best_rf.predict(X_test)
```

Figure 2.3.4.1 A Python Code snippet to select the best parameters to train the model

```python
grid_search.best_params_
```

```
{'max_depth': None,
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 100}
```

```python
r_squared = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
explained_var = explained_variance_score(y_test, y_pred)
n = len(y_test)
k = X.shape[1]  # Number of predictors
adjusted_r_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

# Print the performance metrics
print(f'Optimized R-squared: {r_squared}')
print(f'Optimized Mean Squared Error (MSE): {mse}')
print(f'Optimized Mean Absolute Error (MAE): {mae}')
print(f'OptimizedExplained Variance Score: {explained_var}')
print(f'OptimizedAdjusted R-squared: {adjusted_r_squared}')
```

```
Optimized R-squared: 0.8710419273018513
Optimized Mean Squared Error (MSE): 1.3711159160624085
Optimized Mean Absolute Error (MAE): 0.8240939558623301
OptimizedExplained Variance Score: 0.8710537299924426
OptimizedAdjusted R-squared: 0.8676482938097947
```

Figure 2.3.4.2 A Python Code snippet for the Optimized Model performance

Using optimized hyperparameters (max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=100) for the Random Forest Regressor significantly improved performance: R2 0.8710, MSE 1.3711, MAE 0.8241, Explained Variance 0.8711, and Adjusted R2 0.8676.

2.4 Advanced Question 2 Classification task

2.4.1 Labelling the target column

```python
bins = [0, 8, 16, 20]  # Define the bin edges
labels = [1, 2, 3]  # Define the numerical category labels
df_transformed['achievement_category'] = pd.cut(df_transformed['remainder__G3'], bins=bins, labels=labels, include_lowest=True)
df_transformed
```

	onehot__school_MS	onehot__schoolsup_yes	onehot__higher_yes	onehot__subject_Portuguese	remainder__age	remainder__Medu	remainder__Fedu	rem
0	0.0	1.0	1.0	0.0	18.0	4.0	4.0	
1	0.0	0.0	1.0	0.0	17.0	1.0	1.0	
2	0.0	1.0	1.0	0.0	15.0	1.0	1.0	
3	0.0	0.0	1.0	0.0	15.0	4.0	2.0	
4	0.0	0.0	1.0	0.0	16.0	3.0	3.0	
...	...	...	...	...	...	...	...	
1038	1.0	0.0	1.0	1.0	19.0	2.0	3.0	
1039	1.0	0.0	1.0	1.0	18.0	3.0	1.0	
1040	1.0	0.0	1.0	1.0	18.0	1.0	1.0	
1041	1.0	0.0	1.0	1.0	17.0	3.0	1.0	
1042	1.0	0.0	1.0	1.0	18.0	3.0	2.0	

1043 rows × 16 columns

Figure 2.4 A python code snippet to Show the Labelling of target columns

2.4.2 Factor analysis

```
fa2=FactorAnalyzer(n_factors=6)
fa2.fit(X1)
loadings=fa2.loadings_
factor_loading=pd.DataFrame(loadings,index=X1.columns)
factoredselected_columns = []

# Set a threshold for loading values
loading_threshold = 0.5

for factor in factor_loading.columns:
    factoredselected_columns.extend(factor_loading.index[factor_loading[factor] > loading_threshold])

factoredselected_columns = list(set(factoredselected_columns))
factoredselected_columns
X1=X1[factoredselected_columns]
```

Figure2.4.2 A python code for the Factors analysis

2.4.3 Scaling and Spilt data for training and Testing

```
columns_to_scale = X1.columns
# Create a ColumnTransformer
minmax= ColumnTransformer(transformers=[('scaler', MinMaxScaler(), columns_to_scale)],remainder='passthrough')
# Fit and transform the feature data
X2= minmax.fit_transform(X1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X2, Y1, test_size=0.3, random_state=42)
```

Figure 2.4.3 A python Code to scale and split data for training and testing

2.4.4 Training the model and performance indicators

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))

# Confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           1       0.80      0.80      0.80        49
           2       0.95      0.95      0.95       246
           3       0.83      0.83      0.83        18

    accuracy                           0.92       313
   macro avg       0.86      0.86      0.86       313
weighted avg       0.92      0.92      0.92       313

Confusion Matrix:
[[ 39  10   0]
 [ 10 233   3]
 [  0   3  15]]
```

Figure 2.4.4.1 A Python code for train the model and showing the performance matrix.

The Random Forest Classifier exhibits strong performance with an accuracy of 0.92. Precision, recall, and F1-score metrics indicate balanced performance across classes. The confusion matrix reveals minimal misclassifications, reinforcing the model's effectiveness in distinguishing between classes.

3 Literature Review

The Random Forest technique, originally developed in 2001, has evolved as a popular statistical learning method across various realms. This ensemble approach develops numerous decision trees on subsets of data and synthesizes predictions to boost accuracy and generalization (Breiman, 2001).

3.1 Data Preprocessing for the Random Forest

Effective data preprocessing is key to ensuring quality input data and improving Random Forest performance. Common preprocessing steps include handling duplicated rows, outlier detection, association analysis via chi-squared tests, and dimensionality reduction using factor analysis.

Handling duplicated rows. Identifying and removing duplicated input rows avoids biasing tree construction to repeated data points (Li C, 2019). Alternatively, Rana et al. (2015) proposes handling duplicates by modifying the bagging process to sample uniformly across equivalence groups. Both approaches help improve generalizability.

Outlier detection. Outlier data can skew decision tree splitting and predictions (Xu et al., 2018). Graph-based and distance-based outlier detection help identify anomalies prior to fitting the Random Forest model (Liu et al., 2017). The trees can then be built using the filtered input data for enhanced performance.

Chi-squared test. The chi-squared test filters features during preprocessing by removing variables not strongly associated with the target variable (Kuhn & Johnson, 2019). This shrinks the feature space fed into the Random Forest algorithm while retaining predictive signal.

Factor analysis. When confronted with high-dimensional data, factor analysis provides dimensionality reduction (Espadoto et al.,2019). Extracting the main latent factors in the dataset makes subsequent Random Forest modeling more computationally efficient.

3.2 Random Forest Algorithm

The Random Forest algorithm produces numerous decision trees on bootstrapped versions of the original dataset and then averages the outputs to reduce overfitting (Chi et al.,2020). Built-in randomness when splitting nodes provides greater stability compared to single decision tree models. Enhancements like weighted sampling and tree pruning help further improve accuracy (Favieiro et al.,2019). Work also adapts Random Forest for streaming data environments through online updating (Vassallo et al., 2020).

Performance Evaluation

Suitable metrics are vital for methodically gauging Random Forest regressor and classifier effectiveness over various contexts.

For regression tasks, key indicators comprise Mean Absolute Error (MAE), Mean Squared Error (MSE) and R-squared (R2) (Schonlau, et al., 2021). While MSE weights larger errors more heavily, MAE measures average magnitude discrepancies between predicted and actual values. R-squared quantifies the model’s success explaining variation(Iswaran et al, 2019).

For classification, applicable metrics include accuracy, precision, recall, Receiver Operating Curves (ROC), and Area Under the Curve (AUC) (Kuhn & Johnson, 2022). Precision and recall quantify the trade-off detecting true positives while limiting false alarms. ROC curves plot the signal-tonoise ratio as the classification threshold varies. AUC measures this across all thresholds (Masum et al.,2022).

3.3 Comparative Studies

Multiple studies demonstrate Random Forest advantages over other popular supervised learning models including logistic regression, support vector machines (SVM), neural networks, and basic decision trees. Schonlau et al. (2021) showed Random Forests yielded higher out-of-sample accuracy over alternative models across numerous open datasets. Similarly, Ramaswami et al. (2019) found Random Forests significantly outperformed SVM and regression approaches predicting student performance. Random Forests also prove more computationally efficient handling large feature spaces compared to SVM while avoiding neural network complexities (Xu et al., 2018). However, no single dominating algorithm exists across all data problem contexts (Kuhn & Johnson, 2022).

3.4 Future scope

Ongoing Random Forest research aims to enhance interpretability, adapt for distributed computing, and refine automatic hyperparameter optimization (Abd El-Ghany et al.,2023). Hybrid ensemble methods combining Random Forests with other learners are explored for additional performance gains. Real-world application viability hinges on factors such as data quality, size, and complexity.

3.5 Conclusion

In Conclusion, Random Forest provides versatile supervised learning potential across forecasting and classification situations. Proper data preparation, sound features, and tailored performance measures maximize its utility. Empirical comparisons affirm the approach remains highly competitive within the machine learning landscape.

4 References

Abd El-Ghany, S., & Abd El-Aziz, A. A. (2023). A Robust Tuned Random Forest Classifier Using Randomized Grid Search to Predict Coronary Artery Diseases. CMC-COMPUTERS MATERIALS & CONTINUA, 75(2), 4633-4648.

http://dx.doi.org/10.32604/cmc.2023.035779

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Cai, J., Xu, K., Zhu, Y., Hu, F., & Li, L. (2020). Prediction and analysis of net ecosystem carbon exchange based on gradient boosting regression and random forest. *Applied energy*, *262*, 114566.

https://doi.org/10.1016/j.apenergy.2020.114566

Espadoto, M., Martins, R. M., Kerren, A., Hirata, N. S., & Telea, A. C. (2019). Toward a quantitative survey of dimension reduction techniques. *IEEE transactions on visualization and computer graphics*, *27*(3), 2153-2173.

https://doi.org/10.1109/TVCG.2019.2944182

Favieiro, G. W., & Balbinot, A. (2019). Paraconsistent random forest: An alternative approach for dealing with uncertain data. IEEE Access, 7, 147914-147927.

https://doi.org/10.1109/ACCESS.2019.2946256

Ishwaran, H., & Lu, M. (2019). Standard errors and confidence intervals for variable importance in random forest regression, classification, and survival. *Statistics in medicine*, *38*(4), 558-582.

https://doi.org/10.1002/sim.7803

Kuhn, M., & Johnson, K. (2019). Feature engineering and selection: A practical approach for predictive models. Chapman and Hall/CRC.

https://doi.org/10.1201/9781315108230

Li, C. (2019). Preprocessing methods and pipelines of data mining: An overview. *arXiv preprint arXiv:1906.08510*.

https://doi.org/10.48550/arXiv.1906.08510

Liu, Huawen, Xuelong Li, Jiuyong Li, and Shichao Zhang. "Efficient outlier detection for high-dimensional data." *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48, no. 12 (2017): 2451-2461.

https://doi.org/10.1109/TSMC.2017.2718220

Masum, M., Faruk, M. J. H., Shahriar, H., Qian, K., Lo, D., & Adnan, M. I. (2022, January). Ransomware classification and detection with machine learning algorithms. In 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0316-0322). IEEE.

https://doi.org/10.1109/CCWC54503.2022.9720869

Rana, S., Gupta, S. K., & Venkatesh, S. (2015). Differentially private random forest with high utility. I*EEE International Conference on Data Mining* (pp. 955-960). IEEE.

https://doi.org/10.1109/ICDM.2015.76

Ramaswami, G., Susnjak, T., Mathrani, A., Lim, J. and Garcia, P. (2019), "Using educational data mining techniques to increase the prediction accuracy of student academic performance", Information and Learning Sciences, Vol. 120 No. 7/8, pp. 451-467.

https://doi.org/10.1108/ILS-03-2019-0017

Schonlau, M., & Zou, R. Y. (2020). The random forest algorithm for statistical learning. *The Stata Journal*, *20*(1), 3-29.

https://doi.org/10.1177/1536867X20909688

Vassallo, D., Krishnamurthy, R., Sherman, T., & Fernando, H. J. (2020). Analysis of random forest modeling strategies for multi-step wind speed forecasting. *Energies*, *13*(20), 5488.

https://doi.org/10.3390/en13205488

Xu, X., Liu, H., Li, L., & Yao, M. (2018). A comparison of outlier detection techniques for high-dimensional data. *International Journal of Computational Intelligence Systems*, *11*(1), 652.

Appendix 1

Appendix which must include the Python commands you used in your analysis. You should provide all the Python codes used for the assignment.

All plots, figures and graphs must be numbered and clearly labelled.

Appendix 1.1 Question 1 Code

#IMPORT RELEVANT PACKAGES:

```
import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

from scipy.stats import chi2_contingency

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split
```

read **the csv file seprated by the semicolun(;) then label the data for maths strudents do same for the portuguese students data set**

```
df1= pd.read_csv('student-mat.csv',sep=';')

df1['subject']= "Maths"

df2= pd.read_csv('student-por.csv',sep=';')

df2['subject']="Portuguese"

df=df1.merge(df2,how='outer',on=df1.columns.tolist())

df.sample(5)
```

 # SIZE OF THE DATA

```
df.size
```

 # NUMBERS OF ROWs

#rows

```
df.shape[0]
```

 # NUMBERS OF COLUMNS

#columns

```
df.shape[1]
```

#sample data of with all columns names

```
df.T.iloc[:,:10]
```

 # # Each columns datatype|

df.info()

check for **the missing value**

df.isna().sum()

 # # Find the duplicated rows according to Idenctical attributes of students

 # in This case maybe happend that same duplicated rows may have diifernt grades

duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus","Medu"," Fedu","Mjob","Fjob","reason","nursery","internet"],keep='first')

 # # Find the duplicated rows according to Idenctical attributes of students and grades

 # in This case maybe happend that same duplicated rows may have study differnt course maths or portuguese

duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus","Medu"," Fedu","Mjob","Fjob","reason","nursery","internet","G1","G2","G3"])

 # # Find the duplicated rows according to Idenctical attributes of students and grades aloge with subject name

duplicated_rows=df.duplicated(subset=["school","sex","age","address","famsize","Pstatus","Medu"," Fedu", "Mjob","Fjob","reason","nursery","internet","G1","G2","G3","subject"],keep='first')

df[duplicated_rows]

 # # remove the duplicated rows according to Idenctical attributes of students and grades aloge with subject name

df=df.drop_duplicates(subset=["school","sex","age","address","famsize","Pstatus","Medu","Fedu"," Mjob","Fjob","reason", "nursery","internet","G1","G2","subject"],keep='last')

df=df.reset_index(drop=True)

 # # unique values according the each columns

unique_counts = df.nunique()

 # # function to plot the boxplot and histogram

 # This function plot the boxplot and histogram of columns which have more then 6 unique values

```
def boxanddistplot(df):
    unique_counts = df.nunique()
    selected_columns = unique_counts[unique_counts >= 6].index.tolist()
    for column_name in selected_columns:
        plt.figure(figsize=(15, 6))
        plt.subplot(1, 2, 1)
```

```
        sns.boxplot(x=column_name,data=df)

        plt.title(f"box Plot of {column_name}",fontsize=16,loc='center')

        plt.xlabel(f"{column_name}")

        plt.subplot(1,2,2)

        sns.histplot(df[column_name], kde=True,bins=df[column_name].nunique()
,kde_kws=dict(cut=3))

        plt.xlabel(f"{column_name}")

        plt.ylabel('no of students')

        plt.title(f" Histogram of {column_name}",fontsize=16,loc='center')

        plt.show()
```

 # # box plot and histogram before outlier capping

```
boxanddistplot(df)

df.describe()
```

 # # outlier capping using IQR method

```
unique_counts = df.nunique()

selected_columns = unique_counts[unique_counts >= 6].index.tolist()

for columns1 in selected_columns:

    percentile25 = df[columns1].quantile(0.25)

    percentile75 = df[columns1].quantile(0.75)

    iqr = percentile75 - percentile25

    upper_limit = percentile75 + 1.5 * iqr

    lower_limit = percentile25 - 1.5 * iqr

    df[columns1] = np.where(

    df[columns1] > upper_limit,upper_limit,np.where(df[columns1] <
lower_limit,lower_limit,df[columns1]))
```

 # # box plot and histogram after outlier capping

```
boxanddistplot(df)

df.describe()
```

 # # Function to plot the countplots, piechart and boxplot

 # this function plot the countsplot,piechart and boxplot of those columns whose unique values is less then 6 and giving the appropiate title of the each plot

```
def plot_categorical_columns(df):
```

```
# Get the list of categorical columns in the DataFrame
dfcopy=df.copy(deep=True)
dfcopy.columns= ["school","sex","age", "home address type","family size","parent's cohabitation status","mother's education",\
"father's education","mother's job","father's job","reason to choose this school","student's guardian",\
"home to school travel time","weekly study time","number of past class failures","extra educational support",\
"family educational support","extra paid classes within the course subject","extra-curricular activities",\
"attended nursery school","wants to take higher education","Internet access at home", "with a romantic relationship",\
"quality of family relationships","free time after school","going out with friends","workday alcohol consumption",\
"weekend alcohol consumption","current health status","number of school absences","first period grade", "second period grade","final grade","subject of course"]
unique_counts = dfcopy.nunique()
selected_columns = unique_counts[unique_counts <= 6].index.tolist()


# Iterate over each categorical column and plot countplot and pie chart
for column_name  in  selected_columns:
    plot_allplot(dfcopy, column_name,xlabel=column_name)
    plt.figure(figsize=(15, 5))
    sns.boxplot(x=column_name, y='final grade', data=dfcopy)
    plt.title(f"box Plot of final grade Variable by {column_name}",fontsize=16,loc='center')
    plt.xlabel(f"{column_name}")
    plt.ylabel('final grades')
    plt.show()
def plot_allplot(df, column_name,xlabel=None,ylabel="No of students",title=None):
    if column_name not in df.columns:
        print(f"Column '{column_name}' not found in the DataFrame.")
        return
    fig=plt.figure(figsize=(15, 5))
```

```
    if title is None:
        title=f"Distribution of Students according to {column_name}"
        fig.suptitle(title,fontsize=19)
    else:
        fig.suptitle(title,fontsize=20)
    plt.subplot(1, 2, 1)
    df[column_name].astype('category')
    order = df[column_name].value_counts().index
    ax=sns.countplot(x=column_name, data=df,order=order,palette="viridis")
    if xlabel is not None:
        plt.xlabel(xlabel,fontsize=10)
    if ylabel is not None:
        plt.ylabel(ylabel,fontsize=10)
    handles, labels = [], []
    for i, category in enumerate(order):
        count = df[column_name].value_counts()[category]
        handles.append(plt.Line2D([0], [0], marker='o', color=sns.color_palette("viridis")[i],
markersize=8))
        labels.append(f'{category} ({count})')
    plt.legend(handles, labels, title=f'{column_name} categories', loc='center left', bbox_to_anchor=(1,
0.1))
    plt.subplot(1, 2, 2)
    category_counts = df[column_name].value_counts()
    plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=0)
    plt.legend(category_counts.index, title=f'{column_name} categories', loc='center left',
bbox_to_anchor=(1, 0.1))
    plt.tight_layout()
    plt.show()
plot_categorical_columns(df)
```

 # # function to check association with final grades

 # this function check the association with final grades using the chi square test if p value is less than 0.05 then it store the value in the one list and if unique values of that columns is less than 6 then it plot the scatter plot with final grade(G3).

```
def chi_square_association(df, categorical_column, output_column):
    contingency_table = pd.crosstab(df[categorical_column], df[output_column])
    associtated=[]
    BOLD = '\033[1m'
    END_BOLD = '\033[0m'
    chi2, p, _, _ = chi2_contingency(contingency_table)
    print(f"{BOLD}column:{categorical_column}{END_BOLD}")
    print(f"Chi-Square Statistic: {chi2}")
    print(f"{BOLD}P-value: {p}{END_BOLD}")
    alpha = 0.05
    print(f"Significance Level: {alpha}")
    print(f"Degrees of Freedom: {(contingency_table.shape[0] - 1) * (contingency_table.shape[1] -
1)}")
    if p < alpha:
        print(f"{BOLD}There is asignificant association between '{categorical_column}' and
'{output_column}{END_BOLD}'.")
        print("   ")
        return categorical_column
    else:
        print(f"There is no significant association between the variables.")
        print("   ")
        return None
def check_association(df,column_name11):
    columns=[]
    for column_name in df.columns:
        result=chi_square_association(df, column_name, output_column=column_name11)
        if  result is not None:
            columns.append(result)
    print("Columns with significant association:")
    print(columns)
```

```
    for associatoncolums in columns:
        unique_values= df[associatoncolums].nunique()
        if unique_values >= 6:
            plt.figure(figsize=(7, 5))
            sns.scatterplot(x=associatoncolums, y=column_name11, data=df)
            plt.title(f"scatterplot  of final grade and {associatoncolums}",fontsize=16,loc='center')
            plt.xlabel(f"{associatoncolums}")
            plt.ylabel('final grades')
            plt.show()
    return columns
#check the association
mycolumns=check_association(df,'G3')
```

Appendix 1.2 Advanced Question 1 Code

 # # Factor Analysis

#create the new dataframe with all columns who have association wuth final grades

```
newdf=df[mycolumns]
newdf=newdf.drop(columns=['address','Mjob','romantic'])
object_cols = newdf.select_dtypes(include=['object']).columns.tolist() #SELECT THE COLUMNS WHOSE HAVE DATATYPE OBJECT
```

 # # ONEHOT ENCODING

```
preprocessor = ColumnTransformer(transformers=[('onehot', OneHotEncoder(drop='first'), object_cols)],remainder='passthrough')#ONEHOT ENCODING OF OBJECT DATATPE COLUMMNS
```

 # # CREATE THE NEW DATAFRAME WITH ENCODED COLUMNS

```
df_transformed = pd.DataFrame(preprocessor.fit_transform(newdf), columns=preprocessor.get_feature_names_out(newdf.columns))
```

 # # Split the data for the input and target columns

```
X=df_transformed.drop(columns=['remainder__G3']) #INPUT COLUMNS (FEATURES)
Y=df_transformed['remainder__G3'] #target column
```

 # # Factor analysis with no of factors one less then no of columns and eigenvalues

#import the factoranalyer

```
from factor_analyzer import FactorAnalyzer
fa=FactorAnalyzer(n_factors=X.shape[1]-1)
```

```
fa.fit(X)

eigenvector, value= fa.get_eigenvalues()

# # plot the scree plot to select the no of factors

plt.scatter(range(1,X.shape[1]+1),eigenvector)

plt.plot(range(1,X.shape[1]+1),eigenvector)

plt.grid(True)

plt.xlabel(' no of Factors')

plt.ylabel('Eigen values')

plt.title("Scree plot of student's data")

plt.show()

# # factor analysis with 6 factors selected from the scree plot

fa1=FactorAnalyzer(n_factors=6)

fa1.fit(X)

loadings=fa1.loadings_

factor_loading=pd.DataFrame(loadings,index=X.columns

# # select the columns whose loading value is greater than 0.5 for the factors

factoredselected_columns = []

# Set a threshold for loading values

loading_threshold = 0.5

for factor in factor_loading.columns:

    factoredselected_columns.extend(factor_loading.index[factor_loading[factor] > loading_threshold])

factoredselected_columns = list(set(factoredselected_columns))

factoredselected_columns

X=X[factoredselected_columns]
```

Appendix 1.3 Question 2 Code

```
# # SCALING THE DATA and Split data for training and testing

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

columns_to_scale = X.columns

# Create a ColumnTransformer
```

minmax= ColumnTransformer(transformers=[('scaler', MinMaxScaler(), columns_to_scale)],remainder='passthrough')

 # Fit and transform the feature data

X2= minmax.fit_transform(X)

 # Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X2, Y, test_size=0.3, random_state=42)

 # # train the Model and performence metrices

from sklearn.ensemble import RandomForestRegressor

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

 # Create the model

rf_model = RandomForestRegressor(n_estimators=300, random_state=42)

 # Train the model

rf_model.fit(X_train, y_train)

 # Make predictions

y_pred = rf_model.predict(X_test)

Evaluate the model **using different metrics**

r_squared = r2_score(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

explained_var = explained_variance_score(y_test, y_pred)

n = len(y_test)

k = X.shape[1] # Number of predictors

adjusted_r_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

 # Print the performance metrics

print(f'R-squared: {r_squared}')

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

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Explained Variance Score: {explained_var}')

print(f'Adjusted R-squared: {adjusted_r_squared}')

```
# # Feature Importance Plot
feature_importances = rf_model.feature_importances_
# Create a DataFrame with feature names and their importance scores
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances,
})
# Sort the DataFrame by importance values in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
# Plot the feature importance graph
plt.figure(figsize=(10, 6))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Random Forest Regressor - Feature Importance')
plt.xticks(rotation=45)
plt.show()
# # parameter Optimazation for batter performance
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100,200,250,300,350,500],
    'max_depth': [None, 10, 20,12,13],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=10, scoring='r2')
grid_search.fit(X_train, y_train)
best_rf = grid_search.best_estimator_
best_rf.fit(X_train, y_train)
```

```
y_pred = best_rf.predict(X_test)

r_squared = r2_score(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

explained_var = explained_variance_score(y_test, y_pred)

n = len(y_test)

k = X.shape[1]  # Number of predictors

adjusted_r_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

# Print the performance metrics

print(f'Optimized R-squared: {r_squared}')

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

print(f'Optimized Mean Absolute Error (MAE): {mae}')

print(f'OptimizedExplained Variance Score: {explained_var}')

print(f'OptimizedAdjusted R-squared: {adjusted_r_squared}')
```

Appendix 1.4 Advanced Question 2 Code

```
# # Classification Model

# # labeling the data

# 1 for poor grades( 0 -8)

# 2 for avarage grades(8 -16)

# 3 for the good grades(16-20)

bins = [0, 8, 16, 20]  # Define the bin edges

labels = [1, 2, 3]  # Define the numerical category labels

df_transformed['achievement_category'] = pd.cut(df_transformed['remainder__G3'], bins=bins,
labels=labels, include_lowest=True)

# # Split the data for the input and target columns

X1=df_transformed.drop(columns=['remainder__G3','achievement_category'])

Y1=df_transformed['achievement_category']

# # Factor analysis for the classifaction task

fa2=FactorAnalyzer(n_factors=6)

fa2.fit(X1)

loadings=fa2.loadings_

factor_loading=pd.DataFrame(loadings,index=X1.columns)
```

factoredselected_columns = []

 # Set a threshold for loading values

loading_threshold = 0.5

for factor in factor_loading.columns:

    factoredselected_columns.extend(factor_loading.index[factor_loading[factor] > loading_threshold])

factoredselected_columns = list(set(factoredselected_columns))

factoredselected_columns

X1=X1[factoredselected_columns]

 # # scaling and split data for the training and testing

columns_to_scale = X1.columns

 # Create a ColumnTransformer

minmax= ColumnTransformer(transformers=[('scaler', MinMaxScaler(), columns_to_scale)],remainder='passthrough')

 # Fit and transform the feature data

X2= minmax.fit_transform(X1)

 # Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X2, Y1, test_size=0.3, random_state=42)

 # # TRAIN THE MODEL AND MAKE PREDICTION

#import the RandomfoestClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

rf_model = RandomForestClassifier(n_estimators=100, random_state=42**)**

 # Train the model

rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

 # # PERFORMANCE METRIC AND ACCURACY

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')

 # Classification report

print('Classification Report:')

print(classification_report(y_test, y_pred))

```
# Confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
# # parameters for the  Optimized model
from sklearn.model_selection import GridSearchCV
# Define the parameter grid
param_grid = {
   'n_estimators': [50, 100, 200,300],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]
}
# Create the RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
# Create GridSearchCV
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
# Fit the model to the data
grid_search.fit(X_train, y_train)
# Print the best parameters
print("Best Parameters:", grid_search.best_params_)
best_rf_model = grid_search.best_estimator_
# Make predictions on the test set using the best model
y_pred = best_rf_model.predict(X_test)
# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Print other matrix
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```