

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
```

✓ Importing Data From Drive

```
from google.colab import drive
drive.mount('/content/drive')

data_path = '/content/drive/My Drive/SchoolManagementDataset/student_data.csv'
```

Mounted at /content/drive

```
df = pd.read_csv(data_path)
```

```
df.head().T
```

	0	1	2	3	4
school	GP	GP	GP	GP	GP
sex	F	F	F	F	F
age	18	17	15	15	16
address	U	U	U	U	U
famsize	GT3	GT3	LE3	GT3	GT3
Pstatus	A	T	T	T	T
Medu	4	1	1	4	3
Fedu	4	1	1	2	3
Mjob	at_home	at_home	at_home	health	other
Fjob	teacher	other	other	services	other
reason	course	course	other	home	home
guardian	mother	father	mother	mother	father
traveltime	2	1	1	1	1
studytime	2	2	2	3	2
failures	0	0	3	0	0
schoolsup	yes	no	yes	no	no
famsup	no	yes	no	yes	yes
paid	no	no	yes	yes	yes
activities	no	no	no	yes	no
nursery	yes	no	yes	yes	yes
higher	yes	yes	yes	yes	yes
internet	no	yes	yes	yes	no
romantic	no	no	no	yes	no
famrel	4	5	4	3	4
freetime	3	3	3	2	3
goout	4	3	2	2	2
Dalc	1	1	2	1	1
Walc	1	1	3	1	2
health	3	3	3	5	5
absences	6	4	10	2	4
G1	5	5	7	15	6
G2	6	5	8	14	10
G3	6	6	10	15	10

✓ Description of Column Names

Variable	Description
school	Student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
sex	Student's sex (binary: 'F' - female or 'M' - male)
age	Student's age (numeric: from 15 to 22)
address	Student's home address type (binary: 'U' - urban or 'R' - rural)
famsize	Family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
Pstatus	Parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
Medu	Mother's education (numeric: 0 - none, 1 - primary education, 2 - 5th to 9th grade, 3 - secondary education, 4 - higher education)
Fedu	Father's education (numeric: 0 - none, 1 - primary education, 2 - 5th to 9th grade, 3 - secondary education, 4 - higher education)
Mjob	Mother's job (nominal: 'teacher', 'health' care related, civil 'services', 'at_home' or 'other')
Fjob	Father's job (nominal: 'teacher', 'health' care related, civil 'services', 'at_home' or 'other')
reason	Reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')
guardian	Student's guardian (nominal: 'mother', 'father' or 'other')

Variable	Description
traveltime	Home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
studytime	Weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
failures	Number of past class failures (numeric: n if 1<=n<3, else 4)
schoolsup	Extra educational support (binary: yes or no)
famsup	Family educational support (binary: yes or no)
paid	Extra paid classes within the course subject (Portuguese) (binary: yes or no)
activities	Extra-curricular activities (binary: yes or no)
nursery	Attended nursery school (binary: yes or no)
higher	Wants to take higher education (binary: yes or no)
internet	Internet access at home (binary: yes or no)
romantic	With a romantic relationship (binary: yes or no)
famrel	Quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
freetime	Free time after school (numeric: from 1 - very low to 5 - very high)
goout	Going out with friends (numeric: from 1 - very low to 5 - very high)
Dalc	Workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
Walc	Weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
health	Current health status (numeric: from 1 - very bad to 5 - very good)
absences	Number of school absences (numeric: from 0 to 93)
G1	First period grade (numeric: from 0 to 20)
G2	Second period grade (numeric: from 0 to 20)
G3	Final grade (numeric: from 0 to 20, output target)

```
df.info()
```

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

```
df.describe()
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel
count	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000
mean	16.696203	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304
std	1.276043	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000

```
new_column_names = {
    'Overall Health': 'overall_health',
    'Time Productivity': 'time_productivity',
}

# Use the rename method of DataFrame to rename the columns
df = df.rename(columns=new_column_names)
```

✓ Exploratory Data Analysis (EDA)

```
df['address'].value_counts()
```

```
address
U    307
R     88
Name: count, dtype: int64
```

```
df['sex'].value_counts()
```

```
sex
F    208
M    187
Name: count, dtype: int64
```

```
df['internet'].value_counts()
```

```
internet
yes    329
no     66
Name: count, dtype: int64
```

```
cols_string = df.columns
col_counts = dict(df.count(axis=0))
col_types = {c: d for c, d in dict(df.dtypes).items() if col_counts[c] > 10 }

print(col_counts)
```

```
{'school': 395, 'sex': 395, 'age': 395, 'address': 395, 'famsize': 395, 'Pstatus': 395, 'Medu': 395, 'Fedu': 395, 'Mjob': 395, 'Fjob': 3
```

```
dict(df.dtypes).items()
```

```
dict_items([('school', dtype('O')), ('sex', dtype('O')), ('age', dtype('int64')), ('address', dtype('O')), ('famsize', dtype('O')), ('Pstatus', dtype('O')), ('Medu', dtype('int64')), ('Fedu', dtype('int64')), ('Mjob', dtype('O')), ('Fjob', dtype('O')), ('reason', dtype('O')), ('guardian', dtype('O')), ('traveltime', dtype('int64')), ('studytime', dtype('int64')), ('failures', dtype('int64')), ('schoolsup', dtype('O')), ('famsup', dtype('O')), ('paid', dtype('O')), ('activities', dtype('O')), ('nursery', dtype('O')), ('higher', dtype('O')), ('internet', dtype('O')), ('romantic', dtype('O')), ('famrel', dtype('int64')), ('freetime', dtype('int64')), ('goout', dtype('int64')), ('Dalc', dtype('int64')), ('Walc', dtype('int64')), ('health', dtype('int64')), ('absences', dtype('int64')), ('G1', dtype('int64')), ('G2', dtype('int64')), ('G3', dtype('int64'))])
```

✓ Plotting Data Distributions

```
#counting non-null values in column
col_counts = dict(df.count(axis=0))

#create a dictionary that contains column names as keys and data types as values, but only for columns with more than 10 non-null values.
col_types = {c: d for c, d in dict(df.dtypes).items() if col_counts[c] > 10 }

# separating numerical and categorical columns
_is_num = lambda x: x in (np.int64, float)
cols_numerical = sorted([col for col, dtp in col_types.items() if _is_num(dtp)])
cols_string = sorted([col for col, dtp in col_types.items() if not _is_num(dtp)])
print(f"NUMERICAL: {cols_numerical}")
print(f"CATEGORICAL: {cols_string}")

NUMERICAL: ['Dalc', 'Fedu', 'G1', 'G2', 'G3', 'Medu', 'Walc', 'absences', 'age', 'failures', 'famrel', 'freetime', 'goout', 'health', 's
CATEGORICAL: ['Fjob', 'Mjob', 'Pstatus', 'activities', 'address', 'famsize', 'famsup', 'guardian', 'higher', 'internet', 'nursery', 'pai
```

```
# Aesthetic configuration for seaborn and matplotlib
```

```
plt.rcParams["font.family"] = "monospace"
sns.set_theme(style='whitegrid', palette='muted')
```

✓ String Data Distributions

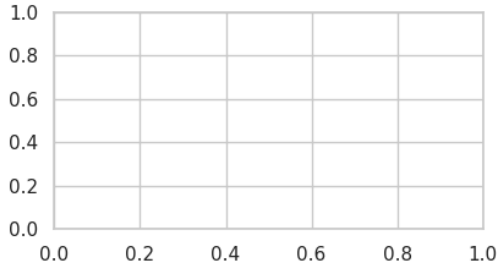
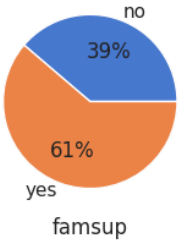
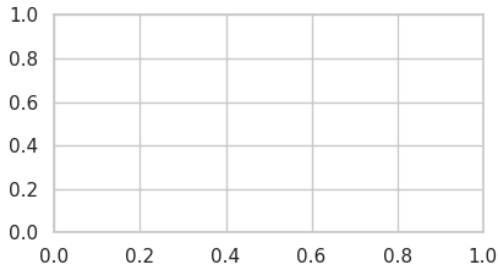
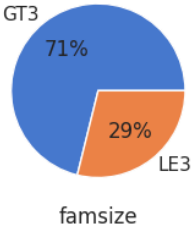
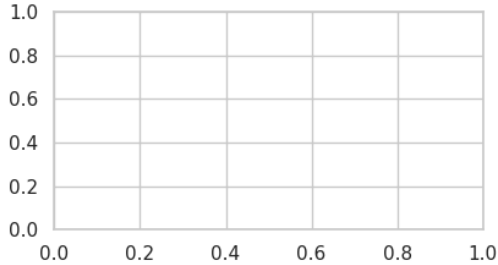
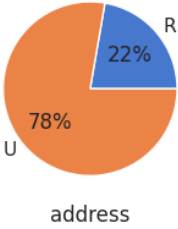
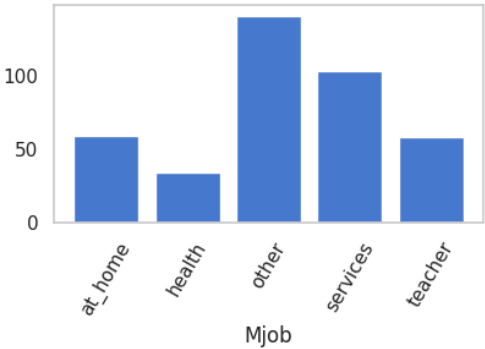
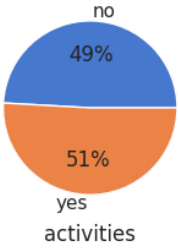
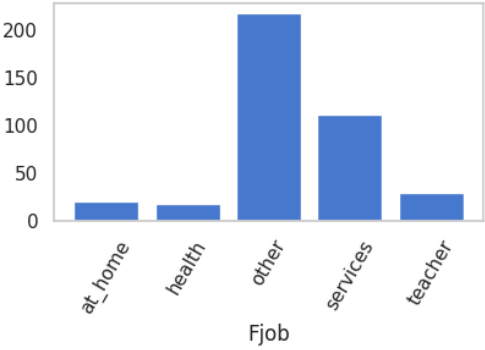
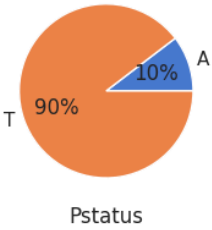
```
nb_pies = sum([len(df[c].dropna().unique()) < 5 for c in cols_string]) # is the maximum of the number of columns with less than 5 unique val
nb_rows = max([nb_pies, len(cols_string) - nb_pies])
print(f"{nb_pies=} | {nb_rows=}")

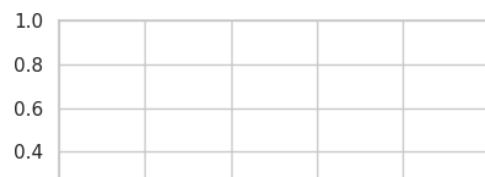
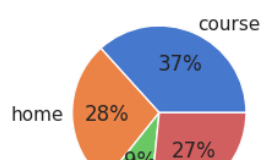
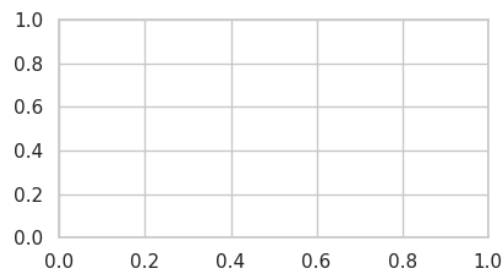
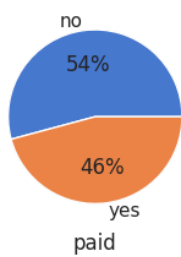
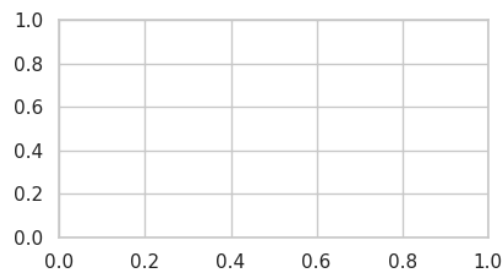
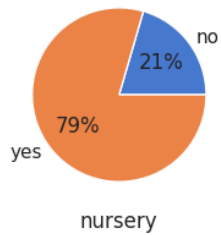
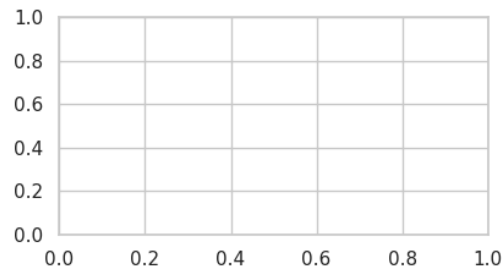
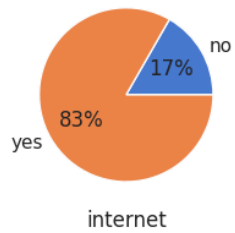
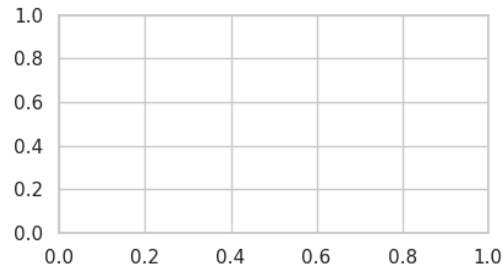
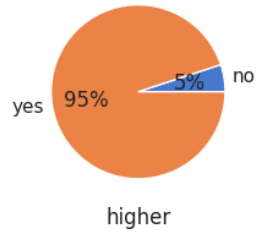
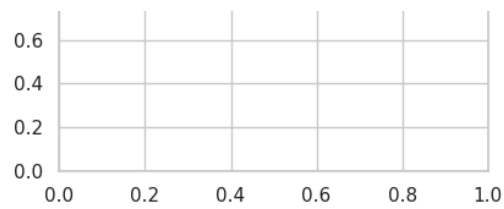
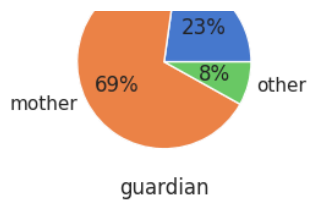
fig, axarr = plt.subplots(ncols=2, nrows=nb_rows, figsize=(8, 3 * nb_rows))
i_0, i_1 = 0, 0
for col in cols_string:
    vals = df[col].values
    vals = vals[~pd.isnull(vals)] # filtered to remove any missing values.
    unique_values, counts = np.unique(vals, return_counts=True) # contain the unique values and counts of the column, respectively.

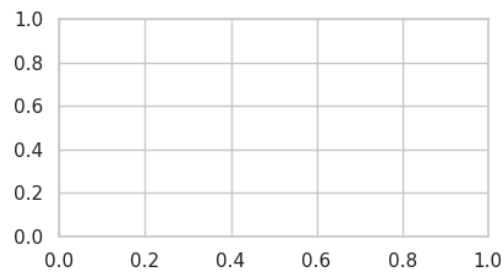
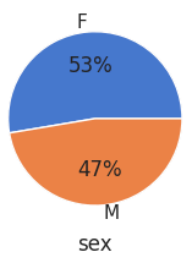
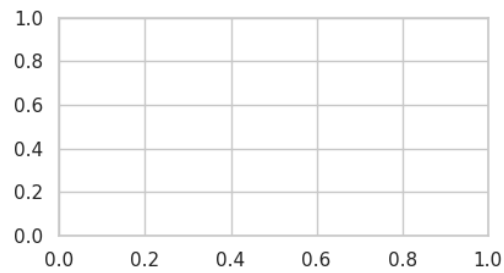
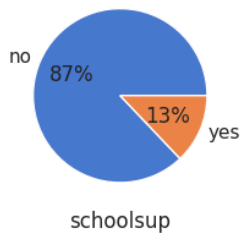
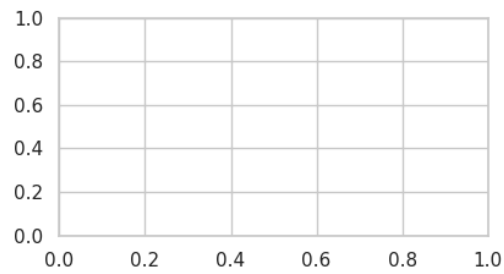
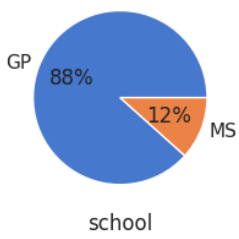
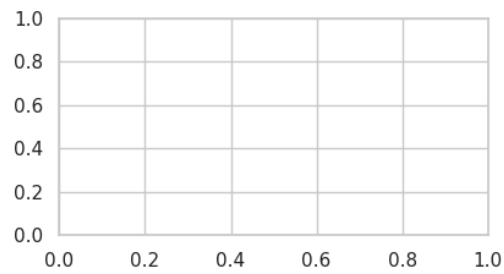
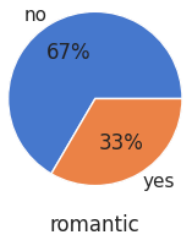
    if len(unique_values) < 5:
        axarr[i_0, 0].pie(counts, labels=unique_values, autopct='%.0f%%')
        axarr[i_0, 0].set_xlabel(col)
        i_0 += 1

    else:
        axarr[i_1, 1].bar(unique_values, counts)
        axarr[i_1, 1].set_xlabel(col)
        axarr[i_1, 1].grid()
        axarr[i_1, 1].tick_params(axis='x', labelrotation=60)
        #plt.xticks(rotation=45, ha='center')
        i_1 += 1
fig.tight_layout()
```

nb_pies=15 | nb_rows=15



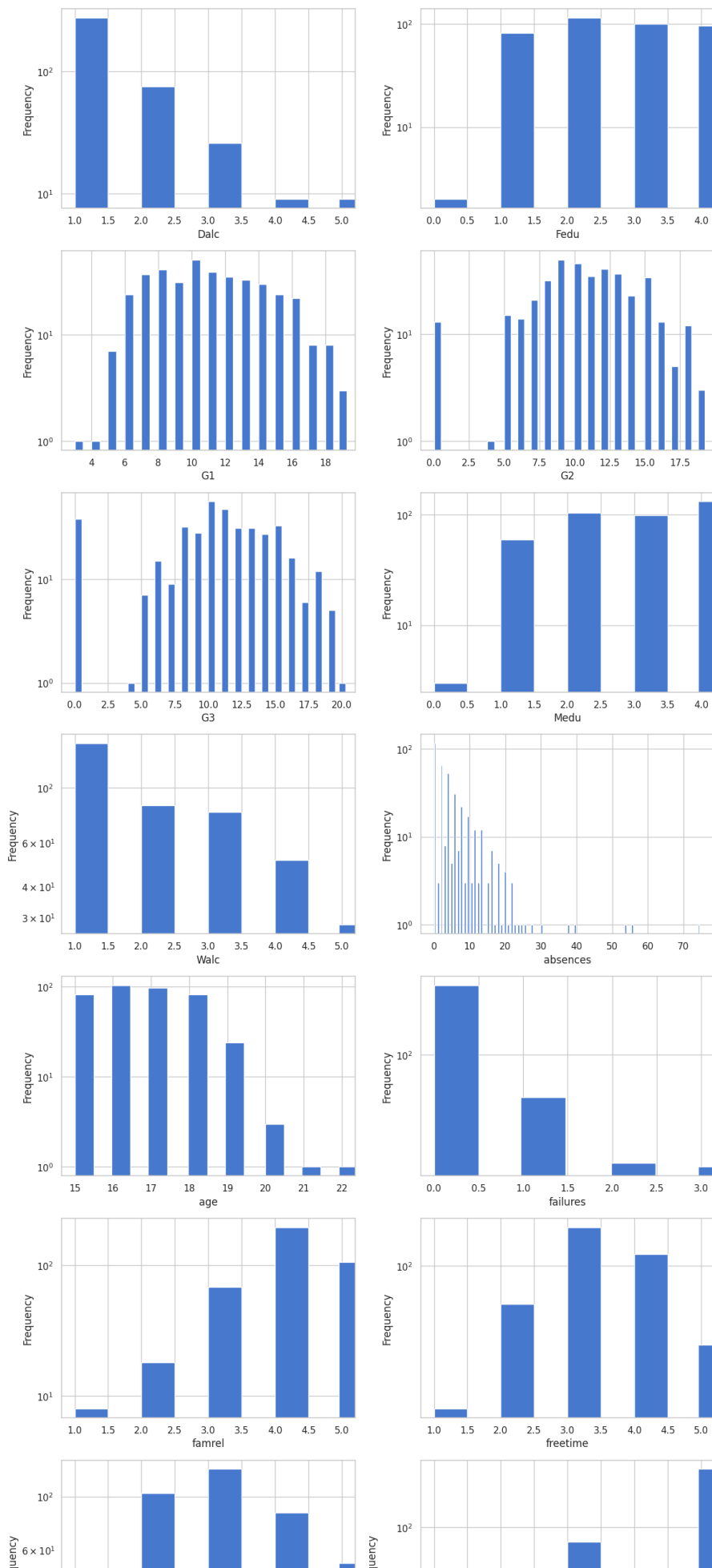


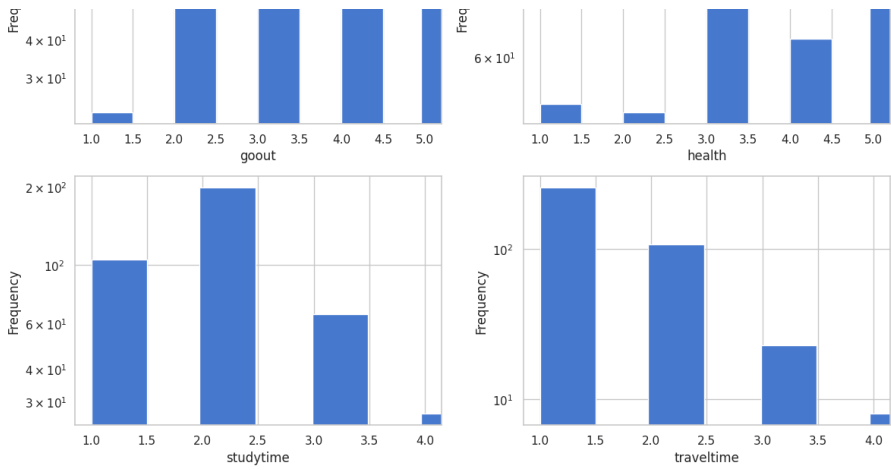


✓ numerical data distributions

```
import math

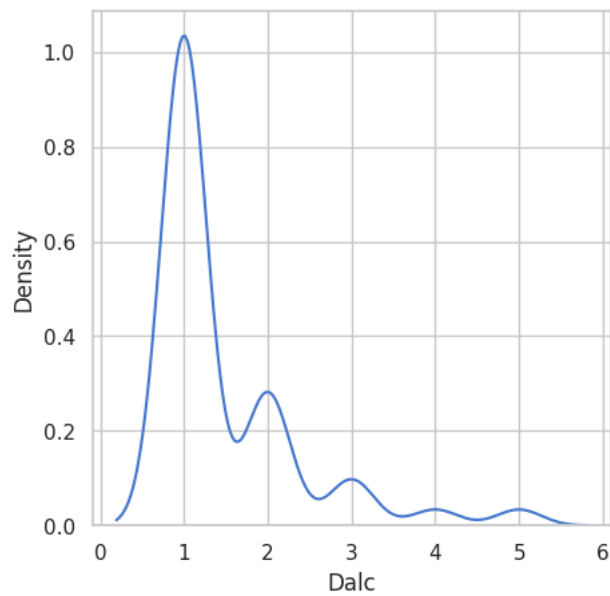
num_of_rows = math.ceil(len(cols_numerical) / 2)
fig, axarr = plt.subplots(ncols=2, nrows=num_of_rows, figsize=(12, 4 * num_of_rows))
for i, col in enumerate(cols_numerical):
    ax = axarr[i // 2, i % 2] # represents one subplot in the grid.
    # i // 2 and i % 2 expressions are used to calculate the row and column index of the subplot for the current column.
    df[col].plot.hist(ax=ax, bins=80, logy=True, xlabel=col, grid=True, width = 0.5)
fig.tight_layout() # to ensure that all of the subplots are properly spaced.
```



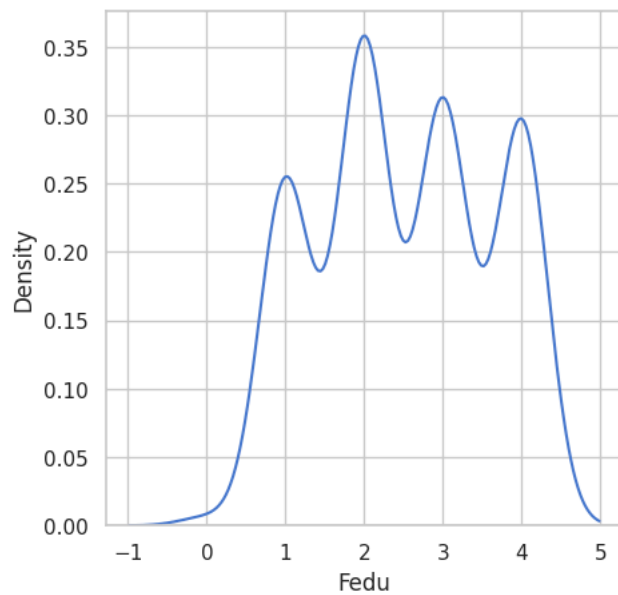


✓ Plotting KDE Charts For Numerical Data

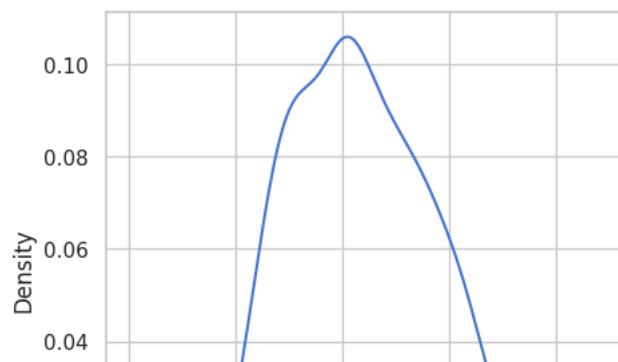
```
def hist_plot(col_2):  
    plt.figure(figsize=(5,5))  
    sns.kdeplot(data=df, x=col_2)  
  
    plt.show()  
    print(col_2, "\n", df[col_2].value_counts())  
# Print  
for i in cols_numerical:  
    hist_plot(i)
```

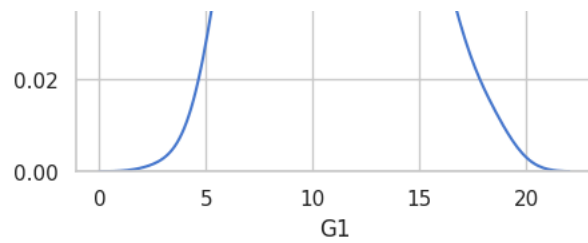


```
Dalc
Dalc
1    276
2     75
3     26
5      9
4      9
Name: count, dtype: int64
```

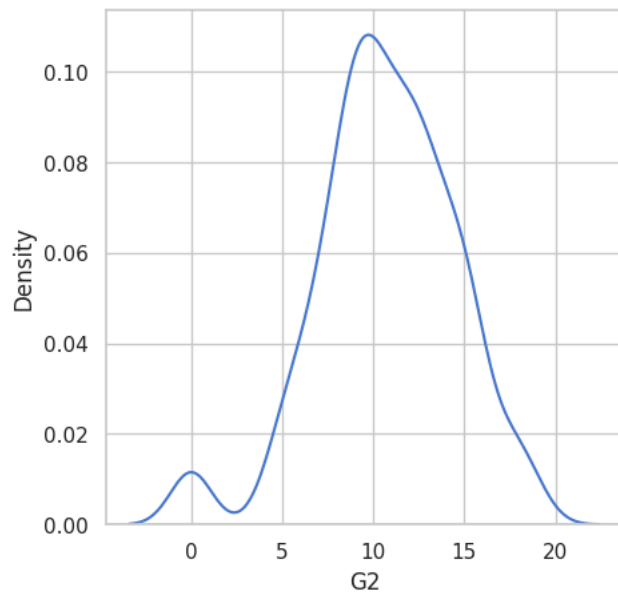


```
Fedu
Fedu
2    115
3    100
4     96
1     82
0       2
Name: count, dtype: int64
```

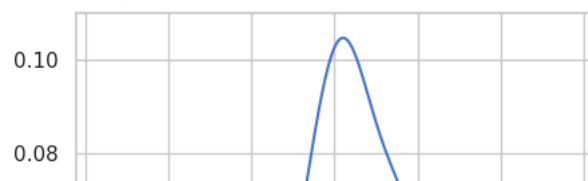


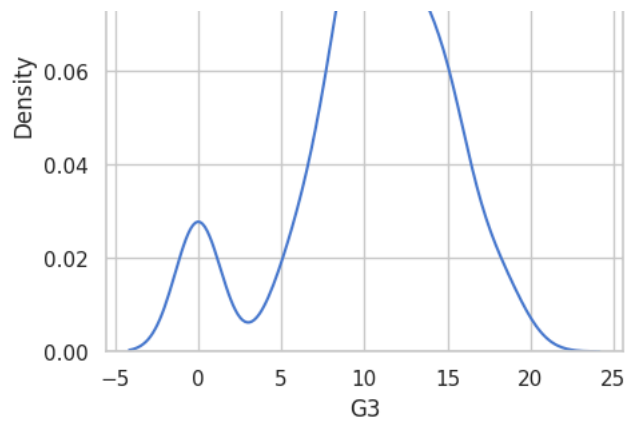


```
G1
G1
10  51
8   41
11  39
7   37
12  35
13  33
9   31
14  30
15  24
6   24
16  22
18   8
17   8
5    7
19   3
4    1
3    1
Name: count, dtype: int64
```

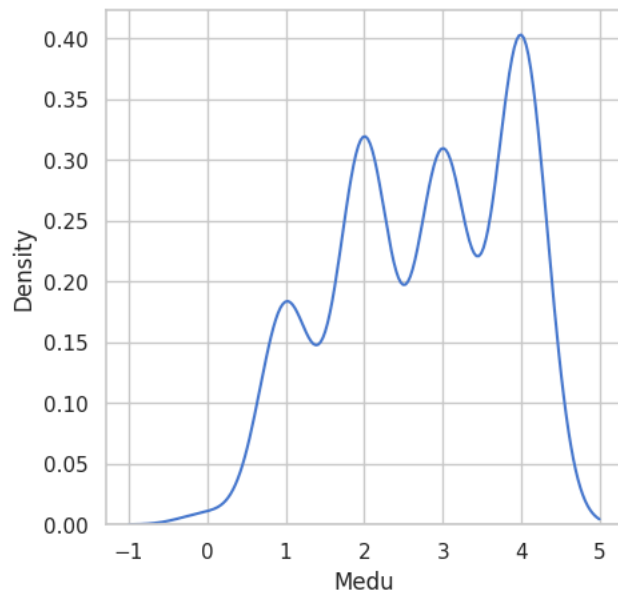


```
G2
G2
9   50
10  46
12  41
13  37
11  35
15  34
8   32
14  23
7   21
5   15
6   14
16  13
0   13
18  12
17   5
19   3
4    1
Name: count, dtype: int64
```

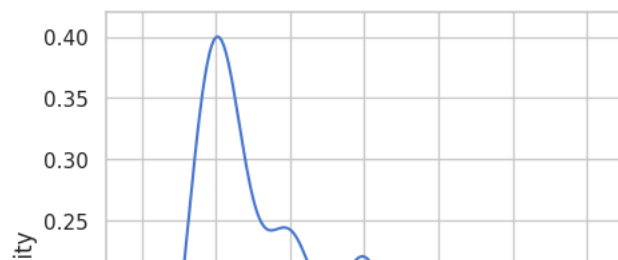


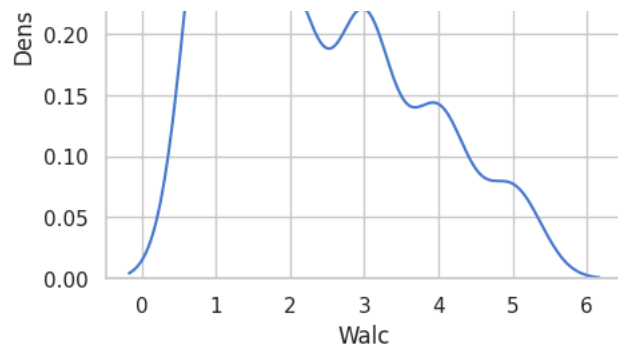


```
G3
G3
10  56
11  47
0   38
15  33
8   32
13  31
12  31
9   28
14  27
16  16
6   15
18  12
7   9
5   7
17  6
19  5
20  1
4   1
Name: count, dtype: int64
```

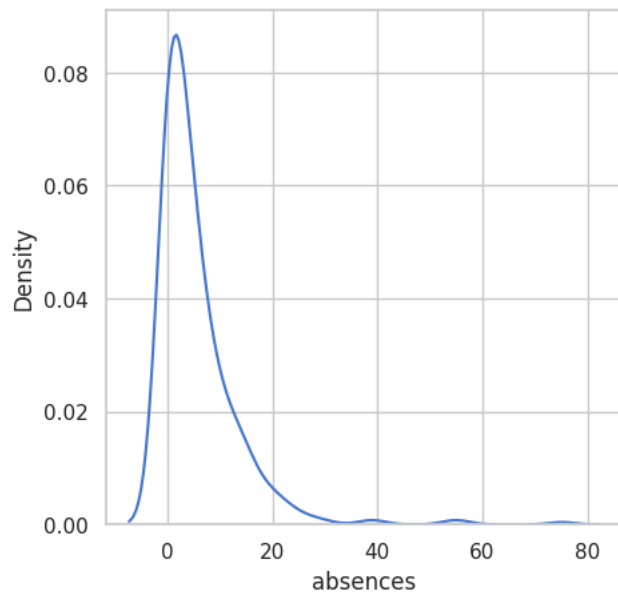


```
Medu
Medu
4   131
2   103
3    99
1    59
0     3
Name: count, dtype: int64
```





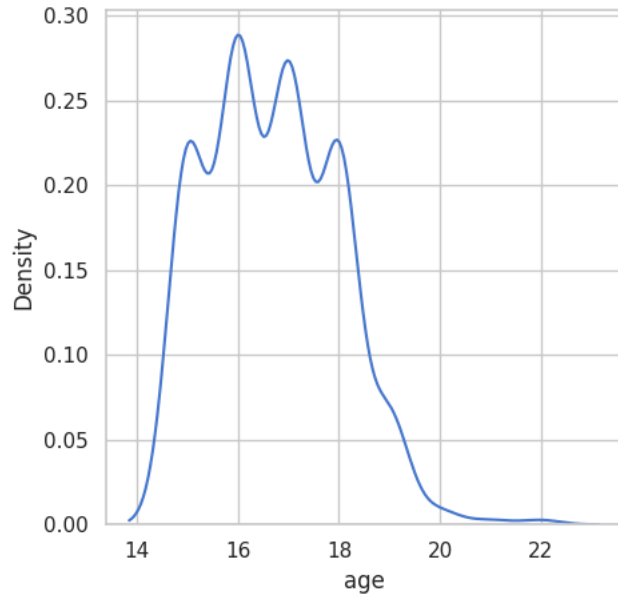
```
Walc
Walc
1    151
2     85
3     80
4     51
5     28
Name: count, dtype: int64
```



```
absences
absences
0     115
2      65
4      53
6      31
8      22
10     17
12     12
14     12
16      8
18      7
20      7
22      5
24      5
26      4
28      3
30      3
32      3
34      3
36      3
38      3
40      3
42      3
44      3
46      3
48      3
50      3
52      3
54      3
56      3
58      3
60      3
62      3
64      3
66      3
68      3
70      3
72      3
74      3
76      3
78      3
80      3
```



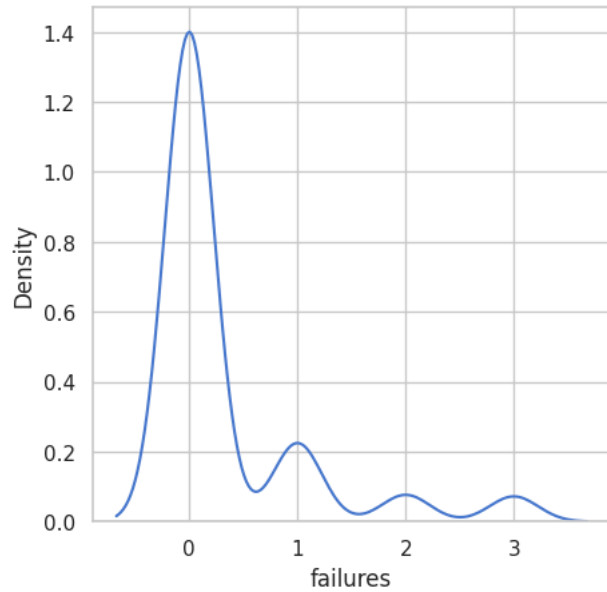
```
1/1
Name: count, dtype: int64
```



```
age
age
```

```
16    104
17     98
18     82
15     82
19     24
20      3
22      1
21      1
```

```
Name: count, dtype: int64
```

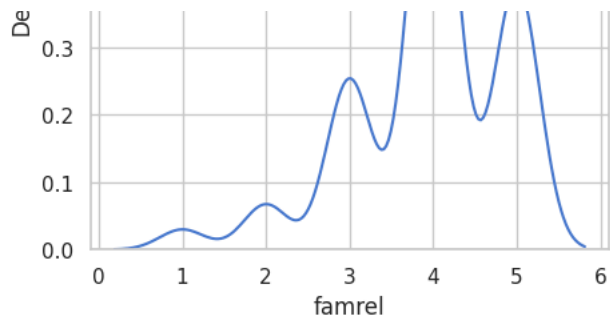


```
failures
failures
```

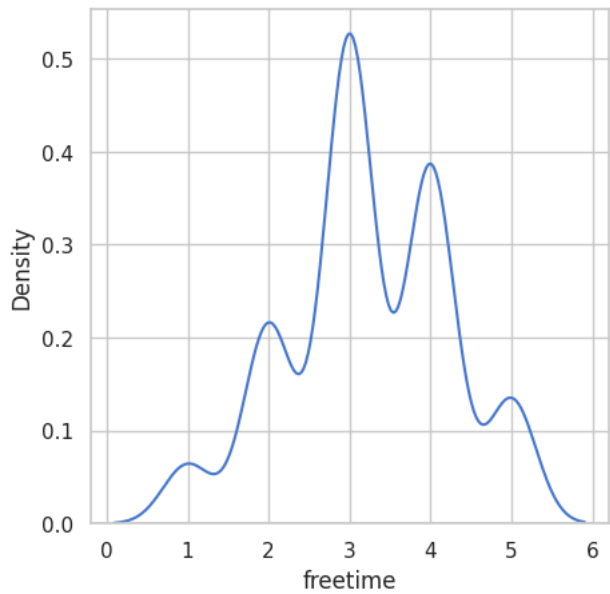
```
0    312
1     50
2     17
3     16
```

```
Name: count, dtype: int64
```

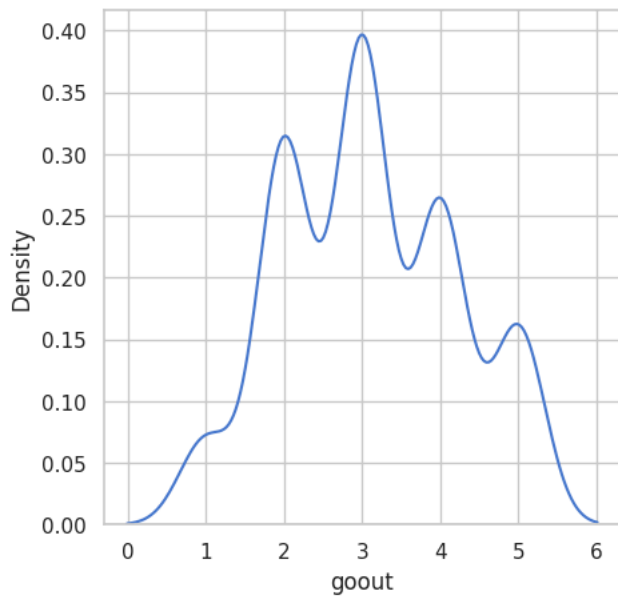




```
famrel
famrel
4    195
5    106
3     68
2     18
1      8
Name: count, dtype: int64
```



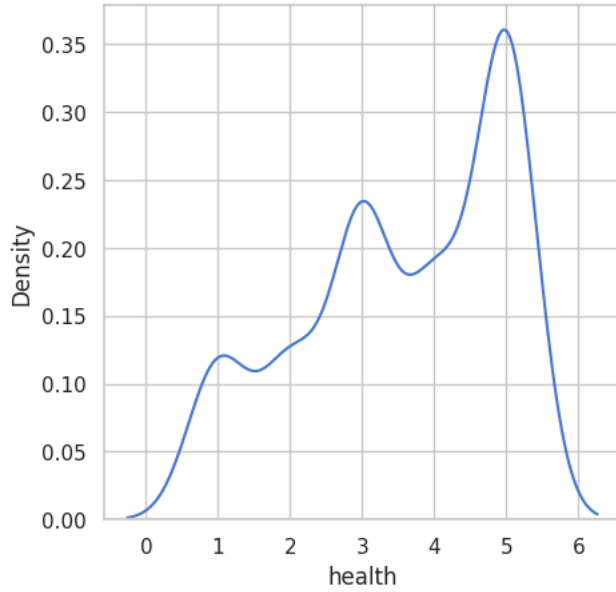
```
freetime
freetime
3    157
4    115
2     64
5     40
1     19
Name: count, dtype: int64
```



```
goout
goout
3    157
4    115
2     64
5     40
1     19
Name: count, dtype: int64
```

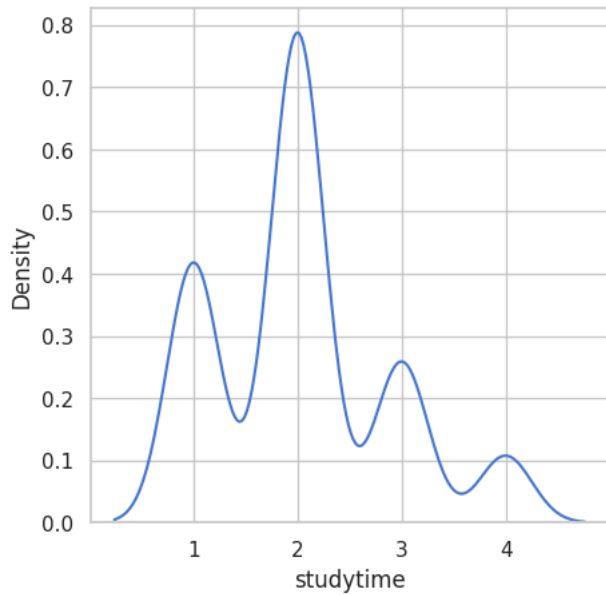
```
3    130
2    103
4     86
5     53
1     23
```

Name: count, dtype: int64



```
health
health
5    146
3     91
4     66
1     47
2     45
```

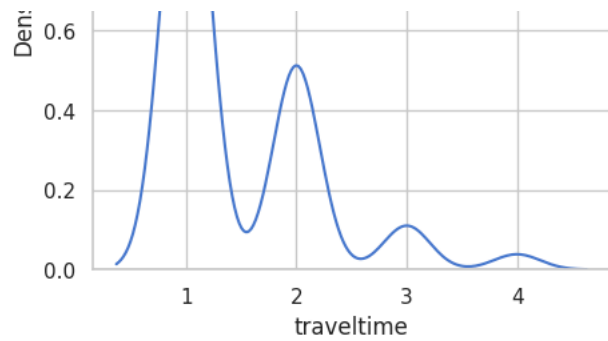
Name: count, dtype: int64



```
studytime
studytime
2    198
1    105
3     65
4     27
```

Name: count, dtype: int64

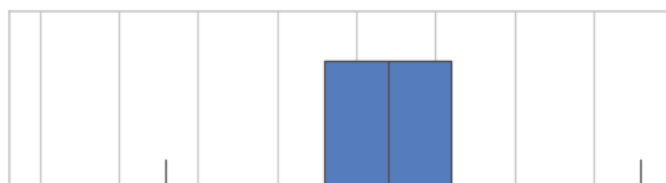
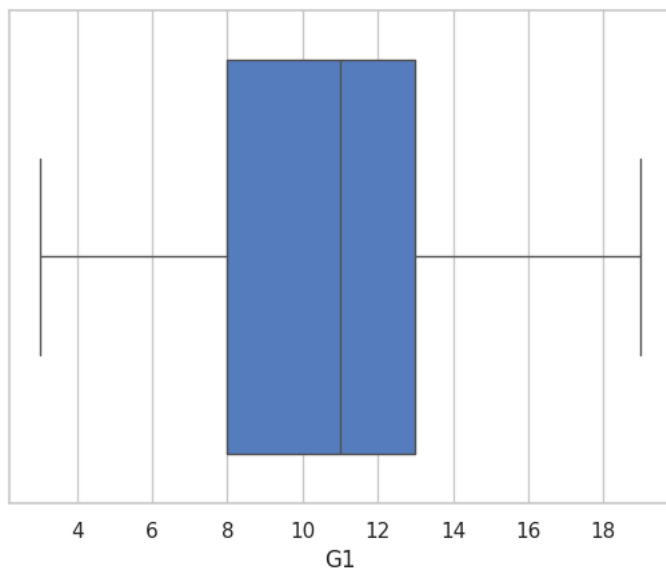
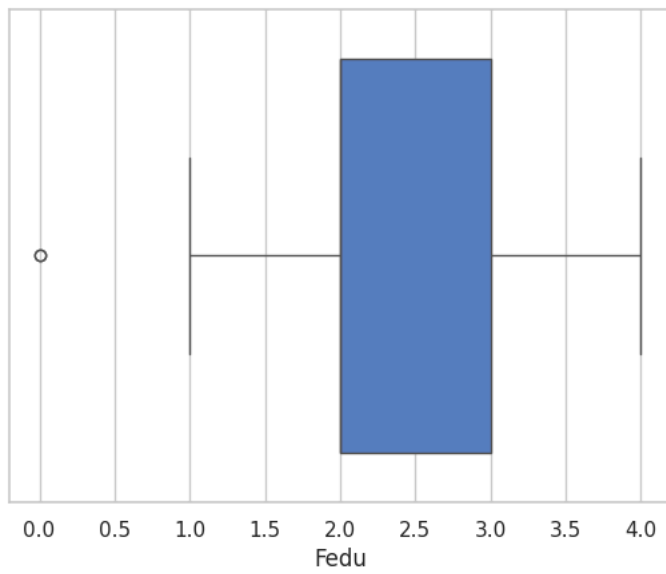
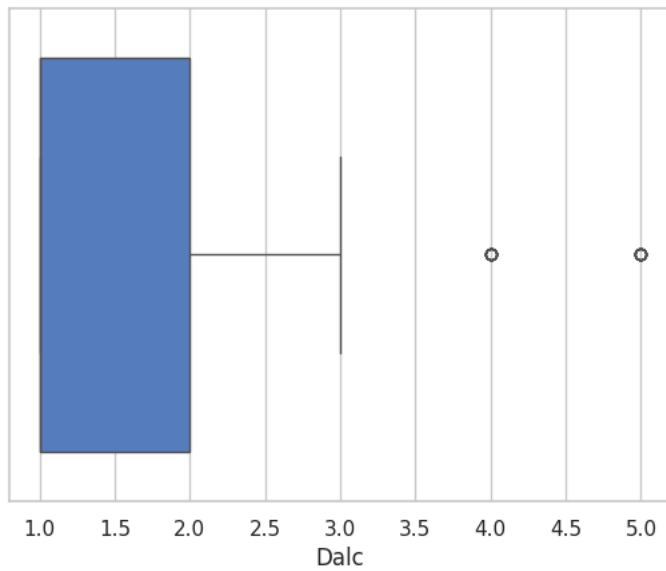


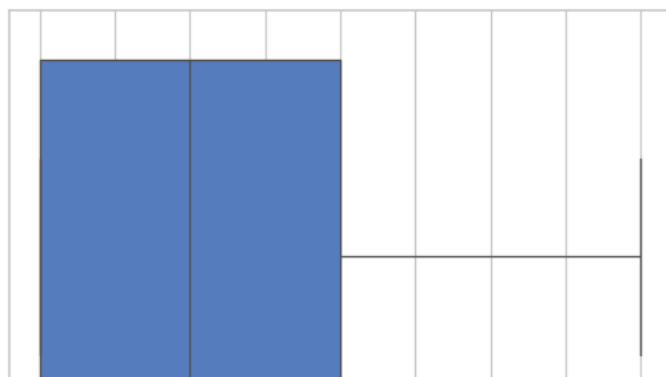
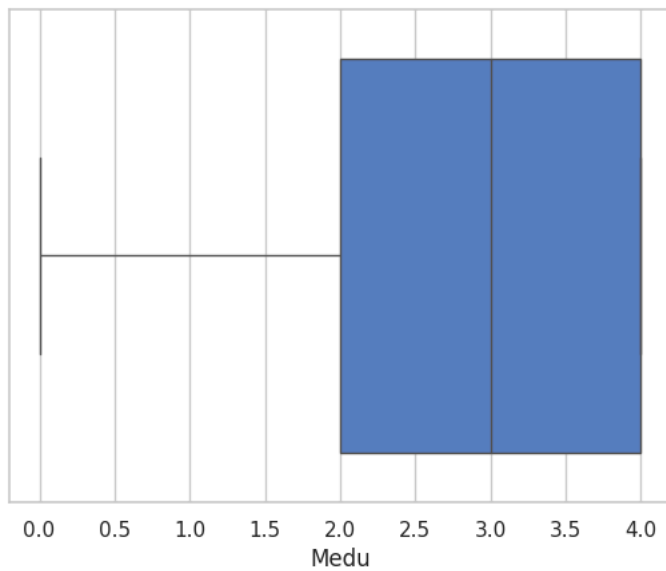
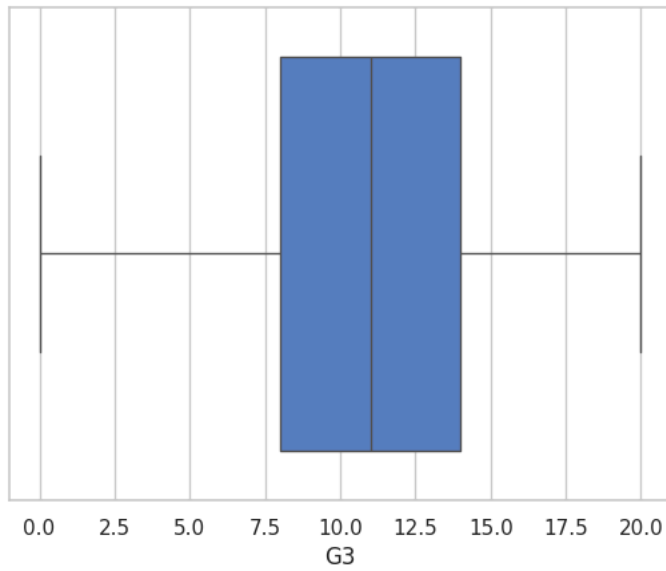
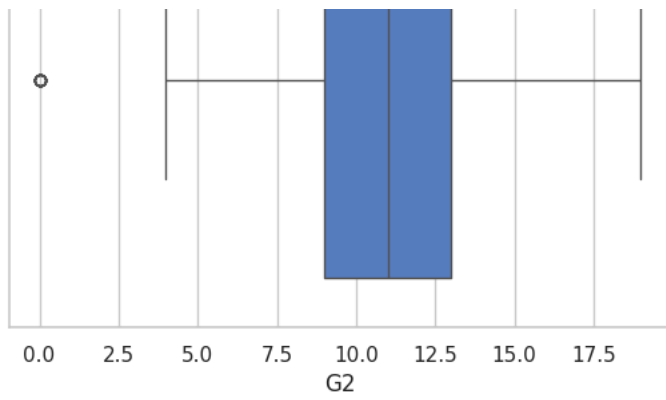


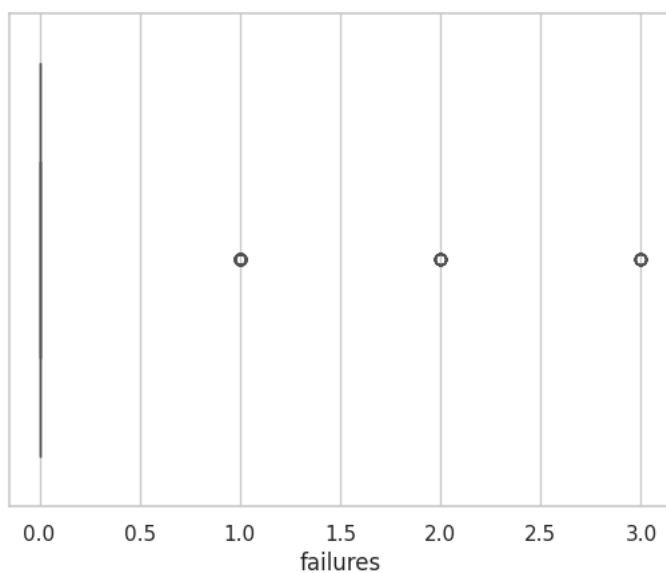
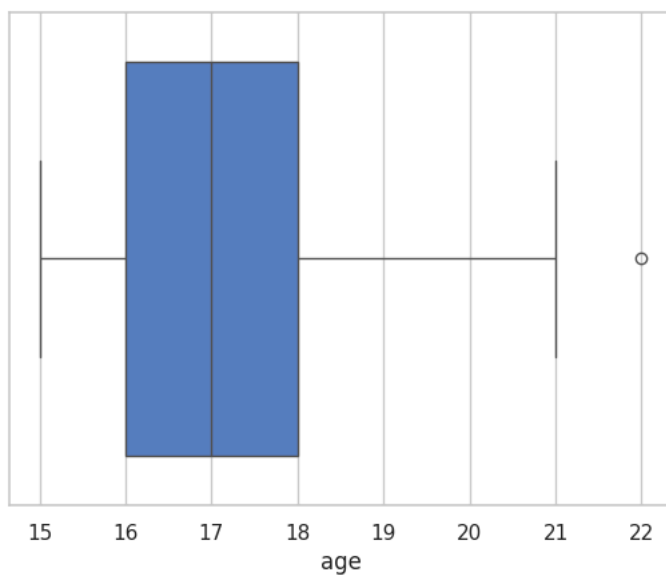
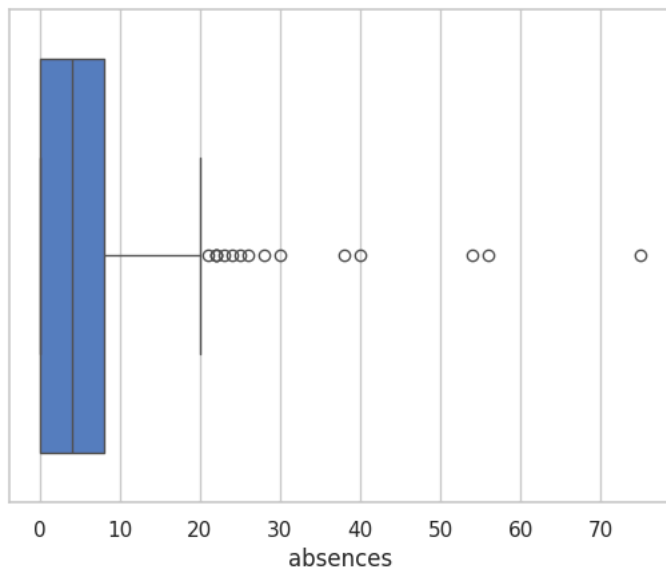
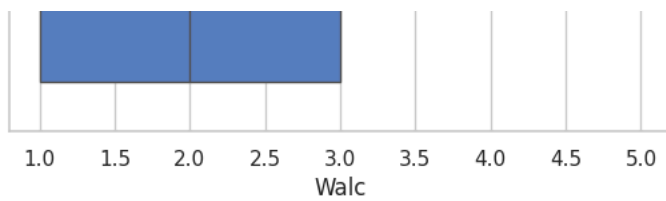
```
traveltime
traveltime
1    257
2    107
3     23
4      8
Name: count, dtype: int64
```

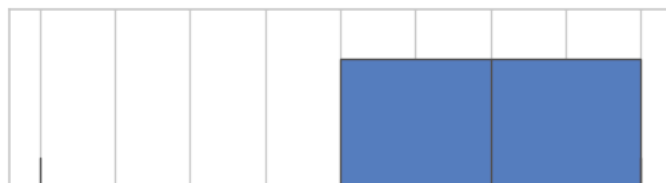
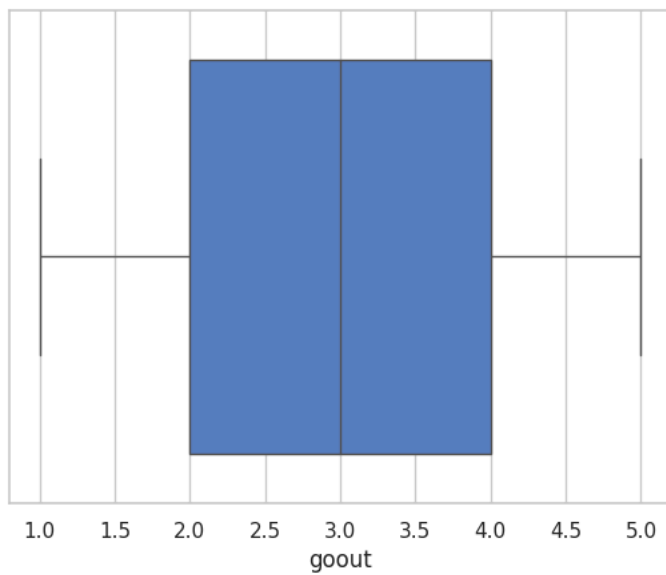
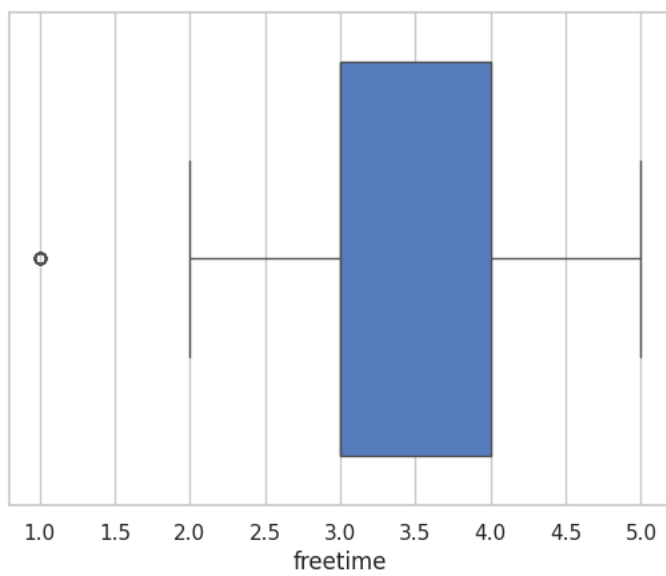
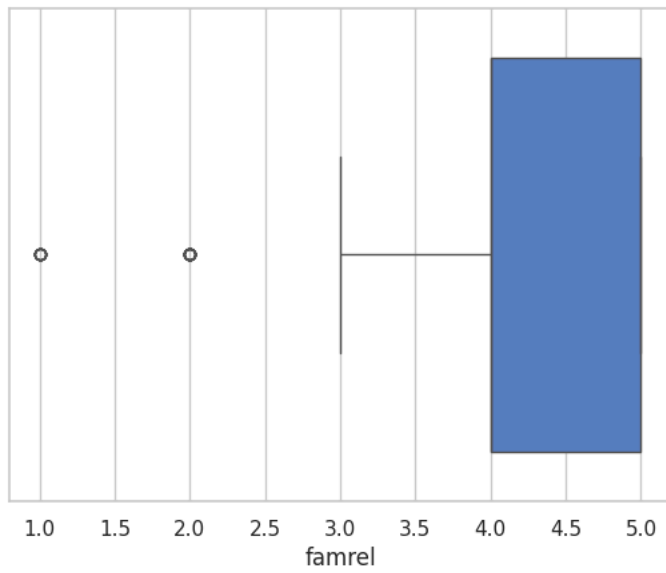
✓ Box Plot For Numerical Data

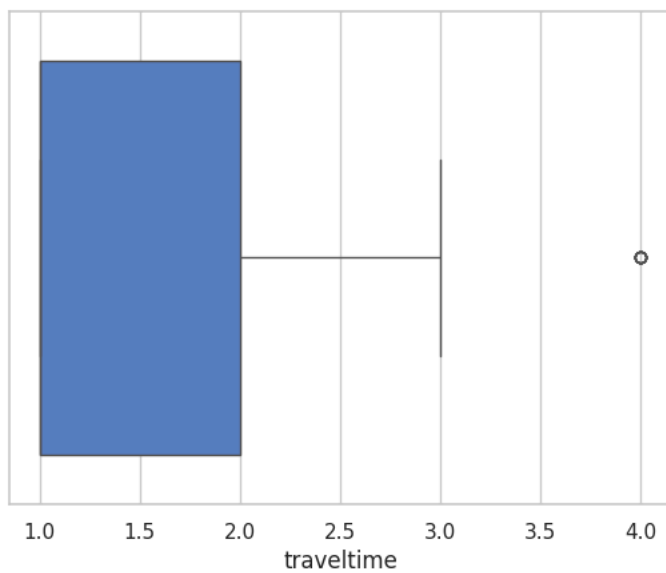
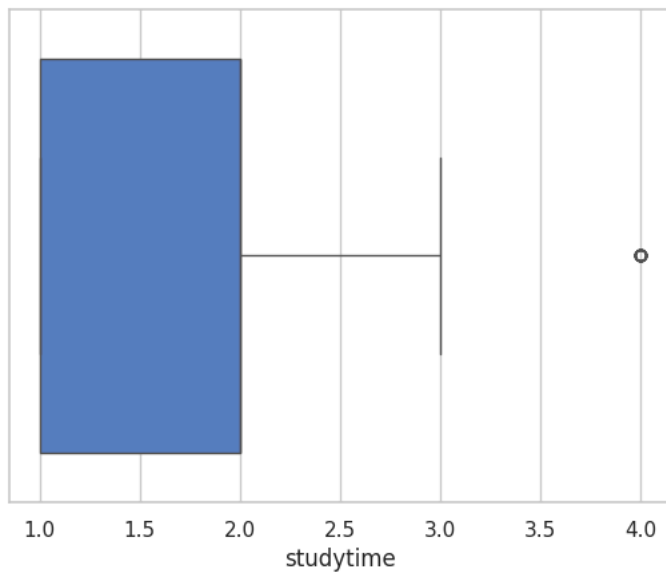
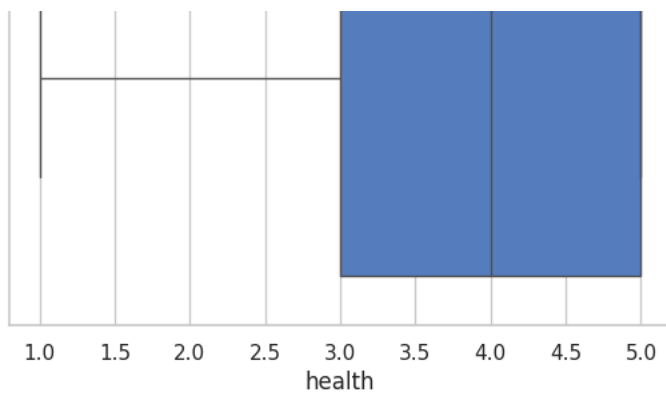
```
for i in cols_numerical:
    ax=sns.boxplot(x=df[i])
    plt.show()
```







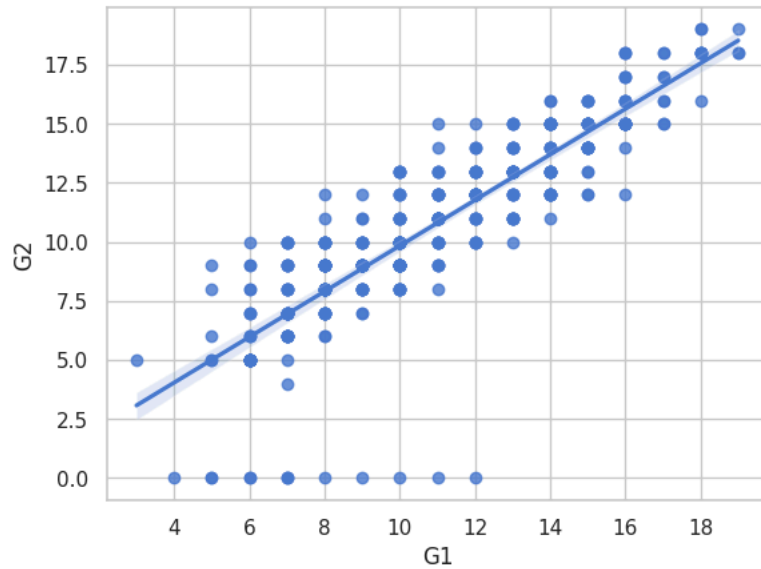




✓ Finding Correlation Between Features

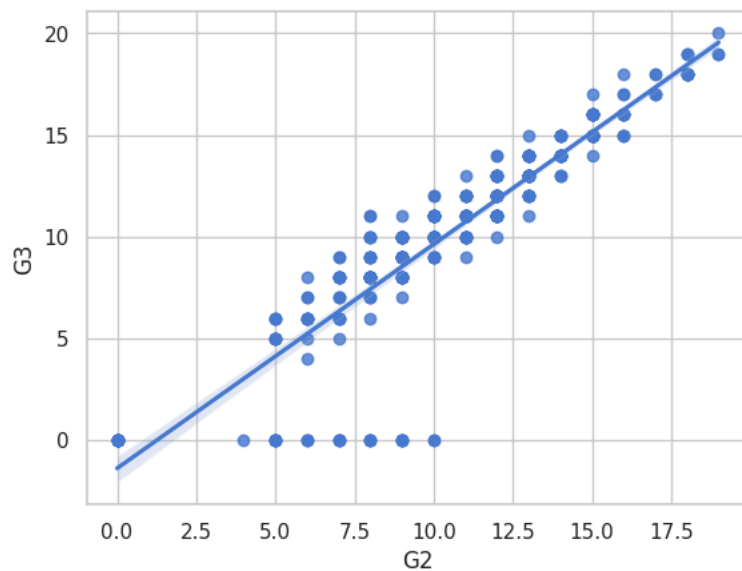
```
sns.regplot(x="G1", y="G2", data=df)
```

<Axes: xlabel='G1', ylabel='G2'>



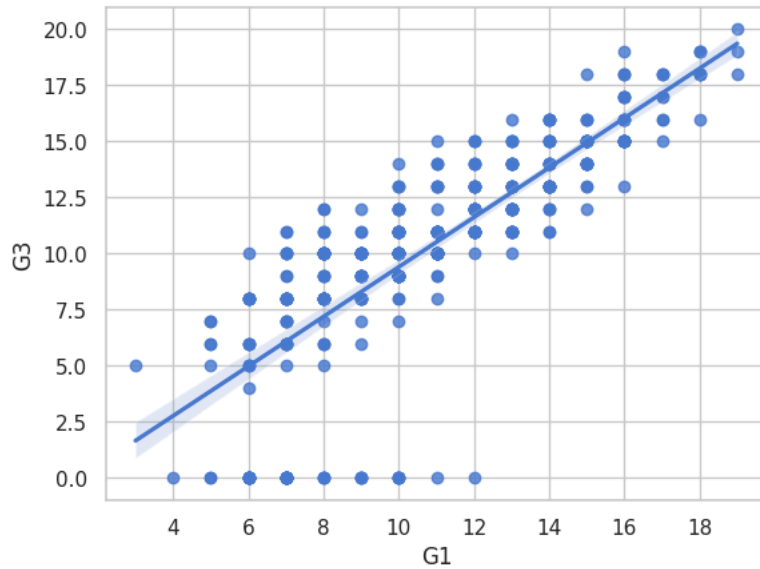
```
sns.regplot(x="G2", y="G3", data=df)
```

<Axes: xlabel='G2', ylabel='G3'>



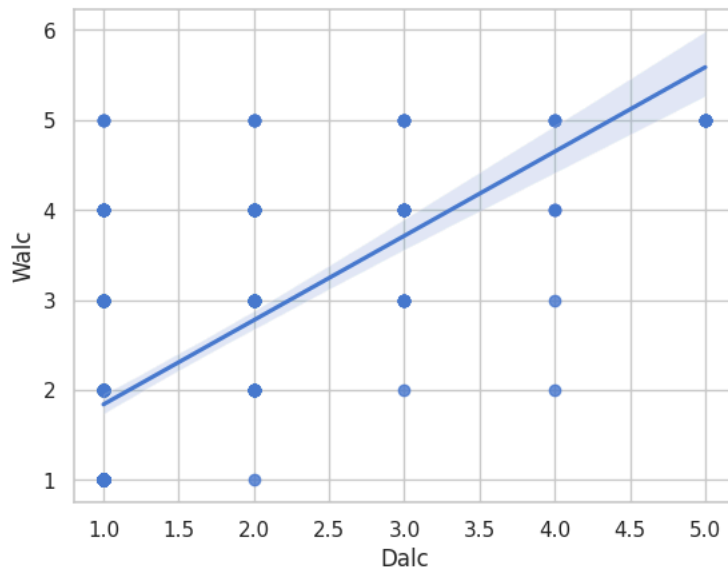
```
sns.regplot(x="G1", y="G3", data=df)
```

<Axes: xlabel='G1', ylabel='G3'>



```
sns.regplot(x="Dalc",y="Walc",data=df)
```

<Axes: xlabel='Dalc', ylabel='Walc'>



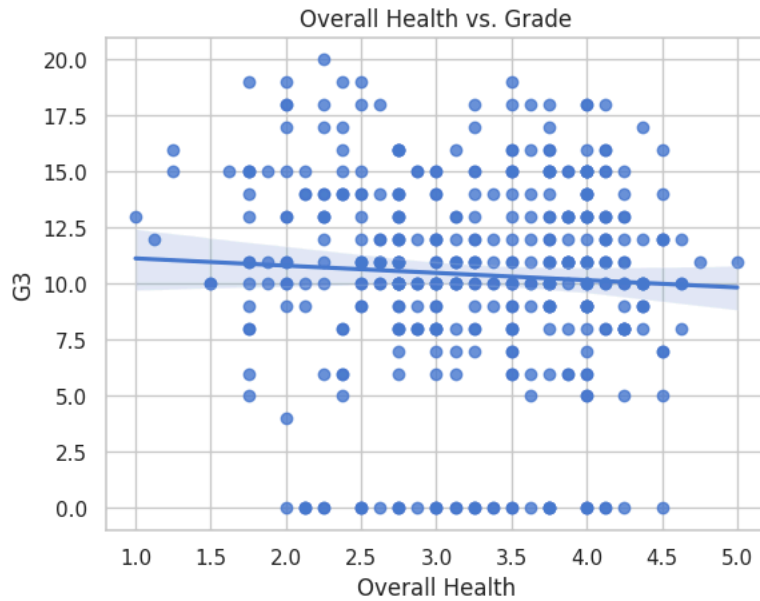
```
df['Overall Health'] = (0.5 * df['Dalc'] + 0.5 * df['Walc'] + 2 * df['health'] + df['famrel']) / 4
df['Overall Health']
```

```
0    2.750
1    3.000
2    3.125
3    3.500
4    3.875
...
390  4.375
391  2.375
392  3.500
393  4.375
394  4.000
Name: Overall Health, Length: 395, dtype: float64
```

Overall Health vs. Grade

```
sns.regplot(x='Overall Health', y='G3', data=df).set(title = 'Overall Health vs. Grade')
```

```
[Text(0.5, 1.0, 'Overall Health vs. Grade')]
```



Observations

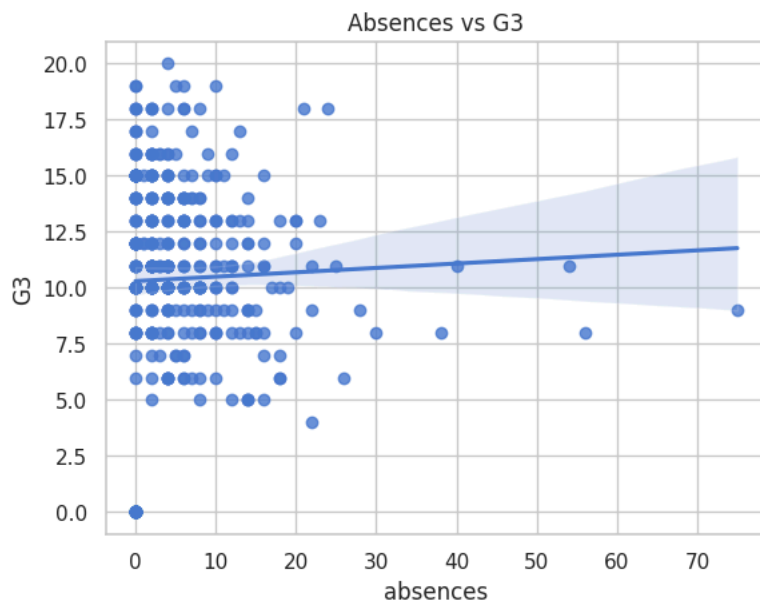
- In general better overall health value corresponds to a lower final grade. Conversely, a lower overall health value generally corresponds to a higher final grade.
- The slope of the best fit line is a small distance away from zero, so the correlation between Overall Health and G3 is low.

We can explore the latter observation further by organising the plots based on the presence or absence of certain attributes

✓ Absences vs. Grade

```
sns.regplot(x='absences', y='G3', data=df).set(title='Absences vs G3')
```

```
[Text(0.5, 1.0, 'Absences vs G3')]
```

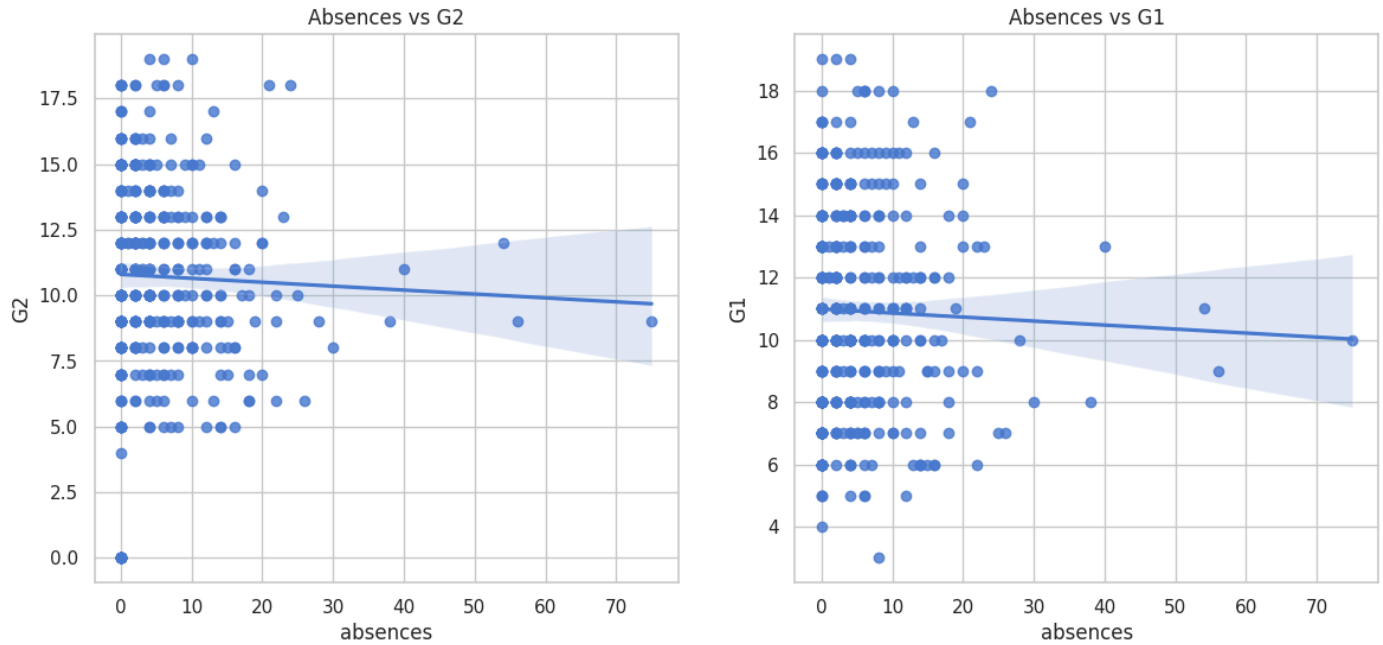


There seems to be no correlation between Absences and G3 and this is interesting we need to identify the potential cause for this phenomenon, and we need to examine more relationships on Absences to determine the cause.

```
fig, axes = plt.subplots(1,2, figsize=(14,6))
sns.regplot(x='absences', y='G2', data=df, ax=axes[0])
axes[0].set(title='Absences vs G2')

sns.regplot(x='absences', y='G1', data=df, ax=axes[1])
axes[1].set(title='Absences vs G1')
```

```
[Text(0.5, 1.0, 'Absences vs G1')]
```

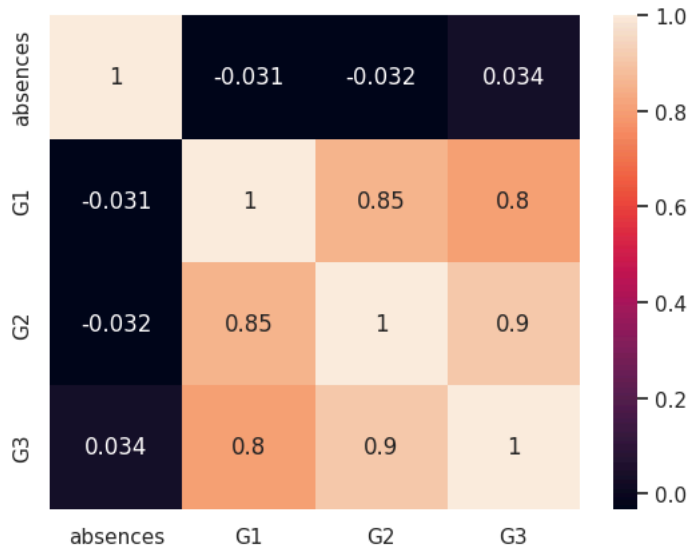


This low correlation is because absent students (usually) revise the material missed, effectively accounting for their absence.

For a further explanation, we need to plot the correlation matrix to validate our hypothesis.

```
sns.heatmap(df[['absences', 'G1', 'G2', 'G3']].corr(), annot=True)
```

<Axes: >

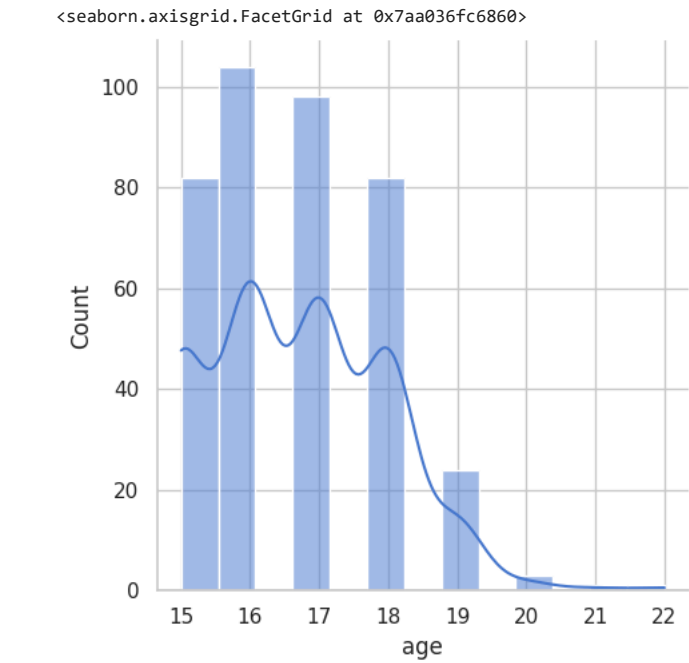


Conclusion

- The number of absences a student has does not necessarily result in a lower grade overall, as absent students typically make up the material they missed, effectively accounting for their absence.

✓ Age vs. Grade

```
sns.displot(x='age', data=df, kind='hist', kde=True)
```



```
age_grade = df.groupby("age").aggregate({'G1': 'mean', 'G2': 'mean', 'G3': 'mean'})
age_grade.reset_index(inplace=True)
age_grade
```

	age	G1	G2	G3	
0	15	11.231707	11.365854	11.256098	
1	16	10.942308	11.182692	11.028846	
2	17	10.897959	10.479592	10.275510	
3	18	10.719512	10.134146	9.548780	
4	19	10.250000	9.250000	8.208333	
5	20	13.666667	13.666667	14.000000	
6	21	10.000000	8.000000	7.000000	
7	22	6.000000	8.000000	8.000000	

Next steps: [View recommended plots](#)

There's an inverse relationship between age and Grads

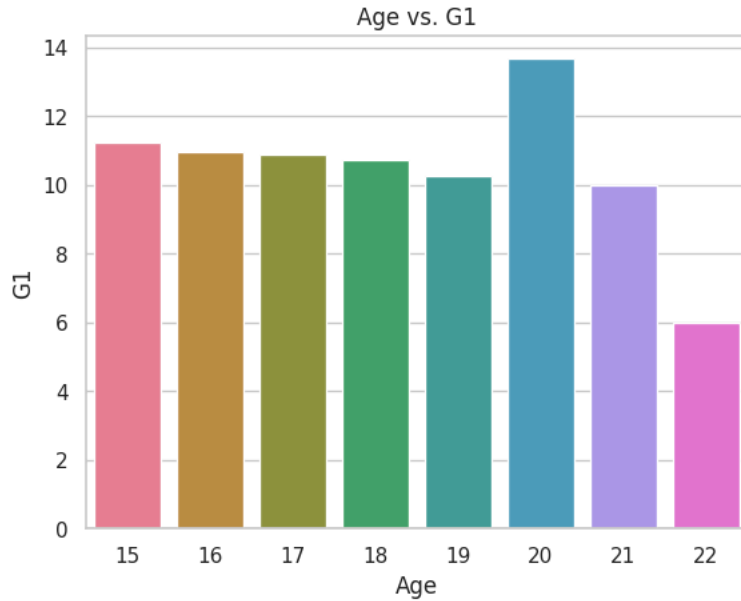
```
grades = ['G1', 'G2', 'G3']

for grade in grades:
    sns.barplot(data=age_grade, x='age', y=grade, palette='husl').set(xlabel='Age', ylabel=grade, title=f'Age vs. {grade}');
    plt.show();
```

```
<ipython-input-39-f868529448ca>:4: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

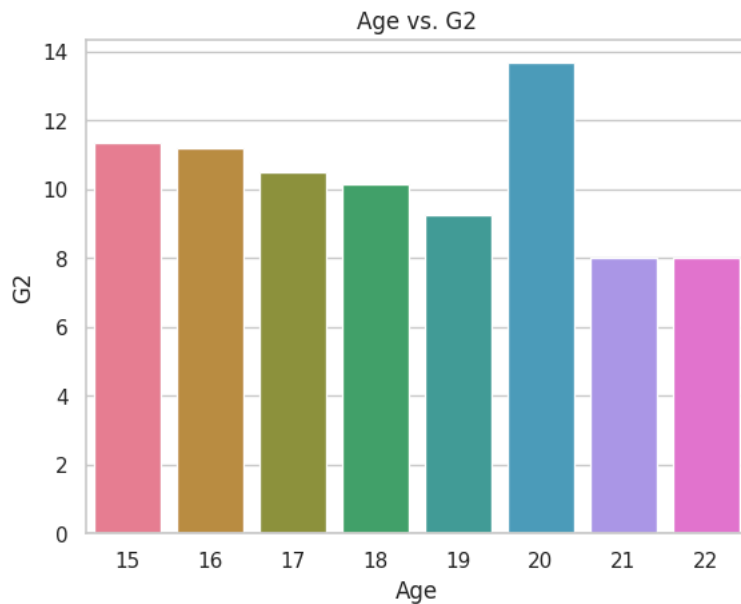
```
sns.barplot(data=age_grade, x='age', y=grade, palette='husl').set(xlabel='Age', ylabel=grade, title=f'Age vs. {grade}');
```



```
<ipython-input-39-f868529448ca>:4: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

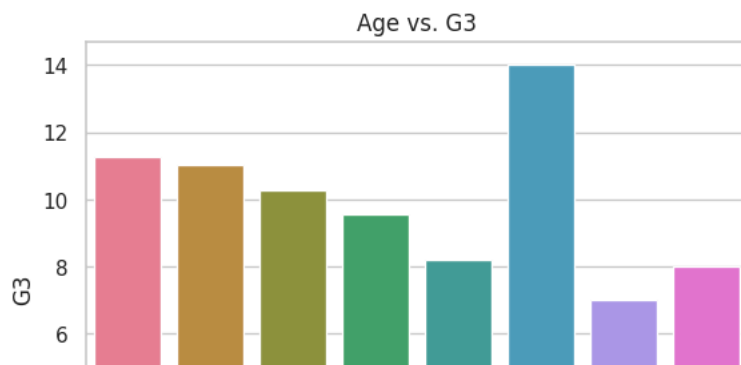
```
sns.barplot(data=age_grade, x='age', y=grade, palette='husl').set(xlabel='Age', ylabel=grade, title=f'Age vs. {grade}');
```

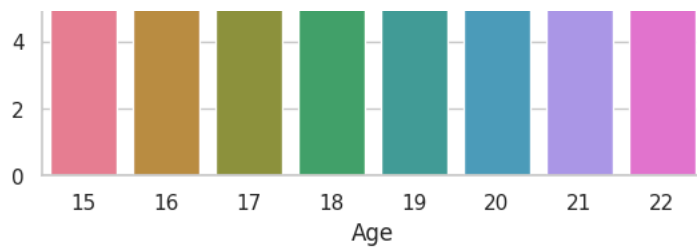


```
<ipython-input-39-f868529448ca>:4: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(data=age_grade, x='age', y=grade, palette='husl').set(xlabel='Age', ylabel=grade, title=f'Age vs. {grade}');
```





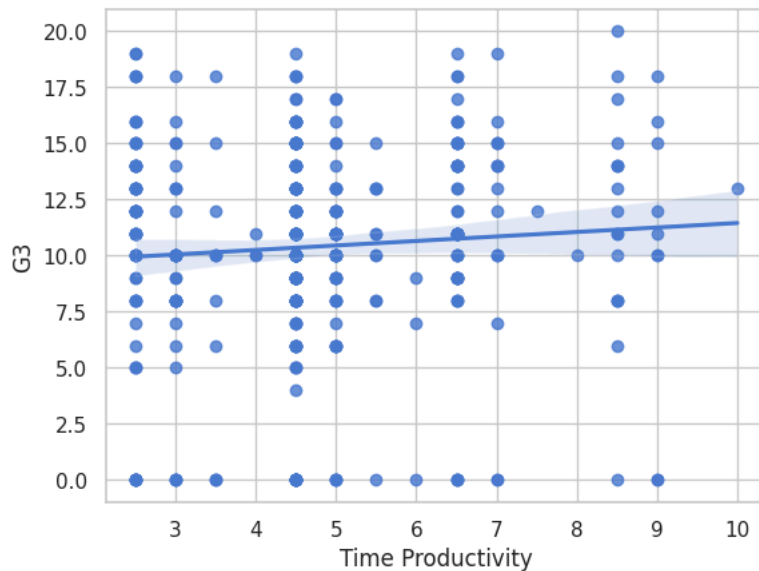
Conclusion

- The older a person is, the lower the grades they receive are; however, twenty-year-olds (in this dataset) exhibit an outstanding performance: they lead by at least 2 whole grade points on a 19-point scale.
- At the moment, we cannot make any substantiated claims as to why such a drastic increase occurs.

✓ Time Productivity vs. Grade

```
df['Time Productivity'] = 0.5 * df['traveltime'] + 2 * df['studytime']
sns.regplot(x='Time Productivity', y='G3', data=df)
```

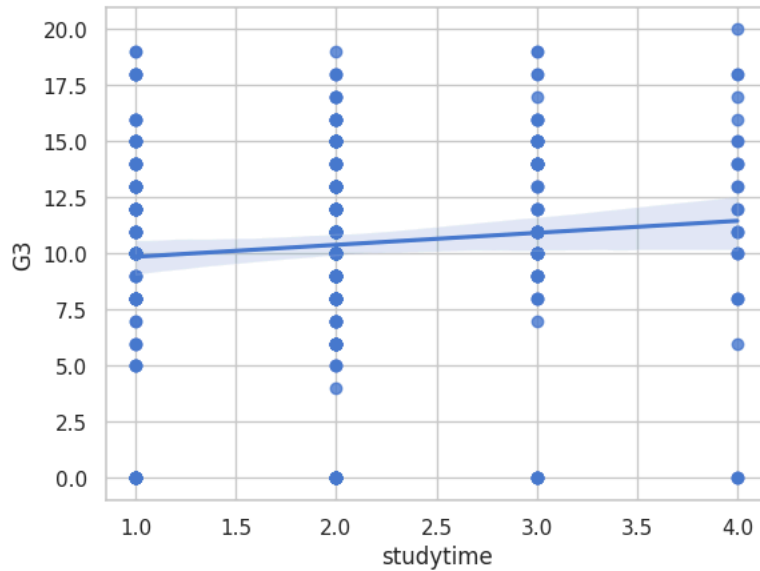
<Axes: xlabel='Time Productivity', ylabel='G3'>



As expected, students with a greater time productivity have better grades.

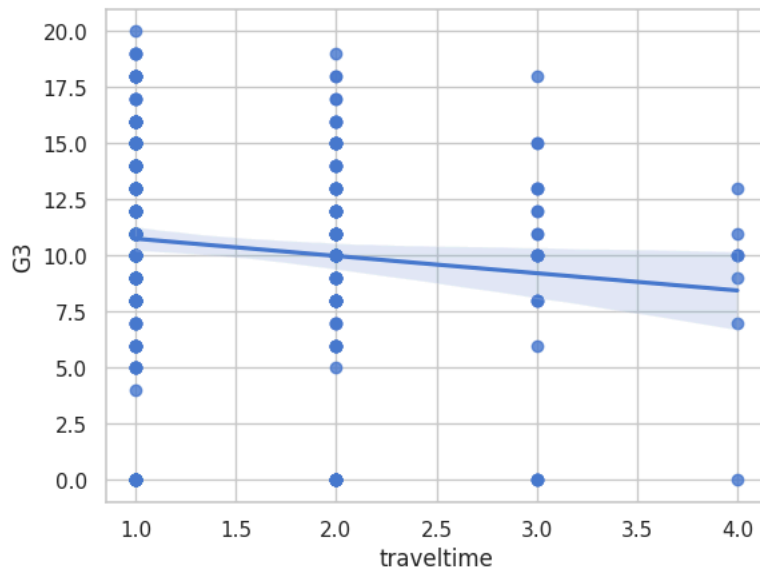
```
sns.regplot(x='studytime', y='G3', data=df)
```


<Axes: xlabel='studytime', ylabel='G3'>



```
sns.regplot(x='traveltime', y='G3', data=df)
```

<Axes: xlabel='traveltime', ylabel='G3'>



Another batch of expected results. Students who study more score better on tests and quizzes. In contrast, students who travel more perform worse on tests and quizzes.

Conclusion

- This section confirmed the obvious: students who study more receive better grades whereas students who travel more or study less receive lower grades.

✓ Mothers' Job and Education

```
MotherJop_Edu = df.groupby("Mjob").aggregate({"Medu": "mean"})
MotherJop_Edu.reset_index(inplace=True)
MotherJop_Edu.sort_values(by='Medu', ascending=False, inplace=True)
```

MotherJop_Edu

	Mjob	Medu	
4	teacher	3.948276	
1	health	3.647059	
3	services	2.844660	
2	other	2.404255	
0	at_home	1.711864	

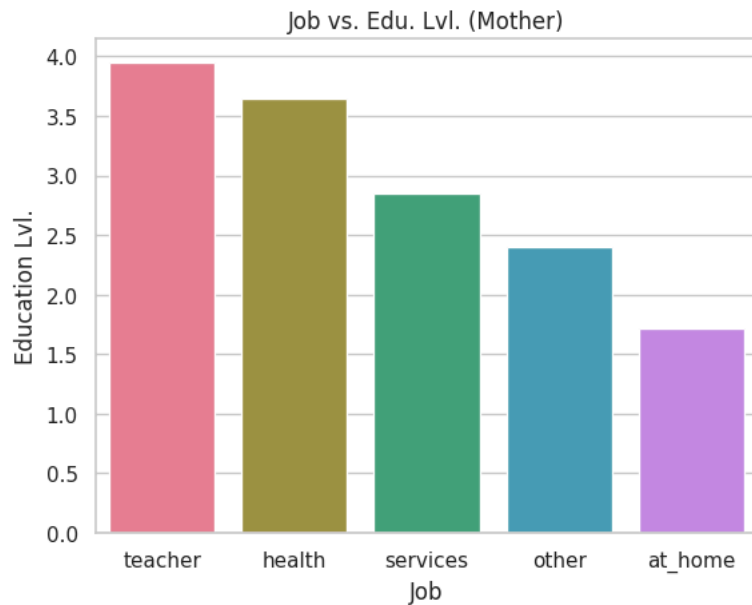
Next steps: [View recommended plots](#)

```
sns.barplot(x='Mjob', y='Medu', data=MotherJop_Edu , palette='husl').set(xlabel='Job', ylabel='Education Lvl.', title='Job vs. Edu. Lvl. (Mot
```

<ipython-input-45-056ce13d9582>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(x='Mjob', y='Medu', data=MotherJop_Edu , palette='husl').set(xlabel='Job', ylabel='Education Lvl.', title='Job vs. Edu. Lv
```



- teachers and health care professionals need to have a high education level in order to acquire a job in the industry, and conversely with at home mother.

✓ Fathers' Job and Education

```
FatherJop_Edu = df.groupby("Fjob").aggregate({"Fedu": "mean"})
FatherJop_Edu.reset_index(inplace=True)
FatherJop_Edu.sort_values(by='Fedu', ascending=False, inplace=True)
```

FatherJop_Edu

	Fjob	Fedu	
4	teacher	3.862069	
1	health	3.333333	
3	services	2.558559	
0	at_home	2.350000	
2	other	2.271889	

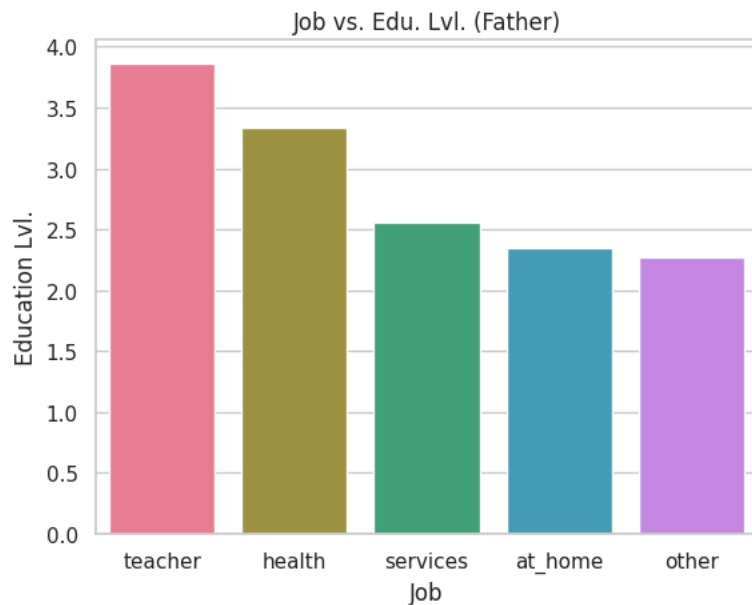
Next steps: [View recommended plots](#)

```
sns.barplot(x='Fjob', y='Fedu', data=FatherJop_Edu ,palette='husl').set(xlabel='Job', ylabel='Education Lvl.', title='Job vs. Edu. Lvl. (Fat
```

<ipython-input-48-e6b26d375e6c>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(x='Fjob', y='Fedu', data=FatherJop_Edu ,palette='husl').set(xlabel='Job', ylabel='Education Lvl.', title='Job vs. Edu. Lvl
```



The graph for the father yields about the same results as the graph for the mother. However, there is one interesting finding: at-home fathers have a higher education level than at-home mothers.

Conclusion

- This mini-exploration yielded reasonable results. Teachers and health care professionals had a higher education level while at-home parents tended to reside at the lower end of the education level spectrum.

✓ Job vs. Grade

✓ 1. Mothers' Job

```
MotherJop_Grade = df.groupby('Mjob').aggregate({'G3': 'mean'}).reset_index()
MotherJop_Grade
```

	Mjob	G3
0	at_home	9.152542
1	health	12.147059
2	other	9.822695
3	services	11.019417
4	teacher	11.051724

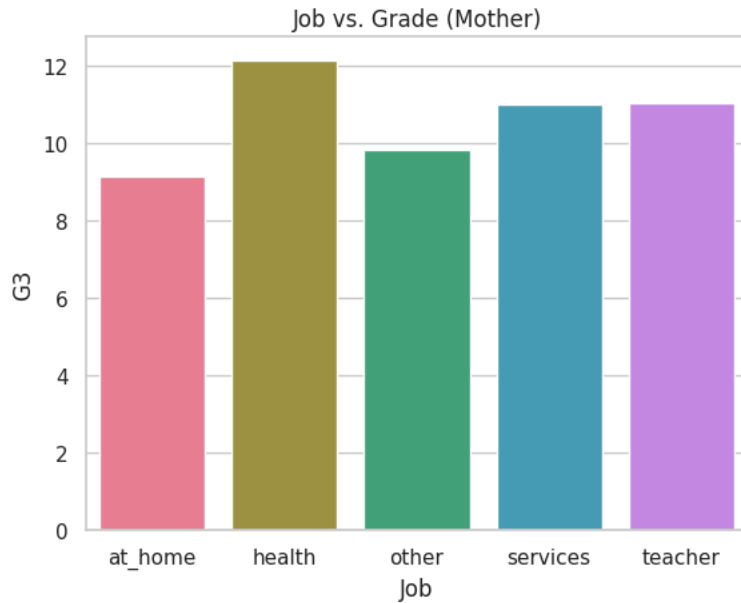
Next steps: [View recommended plots](#)

```
sns.barplot(x = 'Mjob' , y='G3' , palette='husl' , data = MotherJop_Grade).set(xlabel='Job', ylabel='G3', title='Job vs. Grade (Mother)');
```

```
<ipython-input-50-893a923c80c7>:1: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(x = 'Mjob' , y='G3' , palette='husl' , data = MotherJop_Grade).set(xlabel='Job', ylabel='G3', title='Job vs. Grade (Mother
```



Surprisingly, students with mothers who are health care professionals receive a higher grade on average. I expected students with mothers who were teachers to score the highest

2. Fathers' Job

```
FatherJop_Grade = df.groupby('Fjob').aggregate({'G3': 'mean'}).reset_index()
FatherJop_Grade
```

	Fjob	G3
0	at_home	10.150000
1	health	11.611111
2	other	10.193548
3	services	10.297297
4	teacher	11.965517

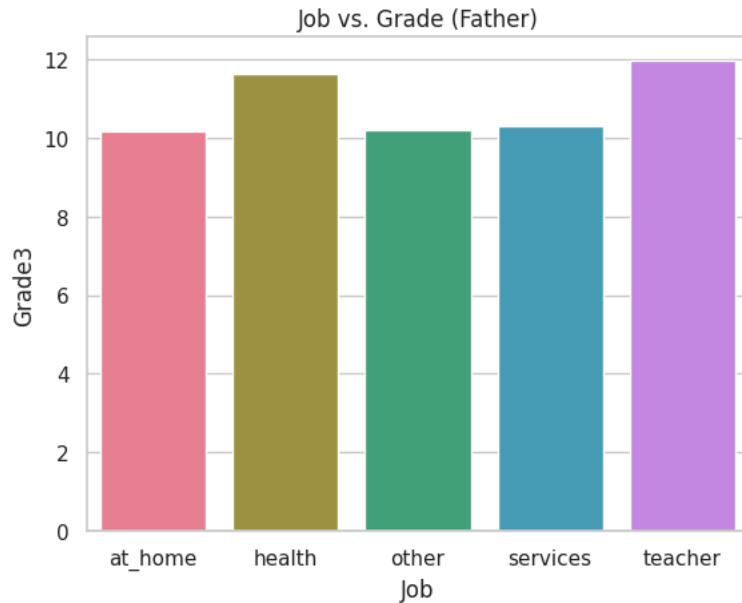
Next steps: [View recommended plots](#)

```
sns.barplot(x='Fjob', y='G3', data=FatherJop_Grade , palette='husl').set(xlabel='Job', ylabel='Grade3', title='Job vs. Grade (Father)');
```

```
<ipython-input-52-6d6a64970956>:1: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(x='Fjob', y='G3', data=FatherJop_Grade , palette='husl').set(xlabel='Job', ylabel='Grade3', title='Job vs. Grade (Father)')
```



These results are more reasonable than the previous set. Notice that the order of these results corresponds with the order of the father's education. In other words, fathers with higher education levels tend to increase their children's performance more than fathers with lower education levels. The father directly passes down his knowledge, thus cultivating the student's knowledge.

Conclusion

- This exploration uncovered the slightly surprising relationship between the Mother's Job and the student's grade. We discovered that mothers who work in the health industry have children who receive higher scores on average than mothers who work in other fields.
- In contrast, for fathers, we observed a relationship one might expect. Fathers with higher educational levels directly passed on that knowledge to the student, eventually improving the student's overall test scores.

Family Size vs. Grade

```
Family_size_grade = df.groupby("famsize").aggregate({"G1": "mean", "G2": "mean", "G3": "mean"}).reset_index()
Family_size_grade
```

	famsize	G1	G2	G3
0	GT3	10.758007	10.519573	10.177936
1	LE3	11.280702	11.192982	11.000000

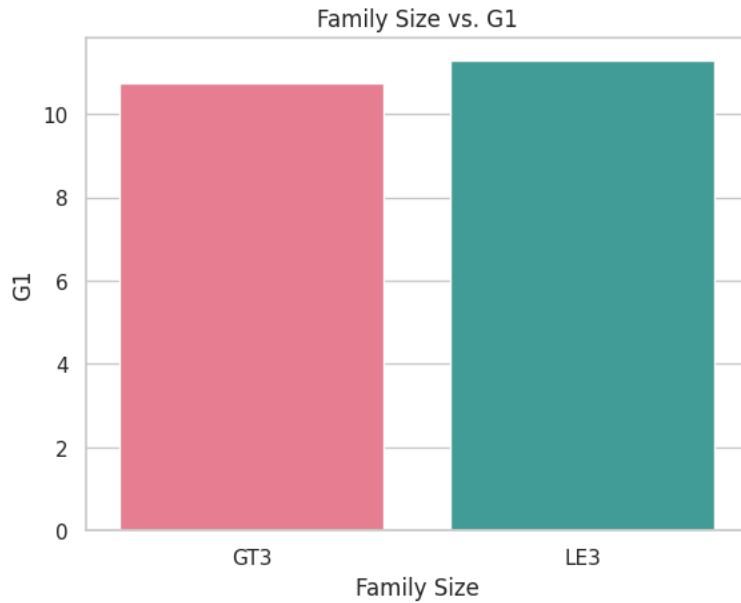
Next steps: [View recommended plots](#)

```
for grade in grades:
    sns.barplot(data=Family_size_grade, x='famsize', y=grade, palette='husl').set(xlabel='Family Size', ylabel=grade, title=f'Family Size vs. {grade}')
    plt.show();
```

```
<ipython-input-54-914a18135fcf>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

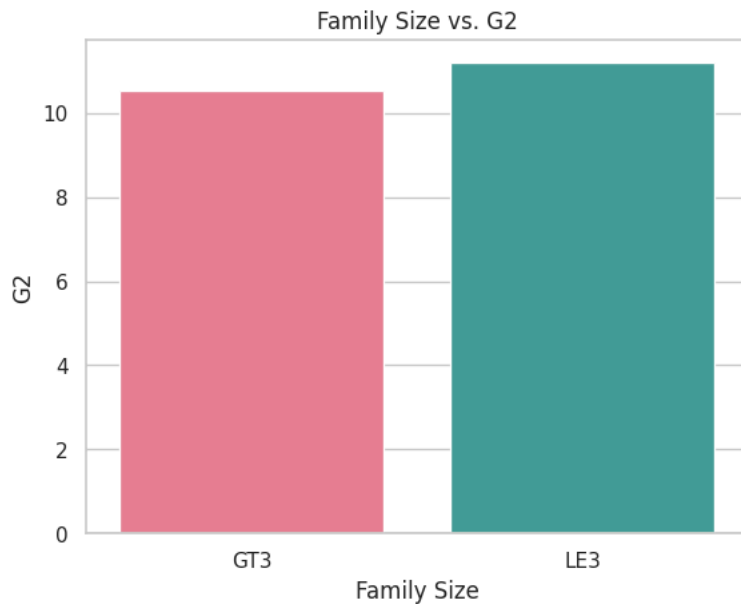
```
sns.barplot(data=Family_size_grade, x='famsize', y=grade, palette='husl').set(xlabel='Family Size', ylabel=grade, title=f'Family Size
```



```
<ipython-input-54-914a18135fcf>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

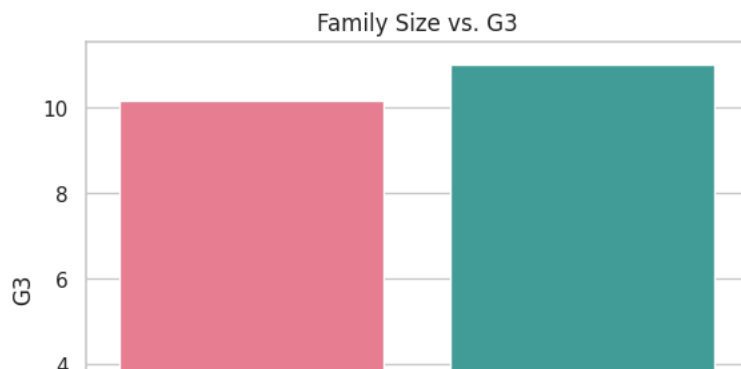
```
sns.barplot(data=Family_size_grade, x='famsize', y=grade, palette='husl').set(xlabel='Family Size', ylabel=grade, title=f'Family Size
```

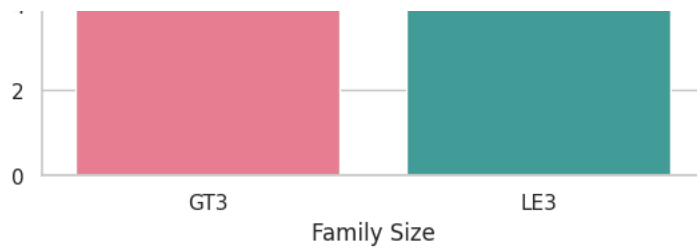


```
<ipython-input-54-914a18135fcf>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(data=Family_size_grade, x='famsize', y=grade, palette='husl').set(xlabel='Family Size', ylabel=grade, title=f'Family Size
```





Conclusion

- As the barplots show, children with no siblings tend to score slightly - indeed, very slightly - higher than students with siblings

Activites vs. Grade

This section explores the relationship between the amount of social interaction a person undergoes and how said person scores on an exam.

```
act_gr = df.groupby("activities").aggregate({"G1": "mean", "G2": "mean", "G3": "mean"}).reset_index()
act_gr
```

	activities	G1	G2	G3	
0	no	10.716495	10.520619	10.340206	
1	yes	11.094527	10.900498	10.487562	

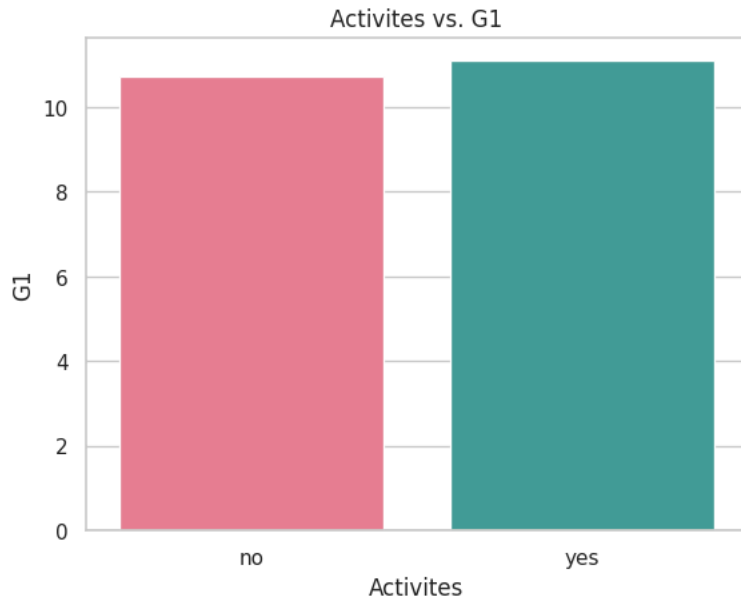
Next steps: [View recommended plots](#)

```
for grade in grades:
    sns.barplot(data=act_gr, x='activities', y=grade, palette='husl').set(xlabel='Activites', ylabel=grade, title=f'Activites vs. {grade}');
    plt.show();
```

```
<ipython-input-56-56512f8967ce>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

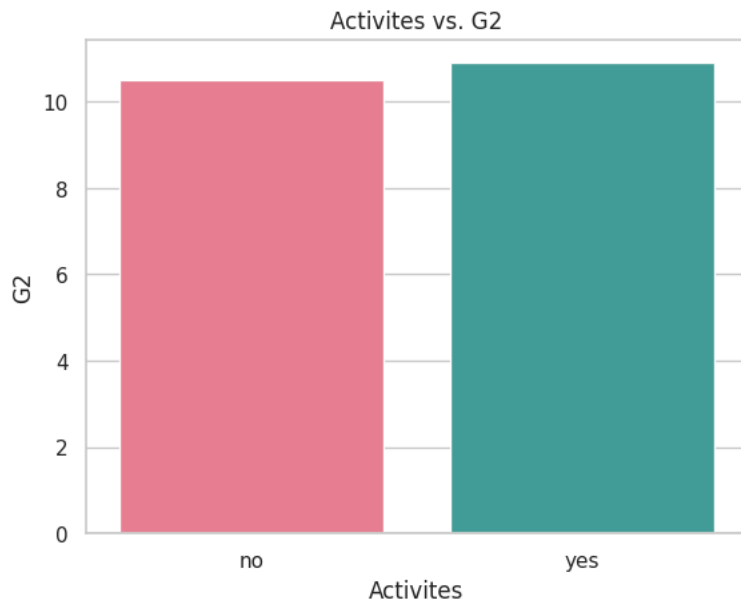
```
sns.barplot(data=act_gr, x='activities', y=grade, palette='husl').set(xlabel='Activites', ylabel=grade, title=f'Activites vs. {grade}')
```



```
<ipython-input-56-56512f8967ce>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

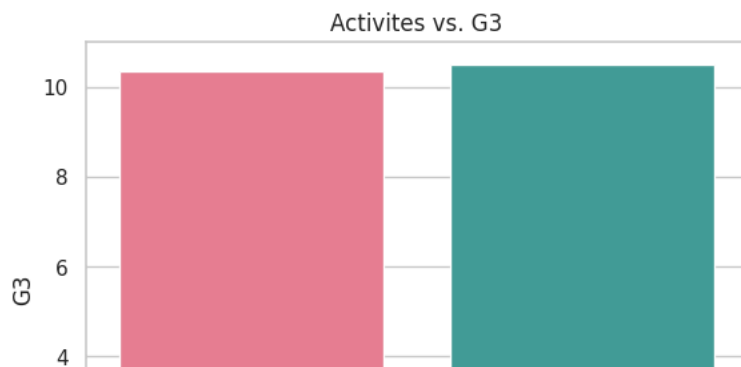
```
sns.barplot(data=act_gr, x='activities', y=grade, palette='husl').set(xlabel='Activites', ylabel=grade, title=f'Activites vs. {grade}')
```



```
<ipython-input-56-56512f8967ce>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(data=act_gr, x='activities', y=grade, palette='husl').set(xlabel='Activites', ylabel=grade, title=f'Activites vs. {grade}')
```





People with activites receive marginally higher grades on average.

Go out vs Grades

```
out_gr = df.groupby("goout").aggregate({"G1": "mean", "G2": "mean", "G3": "mean"}).reset_index()
out_gr
```

	goout	G1	G2	G3
0	1	11.130435	10.782609	9.869565
1	2	11.368932	11.456311	11.194175
2	3	11.276923	11.053846	10.961538
3	4	10.430233	10.058140	9.651163
4	5	9.792453	9.471698	9.037736

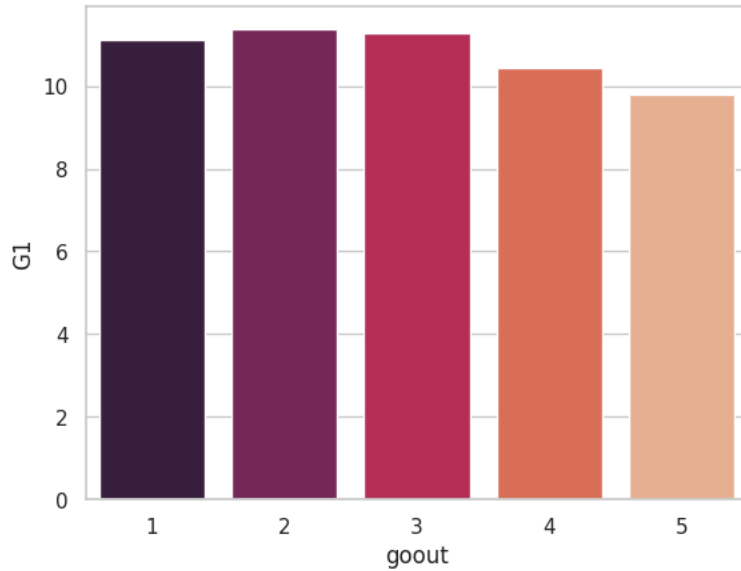
Next steps: [View recommended plots](#)

```
for grade in grades:
    sns.barplot(data=out_gr, x='goout', y=grade, palette='rocket')
plt.show()
```

```
<ipython-input-58-b5aa35db3777>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

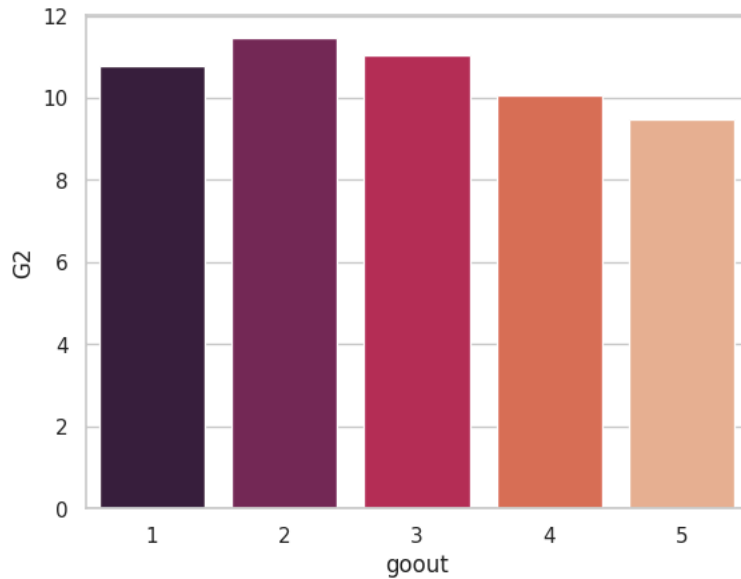
```
sns.barplot(data=out_gr, x='goout', y=grade, palette='rocket')
```



```
<ipython-input-58-b5aa35db3777>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

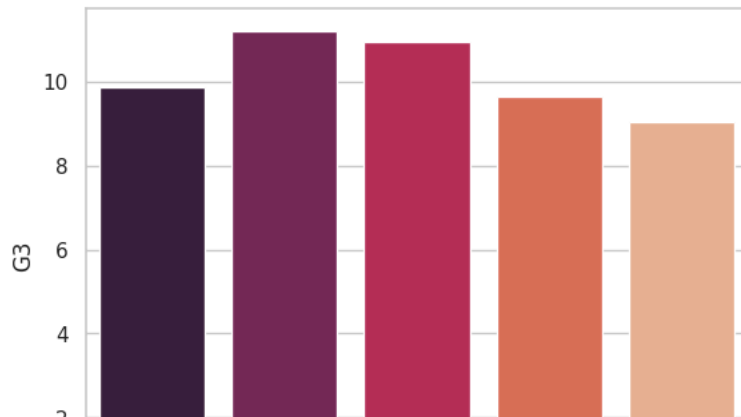
```
sns.barplot(data=out_gr, x='goout', y=grade, palette='rocket')
```

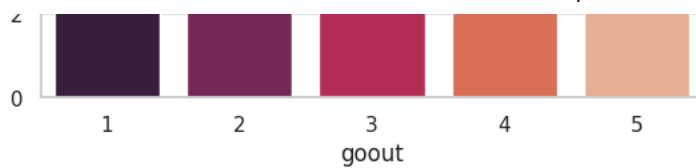


```
<ipython-input-58-b5aa35db3777>:2: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.barplot(data=out_gr, x='goout', y=grade, palette='rocket')
```





Overall, going out more results in an short-lived increase followed by a steep decrease in test scores.

Conclusion

- This section portrayed a surprising relationship between how often a student engages in social activity. Students with social activity, in general, scored better than students without socialization.
- On the contrary, students who went out more often tended to score on the lower side of the grade spectrum; the perfect rating in goout was 2, as it resulted in the highest average score for all scoring metrics.

✓ Data Pre-Processing

Introduction This involves a number of activities such as:

- Assigning numerical values to categorical data;
- Handling missing values;
- Normalizing the features (so that features on small scales do not dominate when fitting a model to the data).

Goal:

- Find the most predictive features of the data and filter it so it will enhance the predictive power of the analytics model.

```
# I already have two lists that have categorical and numerical data but this is other way more easy than last one to get this lists
#list of columns that are categorical
categorical_col = df.select_dtypes(include=['object']).columns.tolist()
#list of columns that are numerical
numerical_col = df.select_dtypes(include=['number']).columns.tolist()
categorical_col
```

```
['school',
 'sex',
 'address',
 'famsize',
 'Pstatus',
 'Mjob',
 'Fjob',
 'reason',
 'guardian',
 'schoolsup',
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic']
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, MinMaxScaler
```

Label encoding

Here, I assign the 32 features to a NumPy array X and encoded the original string representation on the categorical columns into integers to start the machine learning phase.

```
df.columns
```

```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
      'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
      'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
      'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
      'Walc', 'health', 'absences', 'G1', 'G2', 'G3', 'Overall Health',
      'Time Productivity'],
      dtype='object')
```

```
## dropping reason column as it is not corelated with student score
df2 = df.drop('reason', axis =1)
```

```
new_df = pd.get_dummies(df2, columns=['school',
  'sex',
  'address',
  'famsize',
  'Pstatus',
  'Mjob',
  'Fjob',
  'guardian',
  'schoolsup',
  'famsup',
  'paid',
  'activities',
  'nursery',
  'higher',
  'internet',
  'romantic'])
```

```
X = new_df.drop('G3', axis=1).values
y = new_df['G3'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((316, 56), (316,), (79, 56), (79,))
```

```
# Scaling the data using pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
pipeline = Pipeline([
    ('std_scalar', StandardScaler())
])
```

```
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)
```

Modling

Implementation of Functions for Comparing Metrics for Regression Task

```

from sklearn import metrics
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

import numpy as np

# training with cross validation
def cross_val(model, features, targets):
    scoring = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']

    # Perform cross-validation using cross_validate
    results = cross_validate(model, features, targets, cv=5, scoring=scoring)

    # Calculate the mean scores for each metric separately for each target
    mean_scores = {}
    for metric in scoring:
        mean_scores[metric] = -results[f'test_{metric}'].mean()

    return mean_scores

```

Linear Regression

```

# linear regression

from sklearn.linear_model import LinearRegression

# Creating a Linear Regression model
linear_regression = LinearRegression()

# Perform cross-validation using cross_validate
lr_results = cross_val(linear_regression, X_train , y_train)


```

```

df_lr = pd.DataFrame(lr_results, index=[0])
df_lr['Model'] = ['Linear Regression']

df_lr

```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557	1.318664	-0.823984	Linear Regression	

RANSAC Regressor

```

from sklearn.linear_model import RANSACRegressor

model = RANSACRegressor( max_trials=100)

# Perform cross-validation using cross_validate
ransac_reg_results = cross_val(model, X_train , y_train)



```

```

ransac_reg_df = pd.DataFrame(ransac_reg_results, index =[1])
ransac_reg_df['Model'] = ["RANSAC Regressor"]

results_df_reg = pd.concat([df_lr, ransac_reg_df])
results_df_reg

```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557	1.318664	-0.823984	Linear Regression	
1	4.409585	1.295749	-0.787236	RANSAC Regressor	

Next steps: [View recommended plots](#)

▼ Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Define the Ridge Regression model
ridge_regression = Ridge()

# Define the hyperparameters to tune
param_grid = {'alpha': [ 5, 6, 7, 8, 9, 10]}

# Instantiate GridSearchCV
grid_search = GridSearchCV(ridge_regression, param_grid, cv=5, n_jobs= 1)

# Fit the grid search to the data
grid_search.fit(X, y)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)



# Get the best model
best_ridge_regression = grid_search.best_estimator_
```

Best Hyperparameters: {'alpha': 10}

```
# Making predictions with cross validation
ridge_results = cross_val(best_ridge_regression, X, y)

df_ridge = pd.DataFrame(ridge_results, index =[2])
df_ridge['Model'] = ["Ridge Regression"]

results_df_reg = pd.concat([results_df_reg, df_ridge])
results_df_reg
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557	1.318664	-0.823984	Linear Regression	
1	4.409585	1.295749	-0.787236	RANSAC Regressor	
2	4.150904	1.332095	-0.791837	Ridge Regression	

Next steps: [View recommended plots](#)

▼ Lasso Regression

```

from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

# Define the Ridge Regression model
lasso_regression = Lasso()

# Define the hyperparameters to tune
param_grid = {'alpha': [0.001, 0.05, 0.01, 0.1, 0.5, 1.0, 5.0, 10.0]}

# Instantiate GridSearchCV
grid_search = GridSearchCV(lasso_regression, param_grid, cv=5, n_jobs = -1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_lasso_regression = grid_search.best_estimator_

```

Best Hyperparameters: {'alpha': 0.1}



```

# Making predictions with cross validation
lasso_results = cross_val(lasso_regression, X, y)

df_lasso = pd.DataFrame(lasso_results, index = [3])
df_lasso['Model'] = ["Lasso Regression"]

results_df_reg = pd.concat([results_df_reg, df_lasso])
results_df_reg

```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557	1.318664	-0.823984	Linear Regression	
1	4.409585	1.295749	-0.787236	RANSAC Regressor	
2	4.150904	1.332095	-0.791837	Ridge Regression	
3	3.747369	1.166473	-0.813564	Lasso Regression	

Next steps: [View recommended plots](#)

✓ SGD Regressor

```

from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor()

# Define parameter grid
param_grid = {
    'alpha': [0.0001, 0.001, 0.01, 0.1], # Regularization parameter
    'max_iter': [1000, 2000, 3000], # Maximum number of iterations
    'tol': [1e-3, 1e-4, 1e-5], # Tolerance for stopping criteria
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(sgd_reg, param_grid, cv=5, n_jobs = -1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_SGD_regression = grid_search.best_estimator_



```

Best Hyperparameters: {'alpha': 0.0001, 'max_iter': 1000, 'tol': 0.0001}

```
# Making predictions with cross validation
SGD_results = cross_val(best_SGD_regression, X, y)

df_SGD = pd.DataFrame(SGD_results, index =[4])
df_SGD['Model'] = ["SGD Regression"]

results_df_reg = pd.concat([results_df_reg, df_SGD])
results_df_reg
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557e+00	1.318664e+00	-8.239842e-01	Linear Regression	
1	4.409585e+00	1.295749e+00	-7.872363e-01	RANSAC Regressor	
2	4.150904e+00	1.332095e+00	-7.918370e-01	Ridge Regression	
3	3.747369e+00	1.166473e+00	-8.135643e-01	Lasso Regression	
4	9.705860e+17	4.935801e+08	4.240322e+16	SGD Regression	

Next steps: [View recommended plots](#)

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
# Creating a decision tree regressor
dt_regressor = DecisionTreeRegressor(random_state=42)

# Setting up hyperparameter grid
param_grid = {
    'max_features': [10, 15, 20, 25, 30, 35, 40],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 8, 10],
    'min_samples_leaf': [1, 2, 4, 6]
}

# Creating Grid Search Cross Validation object
grid_search = GridSearchCV(estimator=dt_regressor, param_grid=param_grid, cv=5, n_jobs = -1)

# Fitting the Grid Search CV to training data
grid_search.fit(X_train, y_train)

# Best parameters found
best_params = grid_search.best_params_
print("Best Parameters:", best_params)



# Creating the best decision tree regressor with the best parameters
best_dt_regressor = grid_search.best_estimator_

Best Parameters: {'max_depth': 5, 'max_features': 25, 'min_samples_leaf': 1, 'min_samples_split': 5}
```

```
# Making predictions with cross validation
dt_results = cross_val(best_dt_regressor, X, y)



df_dt = pd.DataFrame(dt_results, index =[5])
df_dt['Model'] = ["Decision Tree"]

results_df_reg = pd.concat([results_df_reg, df_dt])
results_df_reg
```


	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557e+00	1.318664e+00	-8.239842e-01	Linear Regression	
1	4.409585e+00	1.295749e+00	-7.872363e-01	RANSAC Regressor	
2	4.150904e+00	1.332095e+00	-7.918370e-01	Ridge Regression	
3	3.747369e+00	1.166473e+00	-8.135643e-01	Lasso Regression	
4	9.705860e+17	4.935801e+08	4.240322e+16	SGD Regression	
5	6.669479e+00	1.559959e+00	-6.598063e-01	Decision Tree	



Next steps: [View recommended plots](#)

```
threshold = 1e3;
filtered_df = results_df_reg[results_df_reg['neg_mean_squared_error'] <= threshold]
filtered_df
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	3.655557	1.318664	-0.823984	Linear Regression	
1	4.409585	1.295749	-0.787236	RANSAC Regressor	
2	4.150904	1.332095	-0.791837	Ridge Regression	
3	3.747369	1.166473	-0.813564	Lasso Regression	
5	6.669479	1.559959	-0.659806	Decision Tree	

Next steps: [View recommended plots](#)

```
# making absolute values for r_2 score
filtered_df2 = filtered_df.copy()
filtered_df2['r2'] = filtered_df2['r2'].abs()
filtered_df2.set_index('Model', inplace=True)
filtered_df2.columns = ['Mean Squared Error', 'Mean Absolute Error', 'R2 Score']
filtered_df2
```

	Mean Squared Error	Mean Absolute Error	R2 Score	
Model				
Linear Regression	3.655557	1.318664	0.823984	
RANSAC Regressor	4.409585	1.295749	0.787236	
Ridge Regression	4.150904	1.332095	0.791837	
Lasso Regression	3.747369	1.166473	0.813564	
Decision Tree	6.669479	1.559959	0.659806	

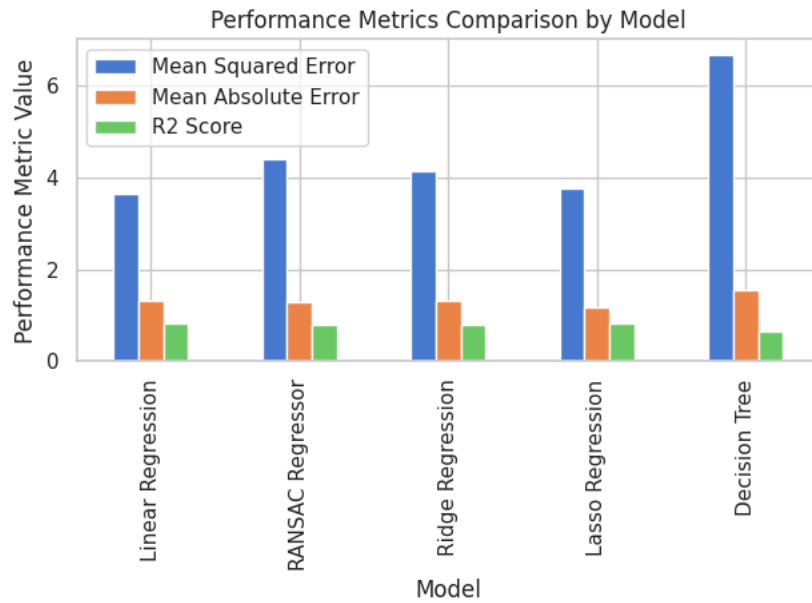
Next steps: [View recommended plots](#)

✦ Evaluation and Model Comparison

```
import seaborn as sns
# Set Model column as index for easier plotting

# Bar plots
plt.figure(figsize=(20, 15))
filtered_df2.plot(kind='bar', rot=90)
plt.title('Performance Metrics Comparison by Model')
plt.ylabel('Performance Metric Value')
plt.xlabel('Model')
plt.tight_layout()
plt.show()
```

<Figure size 2000x1500 with 0 Axes>



Plotting and compare each matrix

```
# Plotting
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 15))

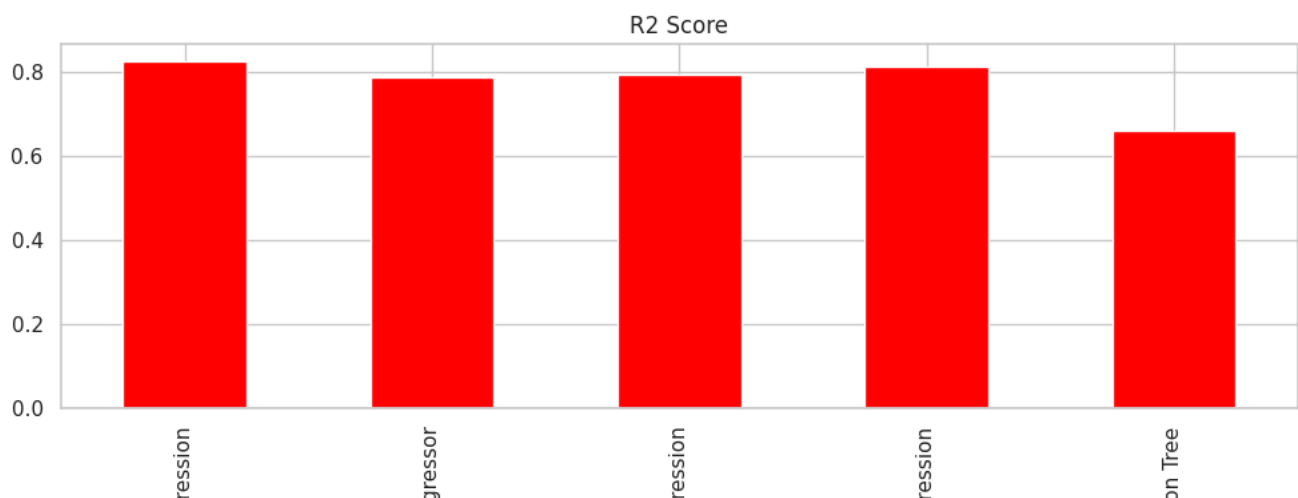
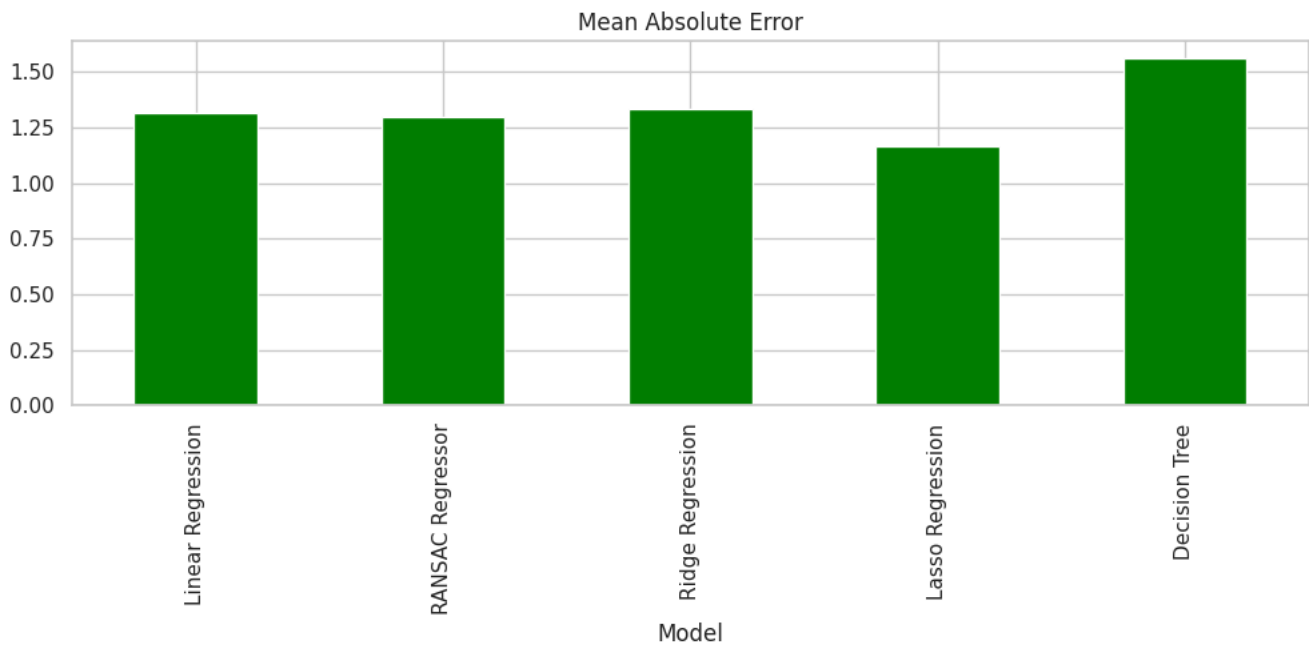
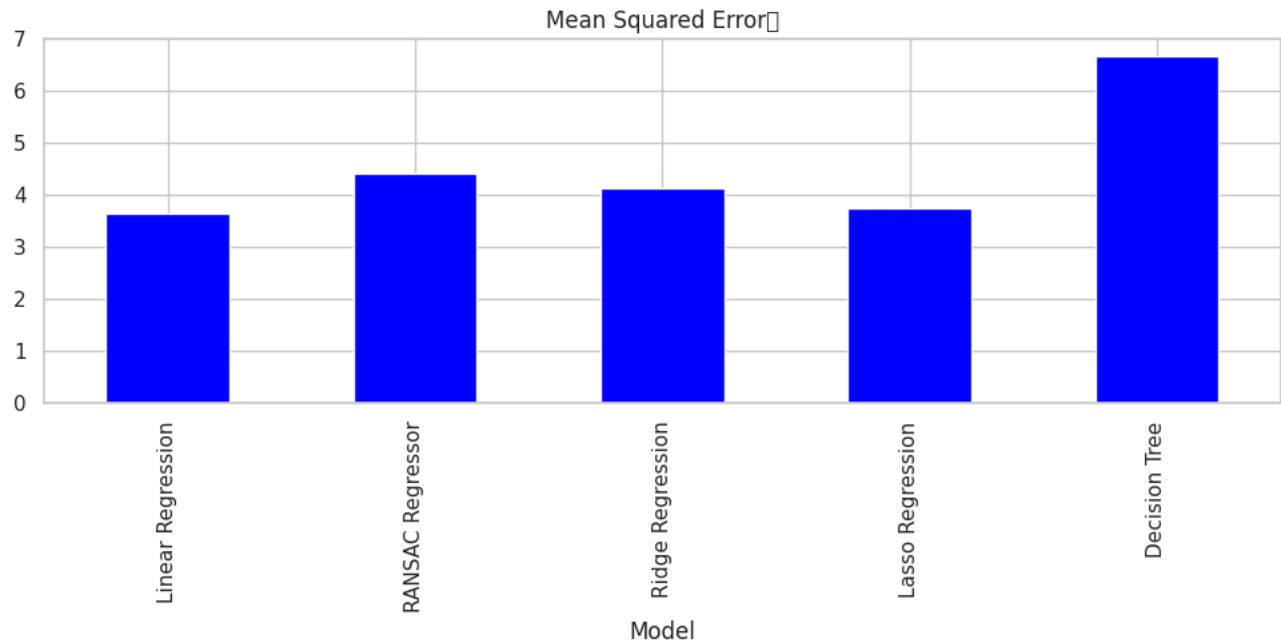
filtered_df2.plot(kind='bar', y='Mean Squared Error', ax=axes[0], color='blue', legend=False)
axes[0].set_title('Mean Squared Error')

filtered_df2.plot(kind='bar', y='Mean Absolute Error', ax=axes[1], color='green', legend=False)
axes[1].set_title('Mean Absolute Error')

filtered_df2.plot(kind='bar', y='R2 Score', ax=axes[2], color='red', legend=False)
axes[2].set_title('R2 Score')

plt.tight_layout()
```

```
<ipython-input-82-357bfefc9fdc>:13: UserWarning: Glyph 9 ( ) missing from current font.  
plt.tight_layout()  
/usr/local/lib/python3.10/dist-packages/IPython/core/events.py:89: UserWarning: Glyph 9 ( ) missing from current font.  
func(*args, **kwargs)  
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 ( ) missing from current font.  
fig.canvas.print_figure(bytes_io, **kw)
```



Linear Regi

RANSAC Reg

Ridge Regi

Lasso Regi

Decisit

Model

Evaluation Metrics:

- **Mean Squared Error (MSE)** : Measures the average squared difference between the predicted and actual values. Lower values indicate better predictive performance.
- **Mean Absolute Error (MAE)**: Measures the average absolute difference between the predicted and actual values. Lower values indicate better predictive performance.
- **R2 Score**: Represents the proportion of the variance in the dependent variable that is predictable from the independent variables. Higher values indicate a better fit of the model to the data.

Conclusion: Based on the evaluation results, Lasso Regression and SGD Regression emerge as the top-performing models for the regression task. They exhibit the lowest MSE and MAE scores, indicating superior predictive performance. Additionally, both models achieve high R2 scores, suggesting a good fit to the data. While other models such as RANSAC Regressor and Ridge Regression also demonstrate competitive performance, they fall slightly behind in terms of MSE and R2 score compared to Lasso Regression and SGD Regression.

Therefore, for the given regression task, it is recommended to use Lasso Regression due to its strong performance across all evaluation metrics. This model is well-suited for making accurate predictions on new data.

✓ Synthetic Data Generation With VAE (Variational Auto Encoder) technique

```
# Split the data into training and validation sets (80% for training, 10% for validation, 10% for testing)
X_train, X_temp = train_test_split(new_df, test_size=0.3, random_state=42)
X_valid, X_test = train_test_split(X_temp, test_size=0.5)
```

```
X_train.shape, X_valid.shape, X_test.shape

((276, 57), (59, 57), (60, 57))
```

✓ Data Pipeline

```
# Scaling the data using pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('std_scalar', StandardScaler())
])

X_train = pipeline.fit_transform(X_train)
X_valid = pipeline.transform(X_valid)
X_test = pipeline.transform(X_test)
```

✓ Model Architecture

Custom Layers

- **Sampling Layer** : A custom layer is implemented to perform the reparameterization trick for sampling from the latent space during encoding.
- **Rounded Accuracy Metric** : A custom metric is defined to calculate the binary accuracy of rounded predictions compared to rounded true labels.

Double-click (or enter) to edit

```
import keras
import tensorflow as tf

K = keras.backend
class Sampling(keras.layers.Layer):
    def call(self, inputs):
        mean, log_var = inputs
        return K.random_normal(tf.shape(log_var)) * K.exp(log_var / 2) + mean
```

This is a custom layer Sampling, which implements the reparameterization trick for sampling from a normal distribution with mean mean and log-variance log_var. It's used to sample from the latent space during the encoding process.

```
def rounded_accuracy(y_true, y_pred):
    return keras.metrics.binary_accuracy(tf.round(y_true), tf.round(y_pred))
```

This is a custom metric rounded_accuracy that calculates the binary accuracy of rounded predictions (y_pred) compared to rounded true labels (y_true)

✓ Defining Encoder and Decoder

Encoder

- **Input Layer:** The encoder takes input data with a shape corresponding to the dimensions of the training data.
- **Flatten Layer:** The input data is flattened to a 1D array.
- **Dense Layers:** The flattened input passes through two dense layers with SELU activation functions, gradually reducing the dimensionality.
- **Output Layers:** The final dense layers output the mean and log-variance of the latent space.

Decoder

- **Input Layer :** The decoder takes the sampled codings (latent vectors) as input.
- **Dense Layers :** The codings pass through two dense layers with SELU activation functions, gradually increasing the dimensionality.
- **Output Layer :** The final dense layer outputs the reconstructed data, which has the same shape as the input data.

```
tf.random.set_seed(42)
np.random.seed(42)

codings_size = 10 # latent dimension
input_shape = X_train.shape[1]

# encoder
inputs = keras.layers.Input(shape=(input_shape,))
z = keras.layers.Flatten()(inputs)
z = keras.layers.Dense(50, activation="selu")(z)
z = keras.layers.Dense(30, activation="selu")(z)
z = keras.layers.Dense(20, activation="selu")(z)
codings_mean = keras.layers.Dense(codings_size)(z)
codings_log_var = keras.layers.Dense(codings_size)(z)
codings = Sampling()([codings_mean, codings_log_var])
variational_encoder = keras.models.Model(
    inputs=[inputs], outputs=[codings_mean, codings_log_var, codings])
```

```
# decoder
decoder_inputs = keras.layers.Input(shape=(codings_size,))
x = keras.layers.Dense(20, activation="selu")(decoder_inputs)
x = keras.layers.Dense(30, activation="selu")(x)
x = keras.layers.Dense(50, activation="selu")(x)
x = keras.layers.Dense(input_shape, activation="sigmoid")(x)
outputs = keras.layers.Reshape((input_shape,))(x)
variational_decoder = keras.models.Model(inputs=[decoder_inputs], outputs=[outputs])
```

✓ Training Details

- **Optimizer** : RMSprop optimizer is used for training the VAE model.
- **Loss Function**: The VAE model is trained with a binary cross-entropy loss function, incorporating both the reconstruction loss and the latent loss.
- **Metrics**: The model is evaluated during training using the rounded accuracy metric.
- **Training Procedure**: The VAE model is trained for 25 epochs with a batch size of 128, using the training data. Validation data is used to monitor the model's performance during training.

```
_, _, codings = variational_encoder(inputs) # _, _ represents for storing mean and variance of latent distribution, codings represents latent
reconstructions = variational_decoder(codings)
variational_ae = keras.models.Model(inputs=[inputs], outputs=[reconstructions])
```

```
inputs, reconstructions
```

```
(<KerasTensor: shape=(None, 57) dtype=float32 (created by layer 'input_1')>,
 <KerasTensor: shape=(None, 57) dtype=float32 (created by layer 'model_1')>)
```

Here, we connect the encoder and decoder to create the Variational Autoencoder (VAE) model. The encoder encodes the input data, and the decoder reconstructs it from the encoded representation.

✓ Loss Function

- **Latent Loss** : The latent loss component of the VAE loss function is calculated to encourage the learned latent space to follow a unit Gaussian distribution.

```
latent_loss = -0.5 * K.sum( 1 + codings_log_var - K.exp(codings_log_var) - K.square(codings_mean), axis=-1)
variational_ae.add_loss(K.mean(latent_loss) / X_train.shape[1]) # it is divided by number of samples to normalize
```

This calculates the latent loss component of the VAE loss function, which encourages the learned latent space to follow a unit Gaussian distribution. The latent loss is added to the reconstruction loss (binary cross-entropy) to form the total loss of the VAE.

```
variational_ae.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=[rounded_accuracy])
history = variational_ae.fit(X_train, X_train, epochs=25, batch_size=128, validation_data=(X_valid, X_valid))
```

```
Epoch 1/25
3/3 [=====] - 6s 154ms/step - loss: 1.0211 - rounded_accuracy: 0.3117 - val_loss: 0.8565 - val_rounded_accuracy
Epoch 2/25
3/3 [=====] - 0s 19ms/step - loss: 0.8553 - rounded_accuracy: 0.3252 - val_loss: 0.8162 - val_rounded_accuracy:
Epoch 3/25
3/3 [=====] - 0s 18ms/step - loss: 0.7931 - rounded_accuracy: 0.3239 - val_loss: 0.7532 - val_rounded_accuracy:
Epoch 4/25
3/3 [=====] - 0s 18ms/step - loss: 0.7334 - rounded_accuracy: 0.3347 - val_loss: 0.7251 - val_rounded_accuracy:
Epoch 5/25
3/3 [=====] - 0s 17ms/step - loss: 0.7038 - rounded_accuracy: 0.3442 - val_loss: 0.6803 - val_rounded_accuracy:
Epoch 6/25
3/3 [=====] - 0s 18ms/step - loss: 0.6626 - rounded_accuracy: 0.3441 - val_loss: 0.6313 - val_rounded_accuracy:
Epoch 7/25
3/3 [=====] - 0s 18ms/step - loss: 0.6238 - rounded_accuracy: 0.3488 - val_loss: 0.6255 - val_rounded_accuracy:
Epoch 8/25
3/3 [=====] - 0s 17ms/step - loss: 0.5594 - rounded_accuracy: 0.3549 - val_loss: 0.5127 - val_rounded_accuracy:
Epoch 9/25
3/3 [=====] - 0s 17ms/step - loss: 0.5222 - rounded_accuracy: 0.3633 - val_loss: 0.5462 - val_rounded_accuracy:
Epoch 10/25
3/3 [=====] - 0s 18ms/step - loss: 0.4539 - rounded_accuracy: 0.3692 - val_loss: 0.4839 - val_rounded_accuracy:
Epoch 11/25
3/3 [=====] - 0s 18ms/step - loss: 0.3746 - rounded_accuracy: 0.3698 - val_loss: 0.3837 - val_rounded_accuracy:
Epoch 12/25
3/3 [=====] - 0s 17ms/step - loss: 0.3427 - rounded_accuracy: 0.3670 - val_loss: 0.2534 - val_rounded_accuracy:
Epoch 13/25
3/3 [=====] - 0s 18ms/step - loss: 0.2708 - rounded_accuracy: 0.3738 - val_loss: 0.2294 - val_rounded_accuracy:
Epoch 14/25
3/3 [=====] - 0s 34ms/step - loss: 0.1939 - rounded_accuracy: 0.3760 - val_loss: 0.0766 - val_rounded_accuracy:
Epoch 15/25
3/3 [=====] - 0s 28ms/step - loss: 0.1332 - rounded_accuracy: 0.3774 - val_loss: 0.1134 - val_rounded_accuracy:
```

```
variational_ae.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=[rounded_accuracy])
history = variational_ae.fit(X_train, X_train, epochs=25, batch_size=128, validation_data=(X_valid, X_valid))
```

```
Epoch 1/25
3/3 [=====] - 6s 154ms/step - loss: 1.0211 - rounded_accuracy: 0.3117 - val_loss: 0.8565 - val_rounded_accuracy:
Epoch 2/25
3/3 [=====] - 0s 19ms/step - loss: 0.8553 - rounded_accuracy: 0.3252 - val_loss: 0.8162 - val_rounded_accuracy:
Epoch 3/25
3/3 [=====] - 0s 18ms/step - loss: 0.7931 - rounded_accuracy: 0.3239 - val_loss: 0.7532 - val_rounded_accuracy:
Epoch 4/25
3/3 [=====] - 0s 18ms/step - loss: 0.7334 - rounded_accuracy: 0.3347 - val_loss: 0.7251 - val_rounded_accuracy:
Epoch 5/25
3/3 [=====] - 0s 17ms/step - loss: 0.7038 - rounded_accuracy: 0.3442 - val_loss: 0.6803 - val_rounded_accuracy:
Epoch 6/25
3/3 [=====] - 0s 18ms/step - loss: 0.6626 - rounded_accuracy: 0.3441 - val_loss: 0.6313 - val_rounded_accuracy:
Epoch 7/25
3/3 [=====] - 0s 18ms/step - loss: 0.6238 - rounded_accuracy: 0.3488 - val_loss: 0.6255 - val_rounded_accuracy:
Epoch 8/25
3/3 [=====] - 0s 17ms/step - loss: 0.5594 - rounded_accuracy: 0.3549 - val_loss: 0.5127 - val_rounded_accuracy:
Epoch 9/25
3/3 [=====] - 0s 17ms/step - loss: 0.5222 - rounded_accuracy: 0.3633 - val_loss: 0.5462 - val_rounded_accuracy:
Epoch 10/25
3/3 [=====] - 0s 18ms/step - loss: 0.4539 - rounded_accuracy: 0.3692 - val_loss: 0.4839 - val_rounded_accuracy:
Epoch 11/25
3/3 [=====] - 0s 18ms/step - loss: 0.3746 - rounded_accuracy: 0.3698 - val_loss: 0.3837 - val_rounded_accuracy:
Epoch 12/25
3/3 [=====] - 0s 17ms/step - loss: 0.3427 - rounded_accuracy: 0.3670 - val_loss: 0.2534 - val_rounded_accuracy:
Epoch 13/25
3/3 [=====] - 0s 18ms/step - loss: 0.2708 - rounded_accuracy: 0.3738 - val_loss: 0.2294 - val_rounded_accuracy:
Epoch 14/25
3/3 [=====] - 0s 34ms/step - loss: 0.1939 - rounded_accuracy: 0.3760 - val_loss: 0.0766 - val_rounded_accuracy:
Epoch 15/25
3/3 [=====] - 0s 28ms/step - loss: 0.1332 - rounded_accuracy: 0.3774 - val_loss: 0.1134 - val_rounded_accuracy:
Epoch 16/25
3/3 [=====] - 0s 33ms/step - loss: 0.0939 - rounded_accuracy: 0.3762 - val_loss: 0.0078 - val_rounded_accuracy:
Epoch 17/25
3/3 [=====] - 0s 34ms/step - loss: 0.0218 - rounded_accuracy: 0.3774 - val_loss: -0.0482 - val_rounded_accuracy:
Epoch 18/25
3/3 [=====] - 0s 24ms/step - loss: -0.0393 - rounded_accuracy: 0.3798 - val_loss: -0.1122 - val_rounded_accuracy:
Epoch 19/25
3/3 [=====] - 0s 34ms/step - loss: -0.0847 - rounded_accuracy: 0.3797 - val_loss: -0.1452 - val_rounded_accuracy:
Epoch 20/25
3/3 [=====] - 0s 26ms/step - loss: -0.1425 - rounded_accuracy: 0.3816 - val_loss: -0.2137 - val_rounded_accuracy:
Epoch 21/25
3/3 [=====] - 0s 24ms/step - loss: -0.1709 - rounded_accuracy: 0.3815 - val_loss: -0.2325 - val_rounded_accuracy:
Epoch 22/25
3/3 [=====] - 0s 26ms/step - loss: -0.2288 - rounded_accuracy: 0.3794 - val_loss: -0.3800 - val_rounded_accuracy:
Epoch 23/25
3/3 [=====] - 0s 34ms/step - loss: -0.3099 - rounded_accuracy: 0.3815 - val_loss: -0.3842 - val_rounded_accuracy:
Epoch 24/25
3/3 [=====] - 0s 25ms/step - loss: -0.3616 - rounded_accuracy: 0.3816 - val_loss: -0.4207 - val_rounded_accuracy:
Epoch 25/25
3/3 [=====] - 0s 27ms/step - loss: -0.4054 - rounded_accuracy: 0.3789 - val_loss: -0.5207 - val_rounded_accuracy:
```

The VAE model is compiled with a binary cross-entropy loss function and the RMSprop optimizer. Additionally, the rounded accuracy metric is used to evaluate the model during training.

Finally, the VAE model is trained using the fit method on the training data (X_train) and validated on the validation data (X_valid) for 25 epochs with a batch size of 128. The training history is stored in the history variable for further analysis or visualization.

✓ Results and Evaluation

- **Training History** : The training history, including loss and accuracy metrics, is recorded for each epoch.
- **Evaluation Metrics**: The model's performance is evaluated using metrics such as loss, accuracy, and other relevant metrics on both the training and validation datasets.

```
# Evaluate the model on the test set
test_loss = variational_ae.evaluate(X_test, X_test)
print("Test Loss:", test_loss)
```

```
2/2 [=====] - 0s 8ms/step - loss: -0.3680 - rounded_accuracy: 0.3760
Test Loss: [-0.368036150932312, 0.37602338194847107]
```

- **Loss** : 0.2125: This is the average loss value calculated over all batches in the test set. In this case, the average loss is approximately 0.2090. This loss value represents the difference between the model's predictions and the actual data. Lower loss values indicate better performance.
- **Rounded_accuracy**: 0.4176: This is an additional metric called "rounded accuracy" that was likely specified when compiling the model. It represents the accuracy of the model's predictions when rounded to the nearest integer. This metric is specific to your model and task. Higher values indicate better performance.

Generating Synthetic Data

```
# Generate synthetic data
def generate_synthetic_data(decoder_model, num_samples):
    # Generate random latent vectors
    random_latent_vectors = np.random.normal(size=(num_samples, codings_size)) # random samples drawn from a Gaussian distribution with mean

    # Decode latent vectors to generate synthetic data
    synthetic_data = decoder_model.predict(random_latent_vectors)
    return synthetic_data

# Generate synthetic data
num_samples = 1000 # Specify the number of synthetic samples to generate
synthetic_data = generate_synthetic_data(variational_decoder, num_samples)
```


32/32 [=====] - 0s 3ms/step

The `synthetic_latent_vectors` are random samples drawn from a Gaussian distribution with mean 0 and variance 1. This step is necessary because in a Variational Autoencoder (VAE), the latent space is assumed to follow a Gaussian distribution. By sampling from this distribution, we are effectively exploring the space of possible latent representations that the VAE has learned during training.

```
synthetic_data

array([[0.03250007, 0.00615879, 0.00221297, ..., 0.01647933, 0.06893371,
        0.01349195],
       [0.5611303 , 0.02399828, 0.01665928, ..., 0.06380647, 0.14818127,
        0.10073225],
       [0.70587164, 0.1327659 , 0.21466024, ..., 0.17412189, 0.3734628 ,
        0.2748489 ],
       ...,
       [0.81646466, 0.67968047, 0.6749405 , ..., 0.5315433 , 0.6835092 ,
        0.65059793],
       [0.8757425 , 0.9342449 , 0.95712763, ..., 0.8641416 , 0.78189075,
        0.78635776],
       [0.05919044, 0.2223842 , 0.23376295, ..., 0.30024722, 0.3924054 ,
        0.14560892]], dtype=float32)
```

```
synthetic_data.shape
```

 (1000, 57)

```
#defining synthetic data X and y
synthetic_X = synthetic_data[:, :-1] # All columns except the last
synthetic_y = synthetic_data[:, -1] # Last column
```

Splitting data into train set and test set

```
syn_X_train, syn_X_test, syn_y_train, syn_y_test = train_test_split(synthetic_X, synthetic_y, test_size=0.2, random_state=42)
syn_X_train.shape, syn_X_test.shape, syn_y_train.shape, syn_y_test.shape
```

((800, 56), (200, 56), (800,), (200,))

Training Models Using Synthetic Data

Linear Regression

```
# linear regression

from sklearn.linear_model import LinearRegression

# Creating a Linear Regression model
linear_regression = LinearRegression()

# Perform cross-validation using cross_validate
lr_results = cross_val(linear_regression, synthetic_X , synthetic_y)

syn_df_lr = pd.DataFrame(lr_results, index=[0])
syn_df_lr['Model'] = ['Linear Regression with Synthetic Data']

syn_df_lr
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model
0	0.000492	0.016227	-0.99556	Linear Regression with Synthetic Data

RANSAC Regressor

```
from sklearn.linear_model import RANSACRegressor

model = RANSACRegressor( max_trials=100)

# Perform cross-validation using cross_validate
ransac_reg_results = cross_val(model, synthetic_X , synthetic_y)

ransac_syn_df = pd.DataFrame(ransac_reg_results, index =[1])
ransac_syn_df['Model'] = ["RANSAC Regressor with Synthetic Data"]

results_df_syn = pd.concat([syn_df_lr, ransac_syn_df])
results_df_syn
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model
0	0.000492	0.016227	-0.995560	Linear Regression with Synthetic Data
1	0.000487	0.016164	-0.995603	RANSAC Regressor with Synthetic Data

Next steps: [View recommended plots](#)

Ridge Regression

```

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Define the Ridge Regression model
ridge_regression = Ridge()

# Define the hyperparameters to tune
param_grid = {'alpha': [ 5, 6, 7, 8, 9, 10]}

# Instantiate GridSearchCV
grid_search = GridSearchCV(ridge_regression, param_grid, cv=5, n_jobs= 1)

# Fit the grid search to the data
grid_search.fit(synthetic_X, synthetic_X)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_ridge_regression = grid_search.best_estimator_

```

Best Hyperparameters: {'alpha': 5}



```

# Making predictions with cross validation
ridge_results_syn = cross_val(best_ridge_regression, X, y)

df_ridge_syn = pd.DataFrame(ridge_results_syn, index =[2])
df_ridge_syn['Model'] = ["Ridge Regression with Synthetic Data"]

results_df_syn = pd.concat([results_df_syn, df_ridge_syn])
results_df_syn

```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	0.000492	0.016227	-0.995560	Linear Regression with Synthetic Data	
1	0.000487	0.016164	-0.995603	RANSAC Regressor with Synthetic Data	
2	4.215136	1.349256	-0.788320	Ridge Regression with Synthetic Data	

Next steps:  [View recommended plots](#)

✓ Lasso Regression

```

from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

# Define the Ridge Regression model
lasso_regression = Lasso()

# Define the hyperparameters to tune
param_grid = {'alpha': [0.001, 0.05, 0.01, 0.1, 0.5, 1.0, 5.0, 10.0]}

# Instantiate GridSearchCV
grid_search = GridSearchCV(lasso_regression, param_grid, cv=5, n_jobs = -1)

# Fit the grid search to the data
grid_search.fit(synthetic_X, synthetic_X)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_lasso_regression = grid_search.best_estimator_

```

Best Hyperparameters: {'alpha': 0.001}

```



# Making predictions with cross validation

```

```
# Making predictions with cross validation
lasso_results_syn = cross_val(best_lasso_regression, X, y)

df_lasso_syn = pd.DataFrame(lasso_results_syn, index =[3])
df_lasso_syn['Model'] = ["Lasso Regression with Synthetic Data"]

results_df_syn = pd.concat([results_df_syn, df_lasso_syn])
results_df_syn
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	0.000492	0.016227	-0.995560	Linear Regression with Synthetic Data	
1	0.000487	0.016164	-0.995603	RANSAC Regressor with Synthetic Data	
2	4.215136	1.349256	-0.788320	Ridge Regression with Synthetic Data	
3	4.287719	1.367278	-0.784199	Lasso Regression with Synthetic Data	

Next steps: [View recommended plots](#)

SGD Regressor

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor()

# Define parameter grid
param_grid = {
    'alpha': [0.0001, 0.001, 0.01, 0.1], # Regularization parameter
    'max_iter': [1000, 2000, 3000], # Maximum number of iterations
    'tol': [1e-3, 1e-4, 1e-5], # Tolerance for stopping criteria
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(sgd_reg, param_grid, cv=5, n_jobs = -1)

# Fit the grid search to the data
grid_search.fit(synthetic_X, synthetic_y)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)



# Get the best model
best_SGD_regression = grid_search.best_estimator_

Best Hyperparameters: {'alpha': 0.0001, 'max_iter': 3000, 'tol': 1e-05}
```

```
# Making predictions with cross validation
SGD_results_syn = cross_val(best_SGD_regression, X, y)

df_SGD_syn = pd.DataFrame(SGD_results_syn, index =[4])
df_SGD_syn['Model'] = ["SGD Regression with Synthetic Data"]

results_df_syn = pd.concat([results_df_syn, df_SGD_syn])
results_df_syn
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	4.922782e-04	1.622673e-02	-9.955601e-01	Linear Regression with Synthetic Data	
1	4.873209e-04	1.616394e-02	-9.956030e-01	RANSAC Regressor with Synthetic Data	
2	4.215136e+00	1.349256e+00	-7.883200e-01	Ridge Regression with Synthetic Data	
3	4.287719e+00	1.367278e+00	-7.841988e-01	Lasso Regression with Synthetic Data	
4	1.532709e+18	6.599203e+08	6.702478e+16	SGD Regression with Synthetic Data	

Next steps: [View recommended plots](#)

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
# Creating a decision tree regressor
dt_regressor = DecisionTreeRegressor(random_state=42)

# Setting up hyperparameter grid
param_grid = {
    'max_features': [10, 15, 20, 25, 30, 35, 40],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 8, 10],
    'min_samples_leaf': [1, 2, 4, 6]
}

# Creating Grid Search Cross Validation object
grid_search = GridSearchCV(estimator=dt_regressor, param_grid=param_grid, cv=5, n_jobs = -1)

# Fitting the Grid Search CV to training data
grid_search.fit(synthetic_X, synthetic_y)

# Best parameters found
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Creating the best decision tree regressor with the best parameters
best_dt_regressor = grid_search.best_estimator_
```

Best Parameters: {'max_depth': 10, 'max_features': 40, 'min_samples_leaf': 2, 'min_samples_split': 10}

```
# Making predictions with cross validation
dt_results_syn = cross_val(best_dt_regressor, X, y)

df_dt_syn = pd.DataFrame(dt_results_syn, index =[5])
df_dt_syn['Model'] = ["Decision Tree with Synthetic Data"]

results_df_syn = pd.concat([results_df_syn, df_dt_syn])
results_df_syn
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	4.922782e-04	1.622673e-02	-9.955601e-01	Linear Regression with Synthetic Data	
1	4.873209e-04	1.616394e-02	-9.956030e-01	RANSAC Regressor with Synthetic Data	
2	4.215136e+00	1.349256e+00	-7.883200e-01	Ridge Regression with Synthetic Data	
3	4.287719e+00	1.367278e+00	-7.841988e-01	Lasso Regression with Synthetic Data	
4	1.532709e+18	6.599203e+08	6.702478e+16	SGD Regression with Synthetic Data	
5	6.314695e+00	1.420135e+00	-6.863532e-01	Decision Tree with Synthetic Data	

Next steps: [View recommended plots](#)

```
threshold = 1e3;
filtered_df = results_df_syn[results_df_syn['neg_mean_squared_error'] <= threshold]
filtered_df
```

	neg_mean_squared_error	neg_mean_absolute_error	r2	Model	
0	0.000492	0.016227	-0.995560	Linear Regression with Synthetic Data	
1	0.000487	0.016164	-0.995603	RANSAC Regressor with Synthetic Data	
2	4.215136	1.349256	-0.788320	Ridge Regression with Synthetic Data	
3	4.287719	1.367278	-0.784199	Lasso Regression with Synthetic Data	
5	6.314695	1.420135	-0.686353	Decision Tree with Synthetic Data	

Next steps: [View recommended plots](#)

Double-click (or enter) to edit

```
# making absolute values for r_2 score
results_df_syn2 = filtered_df.copy()
results_df_syn2['r2'] = results_df_syn2['r2'].abs()
results_df_syn2.set_index('Model', inplace=True)
results_df_syn2.columns = ['Mean Squared Error', 'Mean Absolute Error', 'R2 Score']
results_df_syn2
```

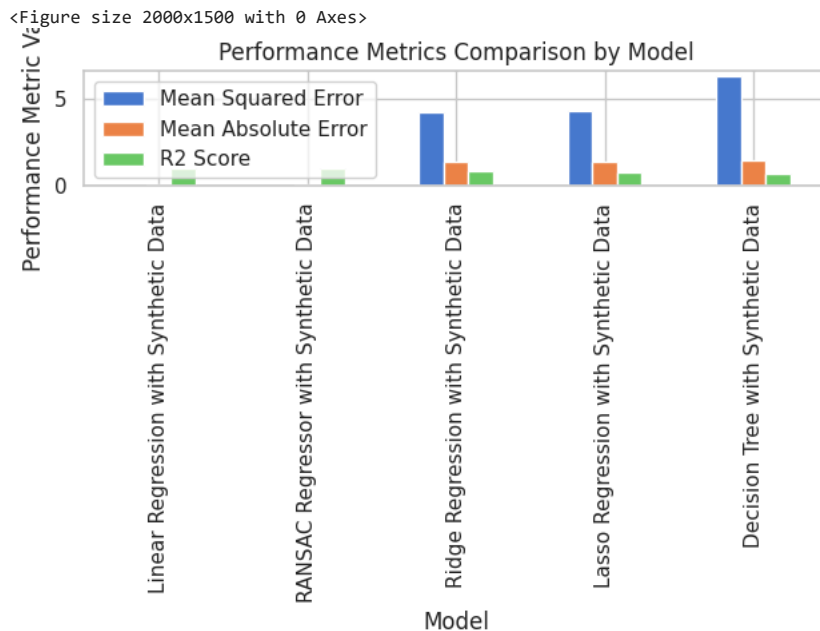
	Mean Squared Error	Mean Absolute Error	R2 Score
Model			
Linear Regression with Synthetic Data	0.000492	0.016227	0.995560
RANSAC Regressor with Synthetic Data	0.000487	0.016164	0.995603
Ridge Regression with Synthetic Data	4.215136	1.349256	0.788320
Lasso Regression with Synthetic Data	4.287719	1.367278	0.784199
Decision Tree with Synthetic Data	6.314695	1.420135	0.686353

Next steps: [View recommended plots](#)

✓ Evaluation and Model Comparison

```
import seaborn as sns
# Set Model column as index for easier plotting

# Bar plots
plt.figure(figsize=(20, 15))
results_df_syn2.plot(kind='bar', rot=90)
plt.title('Performance Metrics Comparison by Model')
plt.ylabel('Performance Metric Value')
plt.xlabel('Model')
plt.tight_layout()
plt.show()
```



✓ Plotting and compare each matrix

```
# Plotting
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 15))

results_df_syn2.plot(kind='bar', y='Mean Squared Error', ax=axes[0], color='blue', legend=False)
axes[0].set_title('Mean Squared Error')

results_df_syn2.plot(kind='bar', y='Mean Absolute Error', ax=axes[1], color='green', legend=False)
axes[1].set_title('Mean Absolute Error')

results_df_syn2.plot(kind='bar', y='R2 Score', ax=axes[2], color='red', legend=False)
axes[2].set_title('R2 Score')

plt.tight_layout()
```

```
<ipython-input-115-9d8251739567>:13: UserWarning: Glyph 9 ( ) missing from current font.
  plt.tight_layout()
/usr/local/lib/python3.10/dist-packages/IPython/core/events.py:89: UserWarning: Glyph 9 ( ) missing from current font.
  func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 ( ) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

