student-performance-prediction

June 3, 2024

1 Student Performance Prediction

For Prediction of Student's Performance, we will use following algorithms:

Linear Regression

Lasso Regression

Decision Tree Regressor

Random Forest Regressor

By using the above algorithms, will firstly explore the data that I have and check for any null or missing values. If found then I'll clean tha data and then visualize it for better understanding. Then I'll proceed by data training i.e. spliting data into training and testing data. Then train our model by providing training data and once the model will be trained, will perform prediction. After prediction, will evaluate the performance of these algorithmns by error check and accuracy check.

Steps followed are as:

Step 1: Data Exploration

Step 2: Data Visualization

Step 3: Data Tra ning

Step 4: Model Creation

Step 5: Performance Evaluation

1.1 Data Exploration

[1]:
```
import pandas as pd
```

1.1.1 Reading Files

[2]:
```
df=pd.read_csv('student-por.csv')
df
```

[2]:
```
    school sex  age address famsize Pstatus  Medu  Fedu     Mjob     Fjob  \
0       GP   F   18       U     GT3       A     4     4  at_home  teacher
1       GP   F   17       U     GT3       T     1     1  at_home    other
2       GP   F   15       U     LE3       T     1     1  at_home    other
```

```
3       GP  F  15       U    GT3       T     4     2   health  services
4       GP  F  16       U    GT3       T     3     3    other     other
..      …  ..  …      …      …    …    …       …       …
644     MS  F  19       R    GT3       T     2     3  services     other
645     MS  F  18       U    LE3       T     3     1   teacher  services
646     MS  F  18       U    GT3       T     1     1     other     other
647     MS  M  17       U    LE3       T     3     1  services  services
648     MS  M  18       R    LE3       T     3     2  services     other

     … famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0    …      4        3      4     1     1      3        4   0  11  11
1    …      5        3      3     1     1      3        2   9  11  11
2    …      4        3      2     2     3      3        6  12  13  12
3    …      3        2      2     1     1      5        0  14  14  14
4    …      4        3      2     1     2      5        0  11  13  13
..   …     …       …     …    …    …     …        …   ..  ..  ..
644  …      5        4      2     1     2      5        4  10  11  10
645  …      4        3      4     1     1      1        4  15  15  16
646  …      1        1      1     1     1      5        6  11  12   9
647  …      2        4      5     3     4      2        6  10  10  10
648  …      4        4      1     3     4      5        4  10  11  11

[649 rows x 33 columns]
```

[3]:
```
df.head()
```

[3]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  … \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
2     GP   F   15       U     LE3       T     1     1  at_home     other  …
3     GP   F   15       U     GT3       T     4     2   health  services  …
4     GP   F   16       U     GT3       T     3     3    other     other  …

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        4   0  11  11
1      5        3      3     1     1      3        2   9  11  11
2      4        3      2     2     3      3        6  12  13  12
3      3        2      2     1     1      5        0  14  14  14
4      4        3      2     1     2      5        0  11  13  13

[5 rows x 33 columns]
```

[4]:
```
df.tail()
```

[4]:
```
    school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
644     MS   F   19       R     GT3       T     2     3  services     other
645     MS   F   18       U     LE3       T     3     1   teacher  services
```

```
646     MS  F  18       U     GT3       T     1     1     other     other
647     MS  M  17       U     LE3       T     3     1  services  services
648     MS  M  18       R     LE3       T     3     2  services     other

     … famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
644  …      5        4      2     1     2      5        4  10  11  10
645  …      4        3      4     1     1      1        4  15  15  16
646  …      1        1      1     1     1      5        6  11  12   9
647  …      2        4      5     3     4      2        6  10  10  10
648  …      4        4      1     3     4      5        4  10  11  11

[5 rows x 33 columns]
```

[6]:
```
df.shape
```

[6]:
```
(649, 33)
```

[8]:
```
df.columns
```

[8]:
```
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'],
      dtype='object')
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64
 7   Fedu        649 non-null    int64
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64
 13  studytime   649 non-null    int64
```

```
 14  failures    649 non-null   int64
 15  schoolsup   649 non-null   object
 16  famsup      649 non-null   object
 17  paid        649 non-null   object
 18  activities  649 non-null   object
 19  nursery     649 non-null   object
 20  higher      649 non-null   object
 21  internet    649 non-null   object
 22  romantic    649 non-null   object
 23  famrel      649 non-null   int64
 24  freetime    649 non-null   int64
 25  goout       649 non-null   int64
 26  Dalc        649 non-null   int64
 27  Walc        649 non-null   int64
 28  health      649 non-null   int64
 29  absences    649 non-null   int64
 30  G1          649 non-null   int64
 31  G2          649 non-null   int64
 32  G3          649 non-null   int64
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

[11]:
```
nRow, nCol = df.shape
print(f'There are {nRow} rows and {nCol} columns')
df.head(5)
```

```
There are 649 rows and 33 columns
```

[11]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        4   0  11  11
1      5        3      3     1     1      3        2   9  11  11
2      4        3      2     2     3      3        6  12  13  12
3      3        2      2     1     1      5        0  14  14  14
4      4        3      2     1     2      5        0  11  13  13

[5 rows x 33 columns]
```

[12]:
```
df.info()
cat_cols   = df.select_dtypes(['object']).columns
int_cols   = df.select_dtypes(['int64']).columns
float_cols = df.select_dtypes(['float']).columns
```

```
print(cat_cols)
print(int_cols)
print(float_cols)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64
 7   Fedu        649 non-null    int64
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64
 13  studytime   649 non-null    int64
 14  failures    649 non-null    int64
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64
 24  freetime    649 non-null    int64
 25  goout       649 non-null    int64
 26  Dalc        649 non-null    int64
 27  Walc        649 non-null    int64
 28  health      649 non-null    int64
 29  absences    649 non-null    int64
 30  G1          649 non-null    int64
 31  G2          649 non-null    int64
 32  G3          649 non-null    int64
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
Index(['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
       'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
       'nursery', 'higher', 'internet', 'romantic'],
```

```
      dtype='object')
Index(['age', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failures', 'famrel',
       'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'G1', 'G2',
       'G3'],
      dtype='object')
Index([], dtype='object')
```

[14]:
```
import numpy as np
cat_cols=df.select_dtypes(include=['object']).columns
num_cols = df.select_dtypes(include=np.number).columns.tolist()
print("Categorical Variables:")
print(cat_cols)
print("Numerical Variables:")
print(num_cols)
```

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

[15]:
```
for col in df.columns:
    print(col, df[col].nunique())
```

```
school 2
sex 2
age 8
address 2
famsize 2
Pstatus 2
Medu 5
Fedu 5
Mjob 5
Fjob 5
reason 4
guardian 3
traveltime 4
studytime 4
failures 4
schoolsup 2
famsup 2
paid 2
activities 2
nursery 2
higher 2
```

```
internet 2
romantic 2
famrel 5
freetime 5
goout 5
Dalc 5
Walc 5
health 5
absences 24
G1 17
G2 16
G3 17
```

[16]:
```
#Checking duplicate line
duplicate = df.duplicated().any()
duplicate
```

[16]: False

[17]:
```
df.describe()
```

[17]:

	age	Medu	Fedu	traveltime	studytime	failures
count	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000
mean	16.744222	2.514638	2.306626	1.568567	1.930663	0.221880
std	1.218138	1.134552	1.099931	0.748660	0.829510	0.593235
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000
25%	16.000000	2.000000	1.000000	1.000000	1.000000	0.000000
50%	17.000000	2.000000	2.000000	1.000000	2.000000	0.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000

	famrel	freetime	goout	Dalc	Walc	health
count	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000
mean	3.930663	3.180277	3.184900	1.502311	2.280431	3.536210
std	0.955717	1.051093	1.175766	0.924834	1.284380	1.446259
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	4.000000	3.000000	2.000000	1.000000	1.000000	2.000000
50%	4.000000	3.000000	3.000000	1.000000	2.000000	4.000000
75%	5.000000	4.000000	4.000000	2.000000	3.000000	5.000000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

	absences	G1	G2	G3
count	649.000000	649.000000	649.000000	649.000000
mean	3.659476	11.399076	11.570108	11.906009
std	4.640759	2.745265	2.913639	3.230656
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	10.000000	10.000000	10.000000

```
50%       2.000000   11.000000   11.000000   12.000000
75%       6.000000   13.000000   13.000000   14.000000
max      32.000000   19.000000   19.000000   19.000000
```

[18]:
```
df.isnull().sum()
```

[18]:
```
school        0
sex           0
age           0
address       0
famsize       0
Pstatus       0
Medu          0
Fedu          0
Mjob          0
Fjob          0
reason        0
guardian      0
traveltime    0
studytime     0
failures      0
schoolsup     0
famsup        0
paid          0
activities    0
nursery       0
higher        0
internet      0
romantic      0
famrel        0
freetime      0
goout         0
Dalc          0
Walc          0
health        0
absences      0
G1            0
G2            0
G3            0
dtype: int64
```

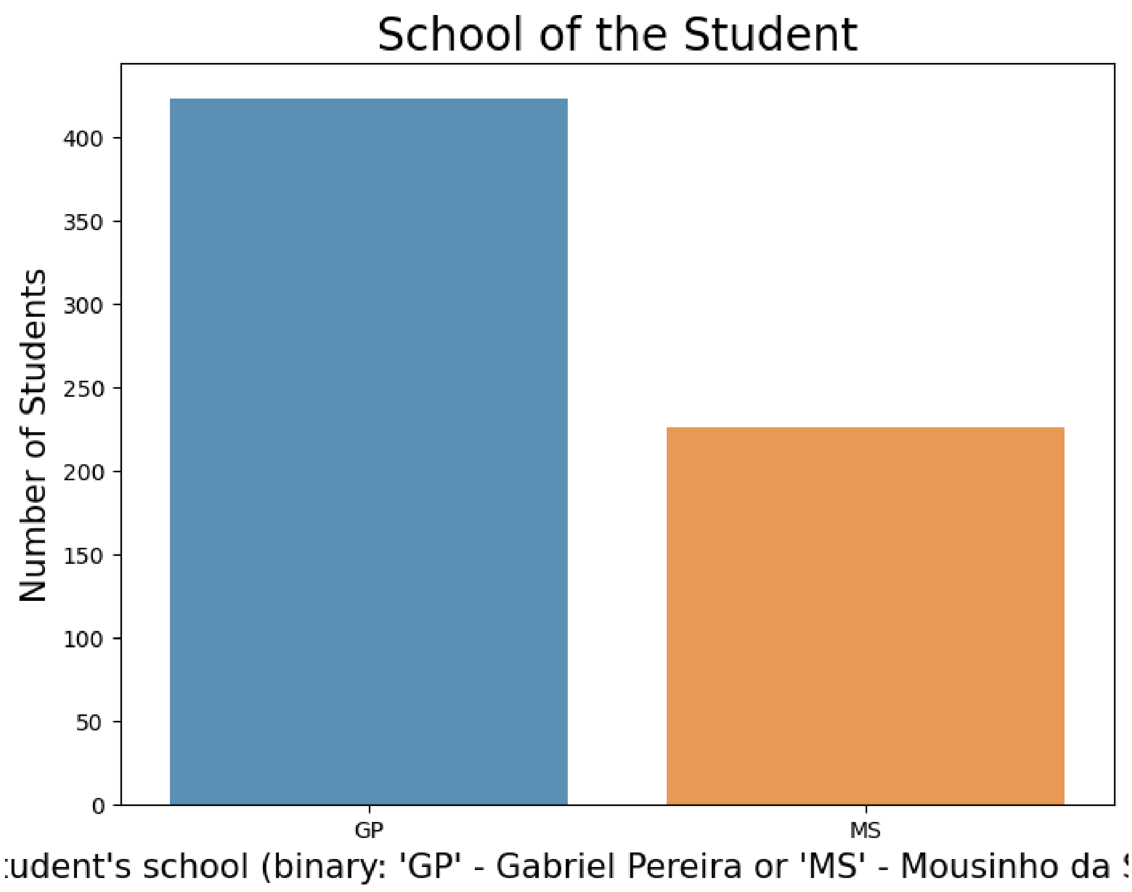
[19]:
```
df.isnull().sum().sum()
```

[19]:
```
0
```

1.2 Data Visualization

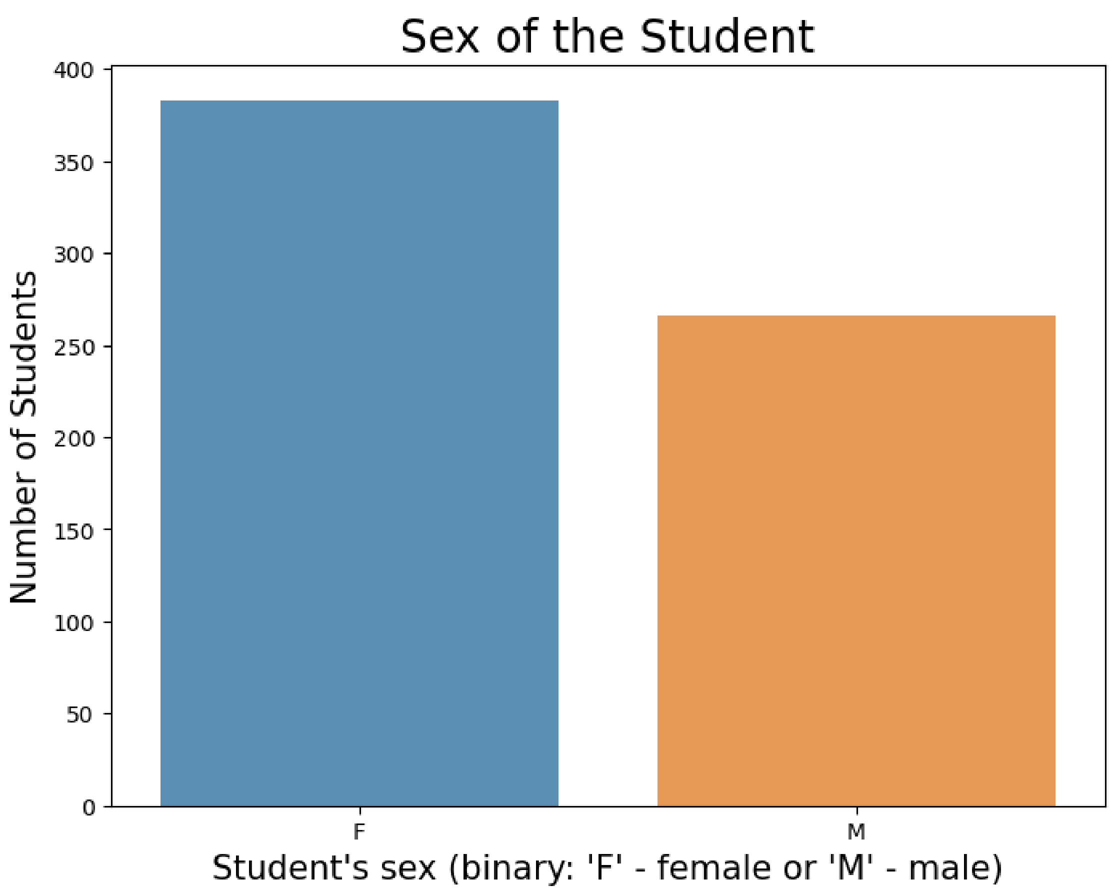
[20]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

[22]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Count the values
count = df['school'].value_counts()

# Plot the data
plt.figure(figsize=(8,6))
sns.barplot(x=count.index, y=count.values, alpha=0.8)
plt.title('School of the Student', fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho␣
↪da Silveira)", fontsize=15)
plt.show()
```

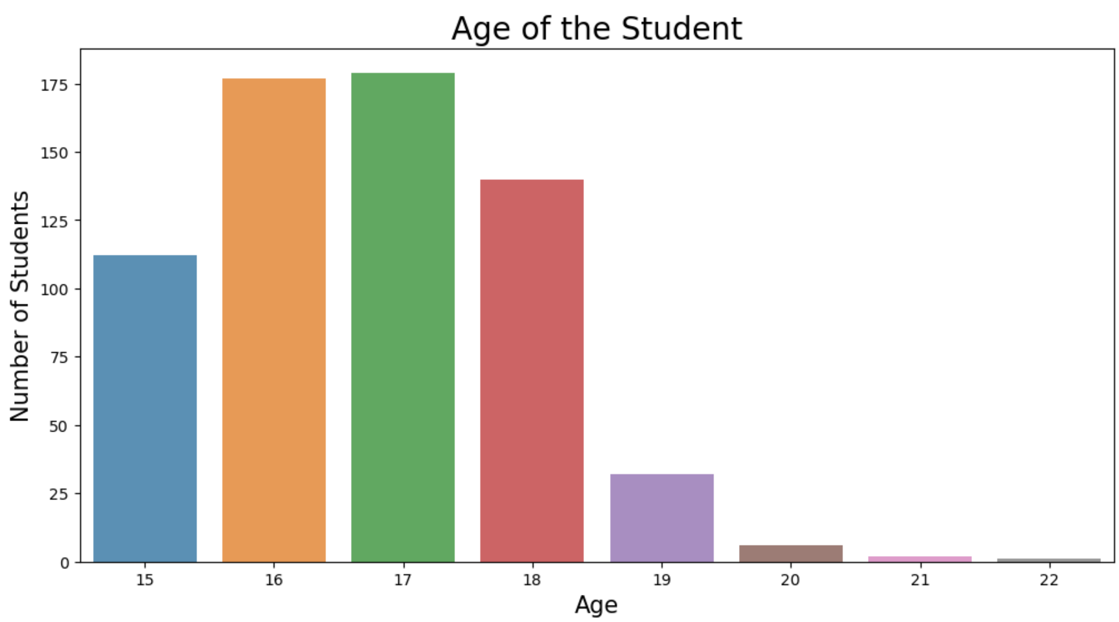


[24]:

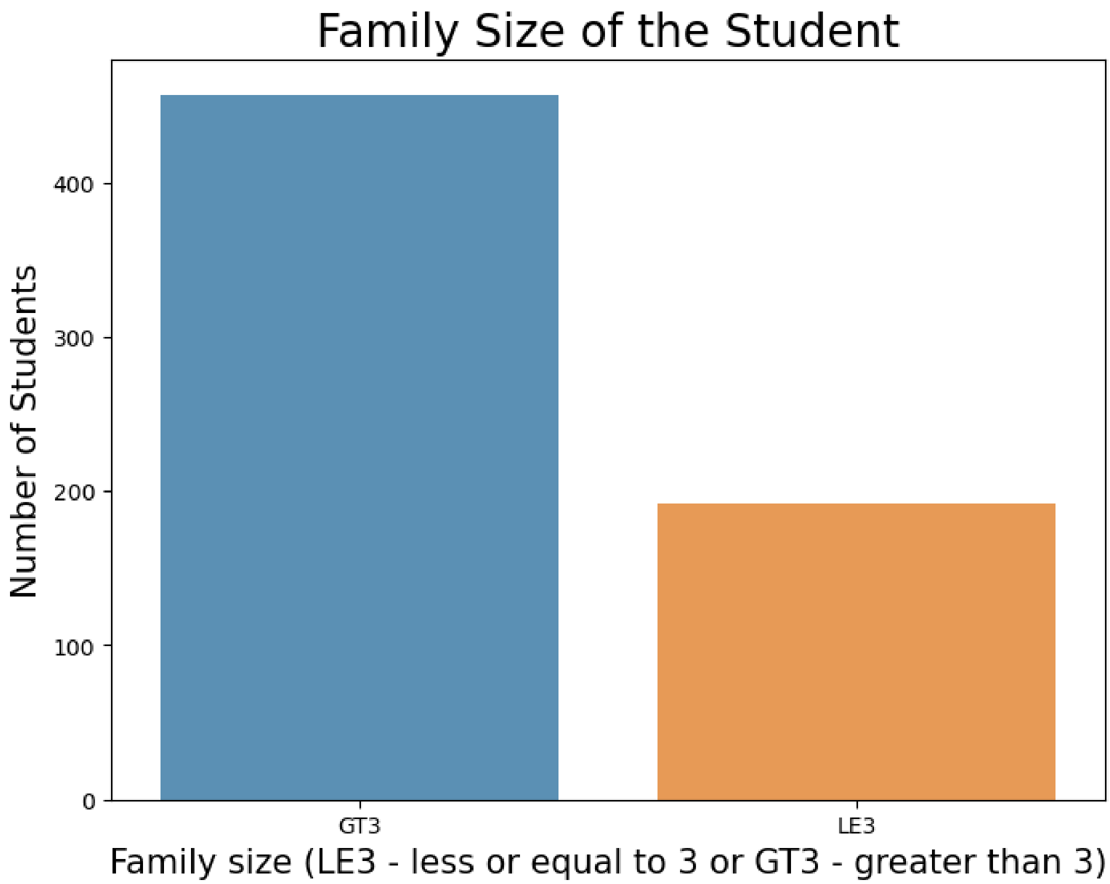
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Count the values
count = df['sex'].value_counts()

# Plot the data
plt.figure(figsize=(8,6))
sns.barplot(x=count.index, y=count.values, alpha=0.8)
plt.title('Sex of the Student', fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Student's sex (binary: 'F' - female or 'M' - male)", fontsize=15)
plt.show()
```



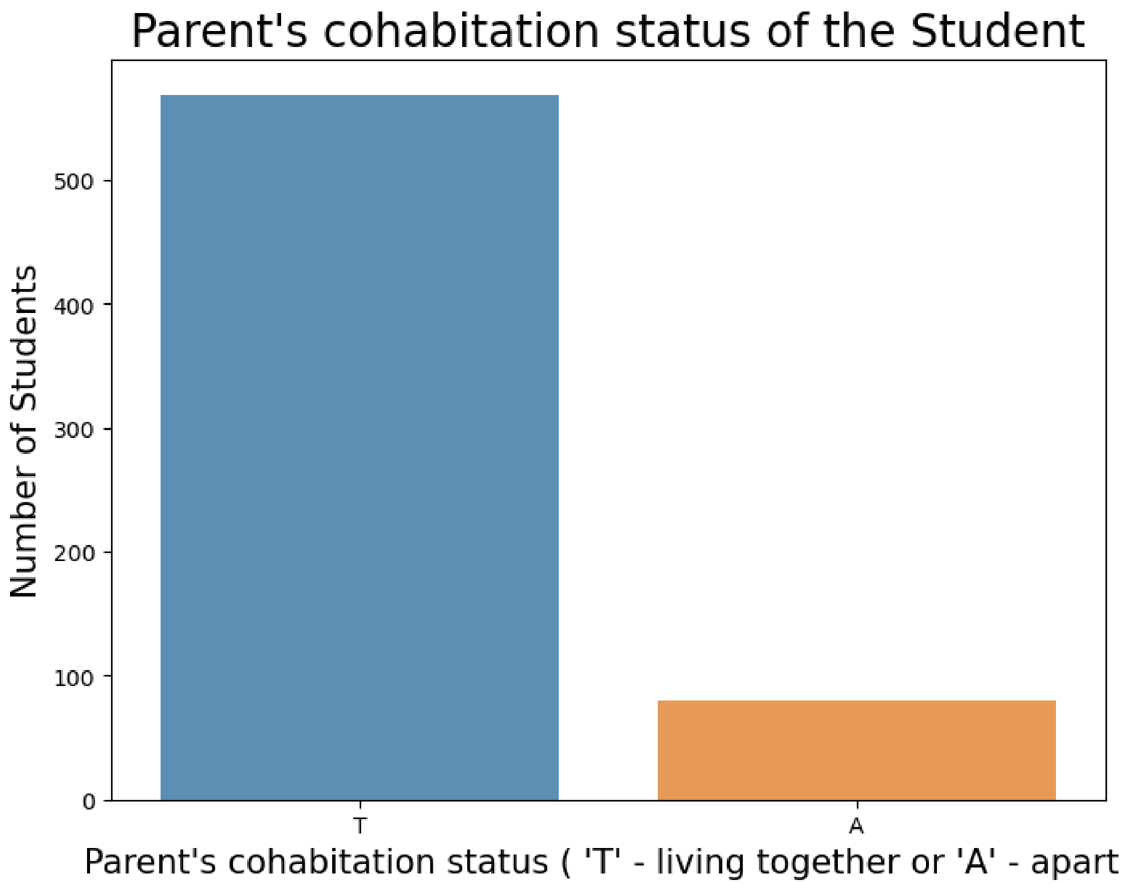
[25]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Count the values
count = df['age'].value_counts()

# Plot the data
plt.figure(figsize=(12,6))
sns.barplot(x=count.index, y=count.values, alpha=0.8)
plt.title('Age of the Student', fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel('Age', fontsize=15)
plt.show()
```



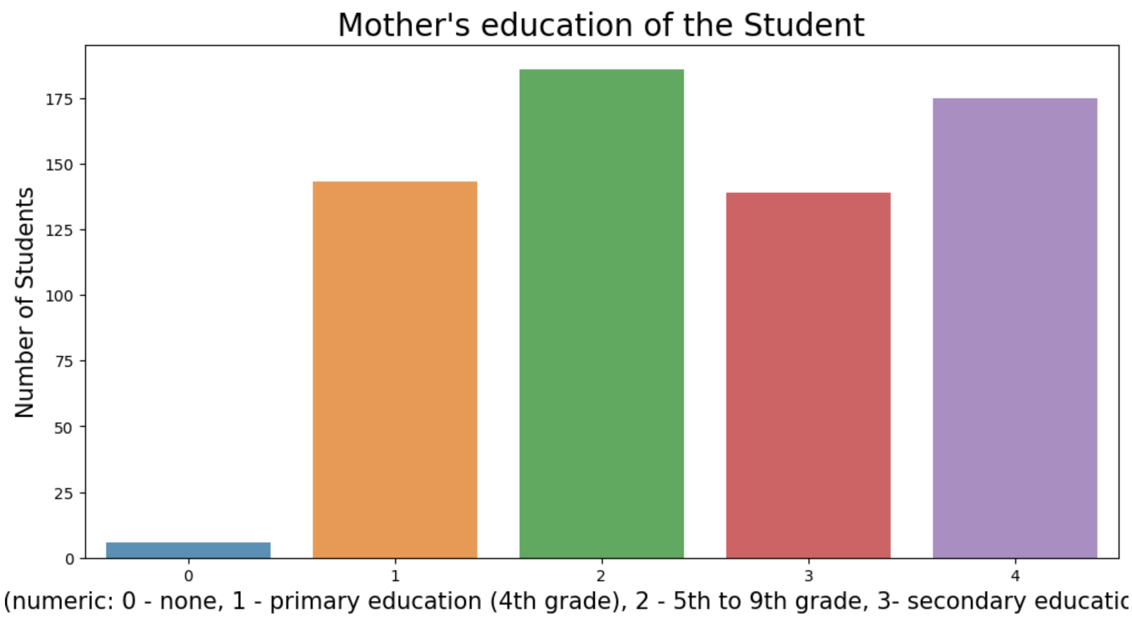
[26]:
```
# Count the values
count = df['famsize'].value_counts()

# Plot the data
plt.figure(figsize=(8,6))
sns.barplot(x=count.index, y=count.values, alpha=0.8)
plt.title('Family Size of the Student', fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel('Family size (LE3 - less or equal to 3 or GT3 - greater than 3)',␣
↪fontsize=15)
plt.show()
```



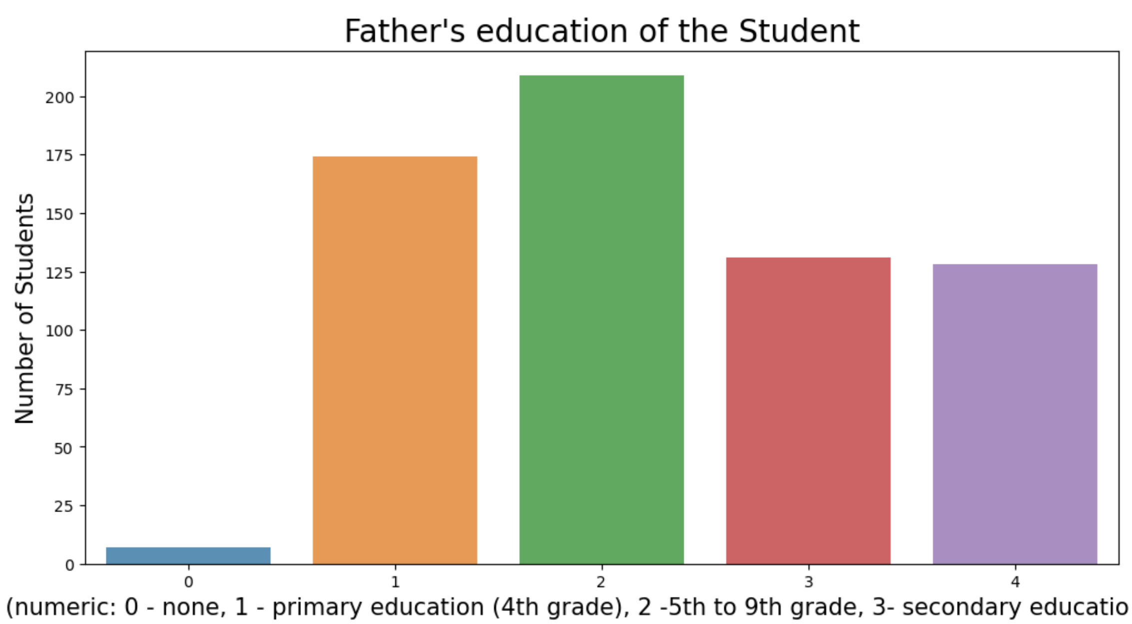
[27]:
```
count=df['Pstatus'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Parent's cohabitation status of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Parent's cohabitation status ( 'T' - living together or 'A' -␣
  ↪apart)", fontsize=15)
plt.show()
```



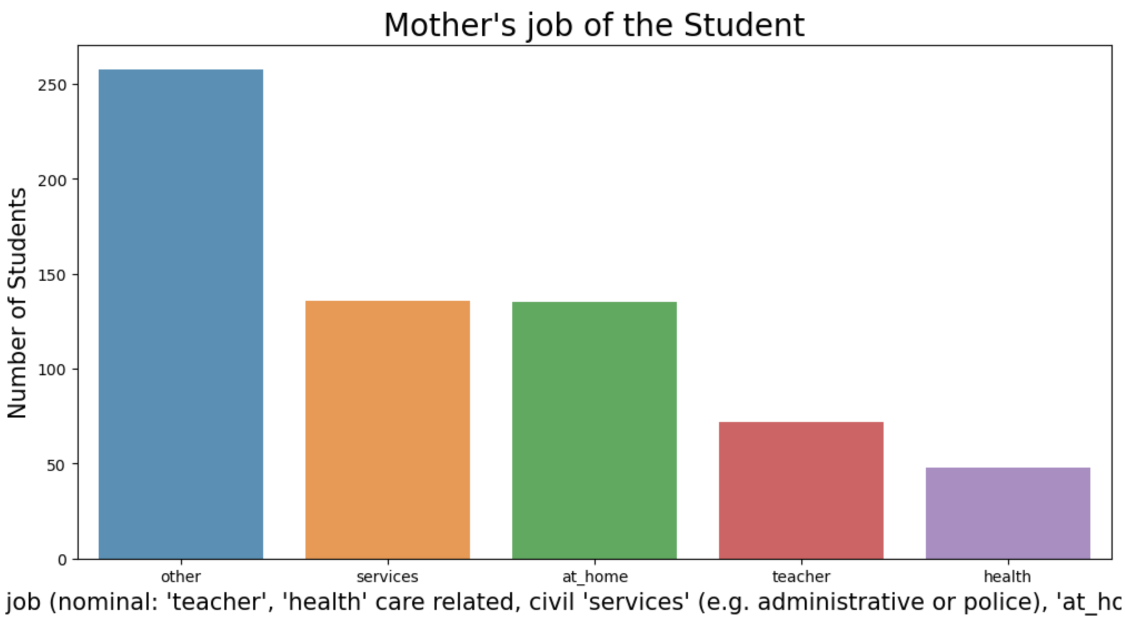
[28]:
```
count=df['Medu'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Mother's education of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Mother's education (numeric: 0 - none, 1 - primary education (4th␣
 ↪grade), 2 - 5th to 9th grade, 3- secondary education or 4 - higher␣
 ↪education)", fontsize=15)
plt.show()
```



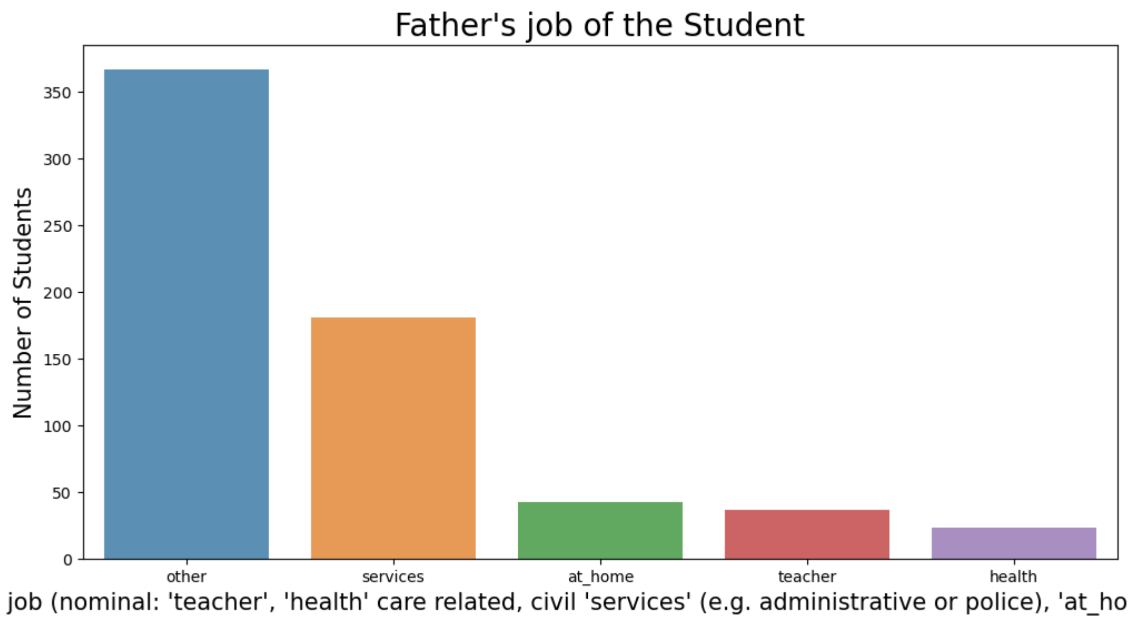
[29]:
```
count=df['Fedu'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Father's education of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Father's education (numeric: 0 - none, 1 - primary education (4th␣
↪grade), 2 -5th to 9th grade, 3- secondary education or 4- higher␣
↪education)", fontsize=15)
plt.show()
```



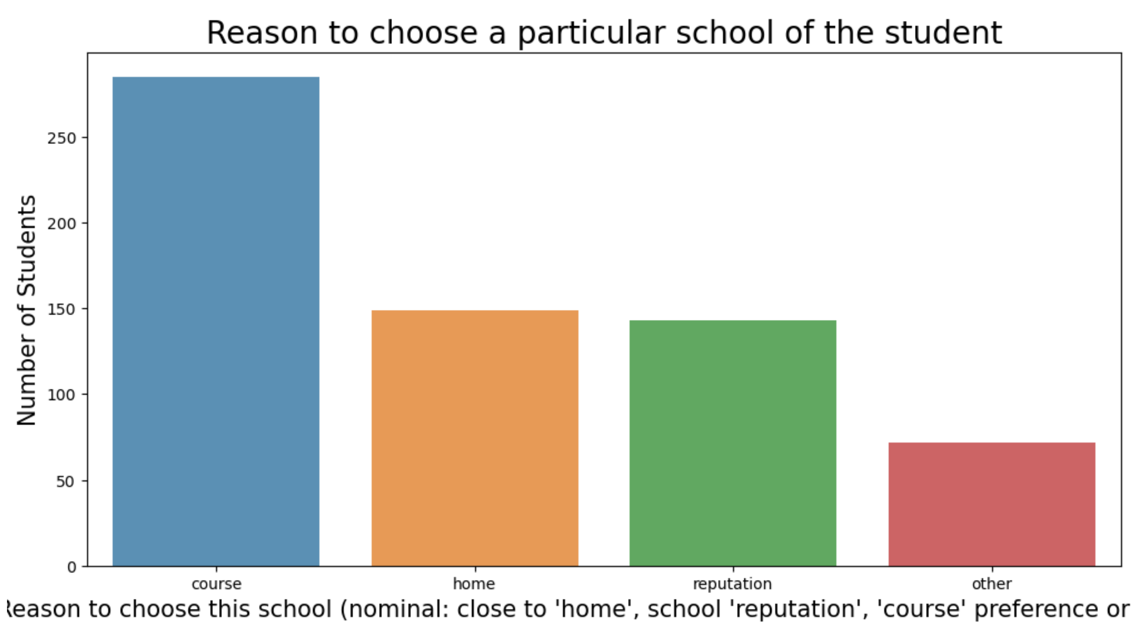
[30]:
```
count=df['Mjob'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Mother's job of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Mother's job (nominal: 'teacher', 'health' care related, civil␣
↪'services' (e.g. administrative or police), 'at_home' or 'other')",␣
↪fontsize=15)
plt.show()
```



[31]:
```
count=df['Fjob'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Father's job of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Father's job (nominal: 'teacher', 'health' care related, civil␣
↪'services' (e.g. administrative or police), 'at_home' or 'other')",␣
↪fontsize=15)
plt.show()
```

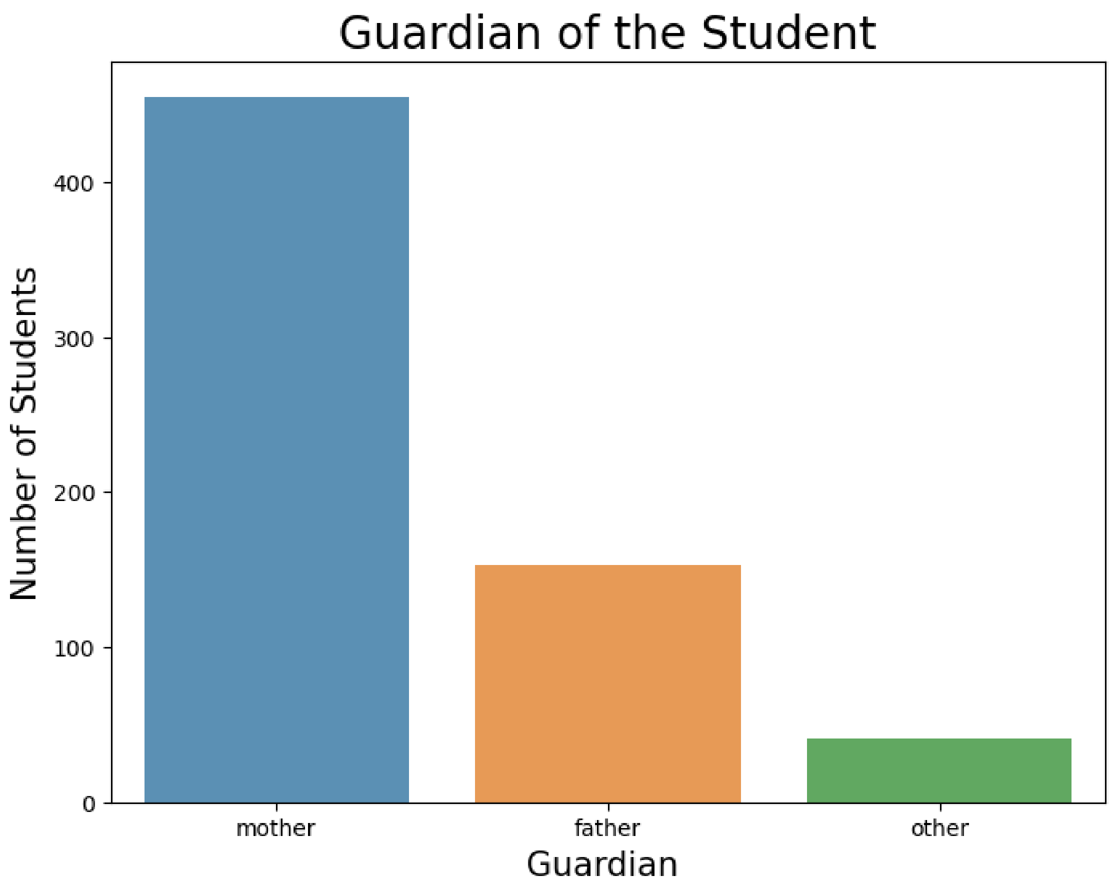


[32]:
```
count=df['reason'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Reason to choose a particular school of the student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Reason to choose this school (nominal: close to 'home', school␣
↪'reputation', 'course' preference or 'other'", fontsize=15)
plt.show()
```



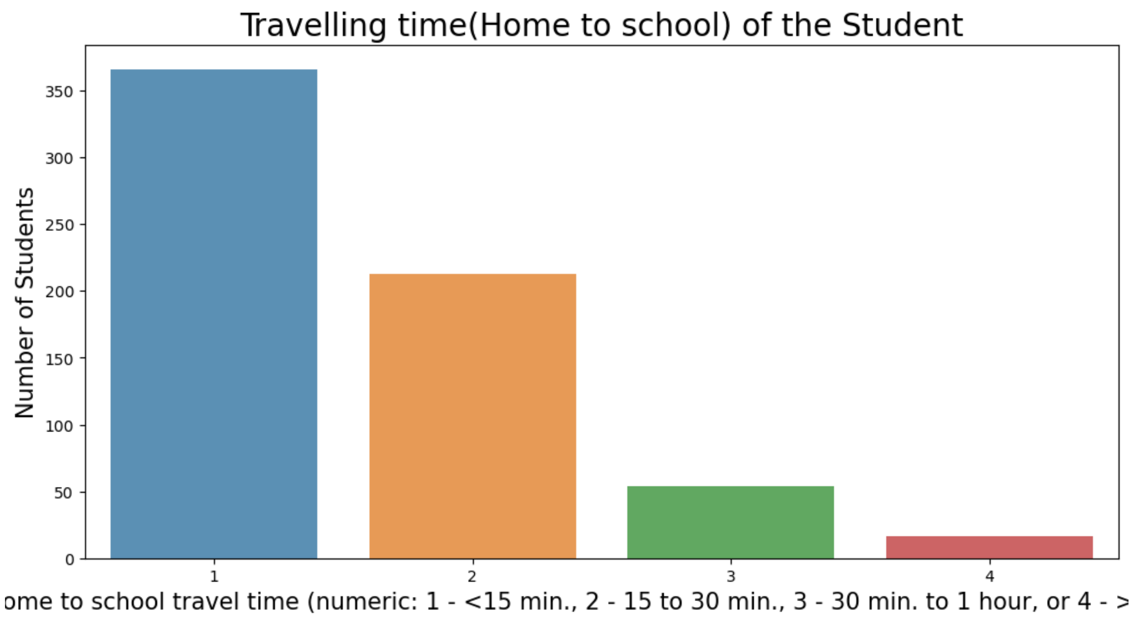
[33]:

```
count=df['guardian'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Guardian of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Guardian ", fontsize=15)
plt.show()
```



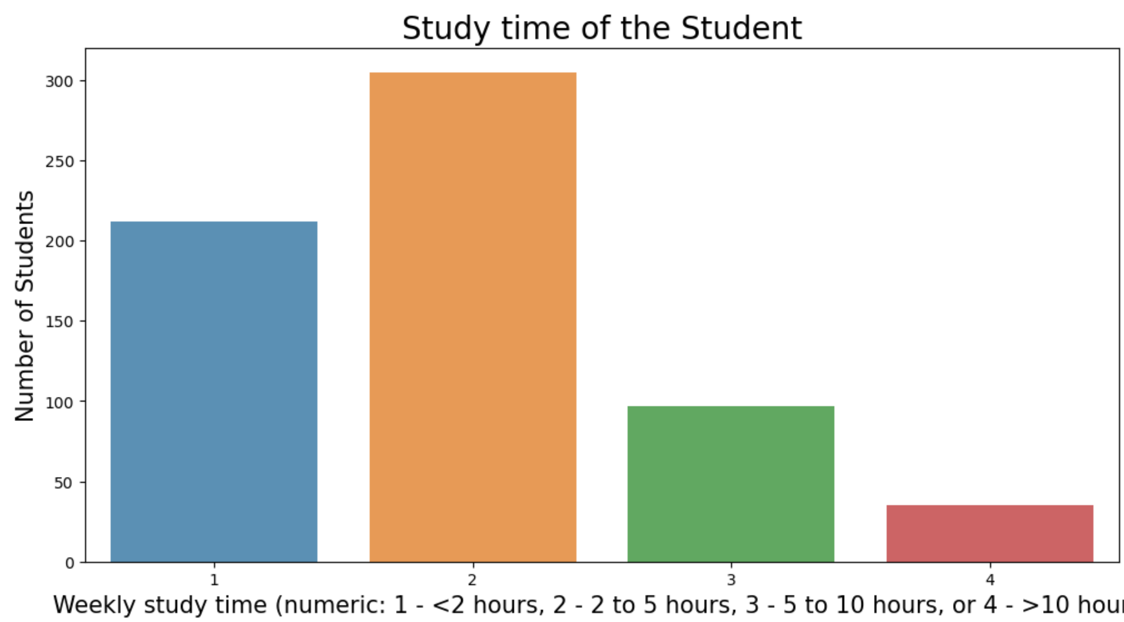
[34]:

```
count=df['traveltime'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Travelling time(Home to school) of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min.
 ↪, 3 - 30 min. to 1 hour, or 4 - >1 hour)", fontsize=15)
plt.show()
```

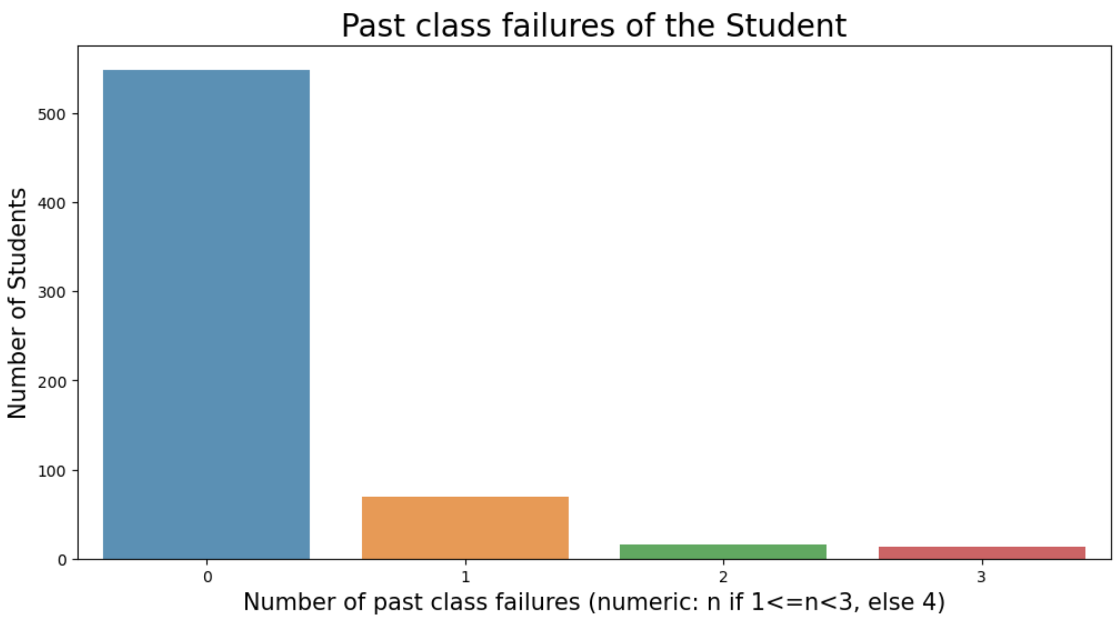


[35]:

```
count=df['studytime'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Study time of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5␣
↪to 10 hours, or 4 - >10 hours)", fontsize=15)
plt.show()
```

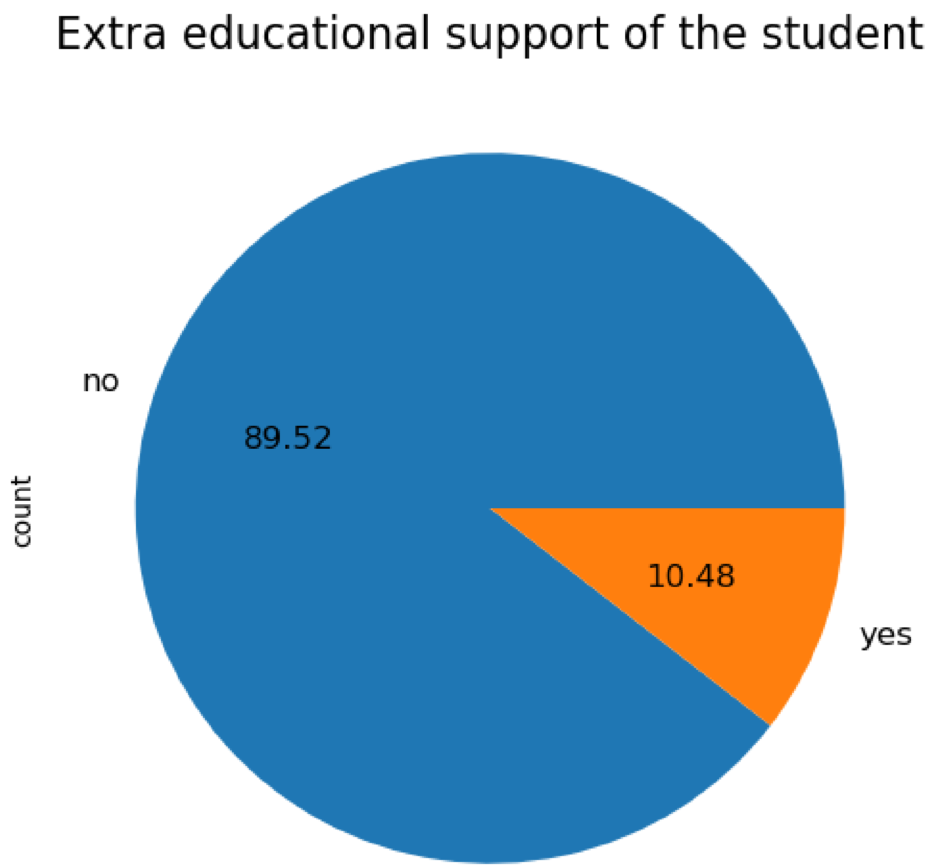


[36]:
```
count=df['failures'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Past class failures of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Number of past class failures (numeric: n if 1<=n<3, else 4)",␣
↪fontsize=15)
plt.show()
```



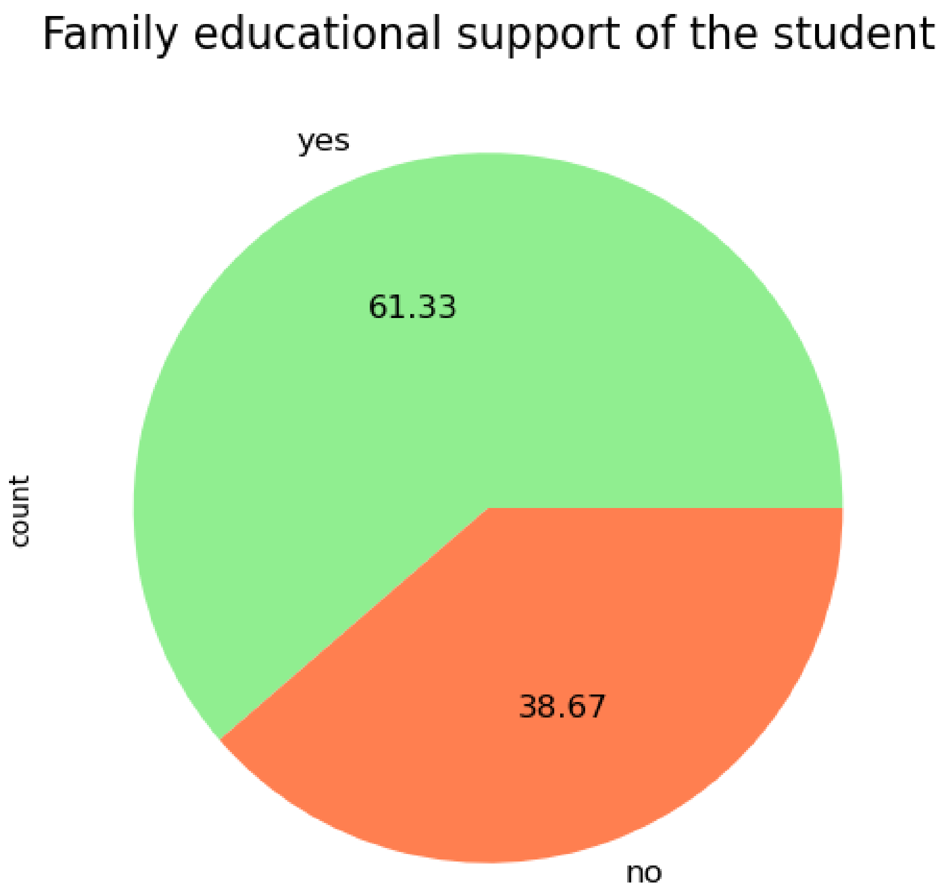
[37]:
```
df['schoolsup'].value_counts().plot.pie(autopct='%.
↪2f',figsize=(8,6),fontsize=12)
plt.title("Extra educational support of the student",fontsize=16)
```

[37]:
```
Text(0.5, 1.0, 'Extra educational support of the student')
```



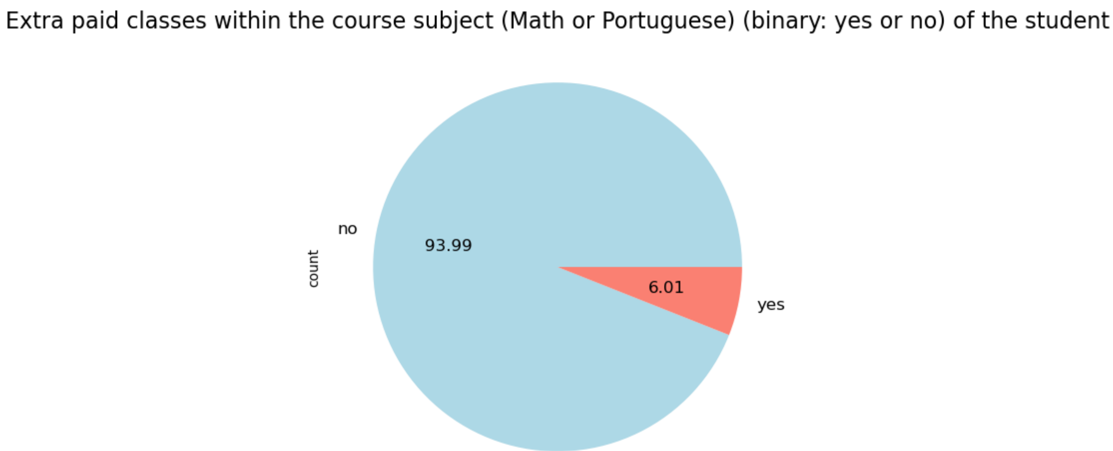

```
[38]: df['famsup'].value_counts().plot.pie(autopct='%.
       ↪2f',figsize=(8,6),fontsize=12,colors=['lightgreen','coral'])
      plt.title("Family educational support of the student",fontsize=16)
```

```
[38]: Text(0.5, 1.0, 'Family educational support of the student')
```



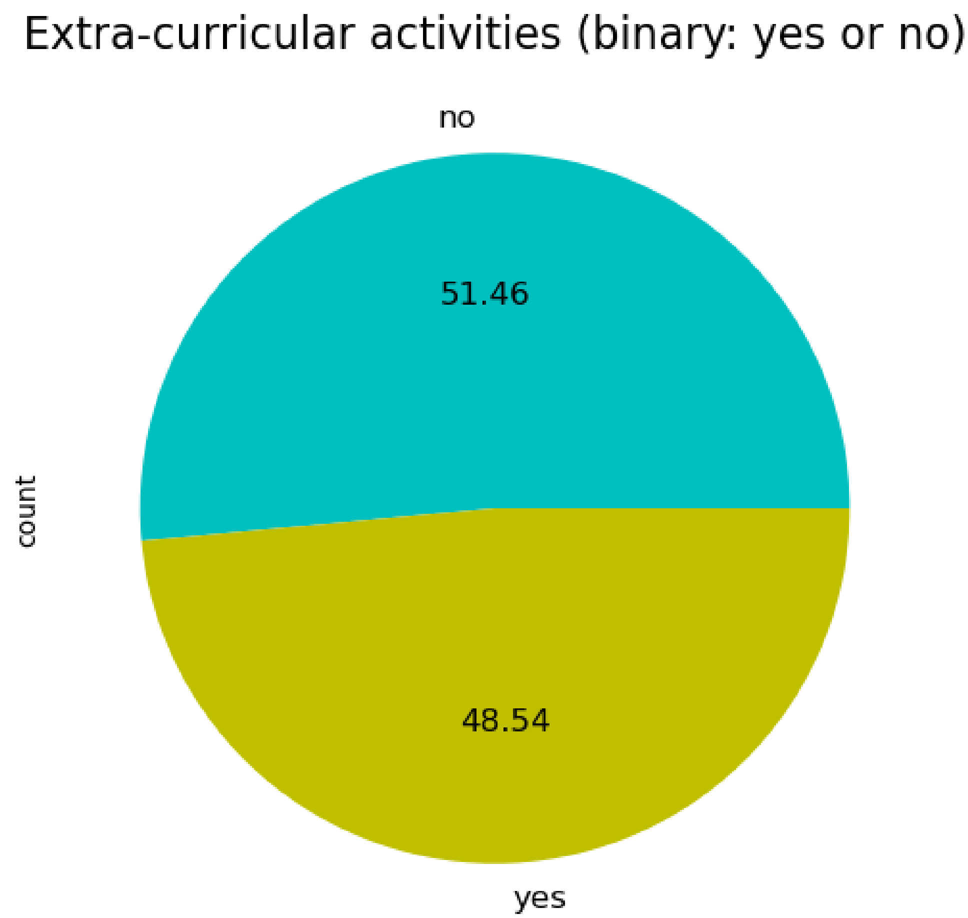
[39]:
```
df['paid'].value_counts().plot.pie(autopct='%.
 ↪2f',figsize=(8,6),fontsize=12,colors=['lightblue','salmon'])
plt.title("Extra paid classes within the course subject (Math or Portuguese)␣
 ↪(binary: yes or no) of the student",fontsize=16)
```

[39]:
```
Text(0.5, 1.0, 'Extra paid classes within the course subject (Math or
Portuguese) (binary: yes or no) of the student')
```



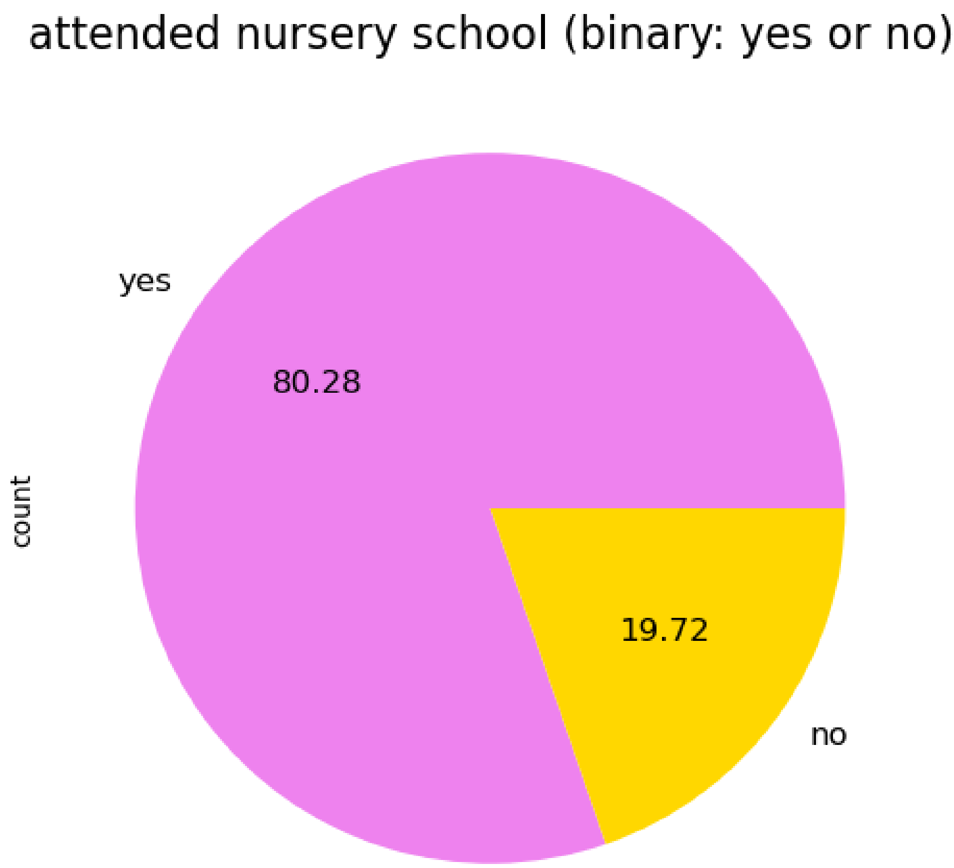
[40]:
```
df['activities'].value_counts().plot.pie(autopct='%.
↪2f',figsize=(8,6),fontsize=12,colors=['c','y'])
plt.title("Extra-curricular activities (binary: yes or no)",fontsize=16)
```

[40]:
```
Text(0.5, 1.0, 'Extra-curricular activities (binary: yes or no)')
```



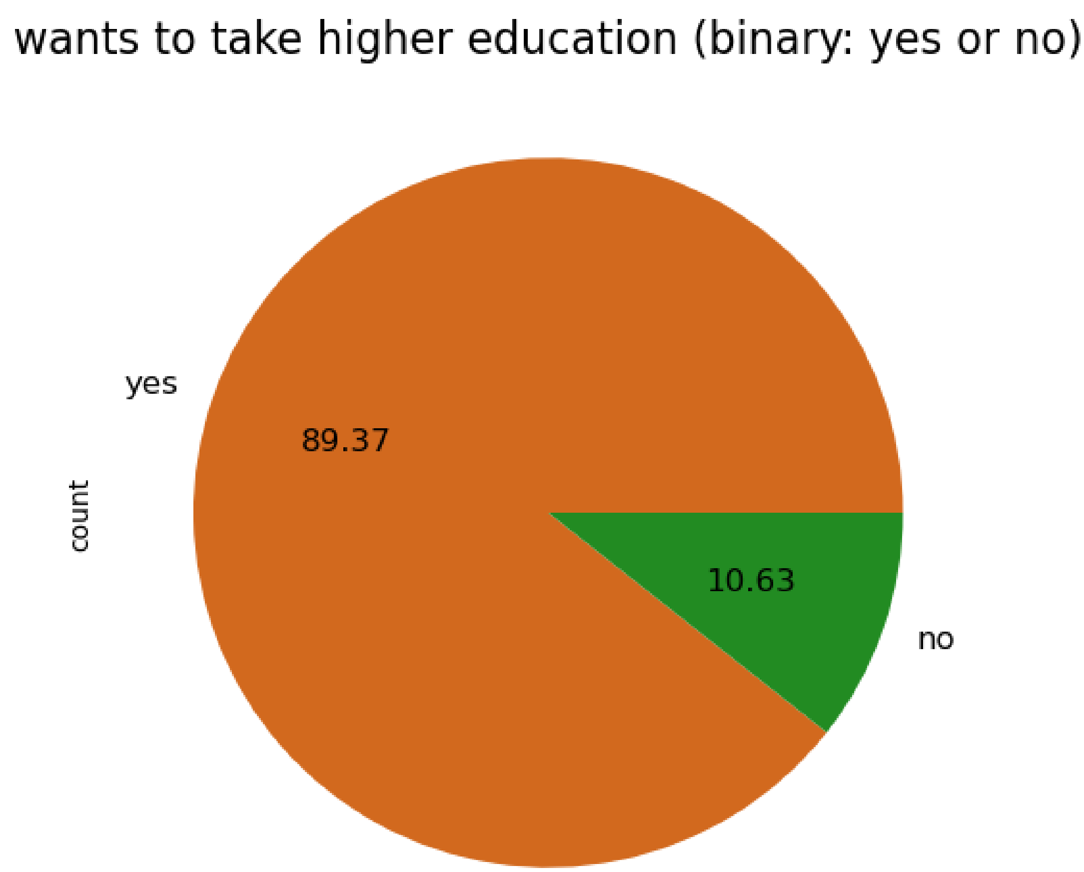
[41]:
```
df['nursery'].value_counts().plot.pie(autopct='%.
↪2f',figsize=(8,6),fontsize=12,colors=['violet','gold'])
plt.title("attended nursery school (binary: yes or no)",fontsize=16)
```

[41]:
```
Text(0.5, 1.0, 'attended nursery school (binary: yes or no)')
```



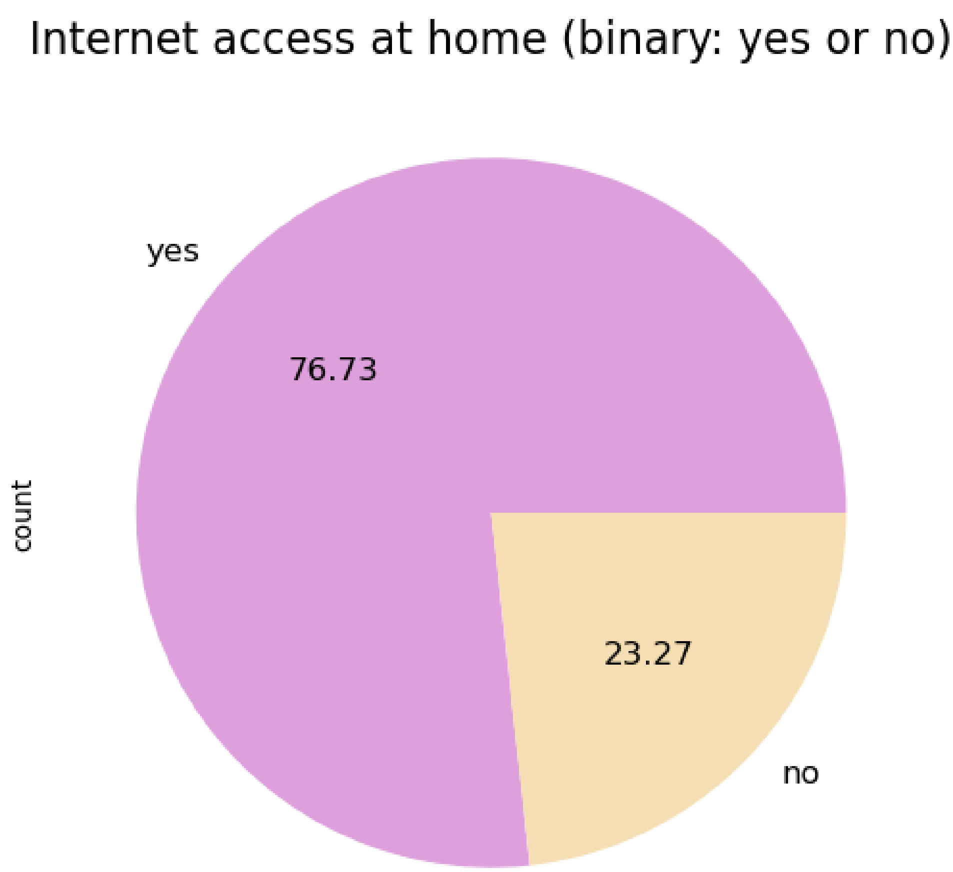

```
[42]: df['higher'].value_counts().plot.pie(autopct='%.
        ↪2f',figsize=(8,6),fontsize=12,colors=['chocolate','forestgreen'])
      plt.title("wants to take higher education (binary: yes or no)",fontsize=16)
```

```
[42]: Text(0.5, 1.0, 'wants to take higher education (binary: yes or no)')
```



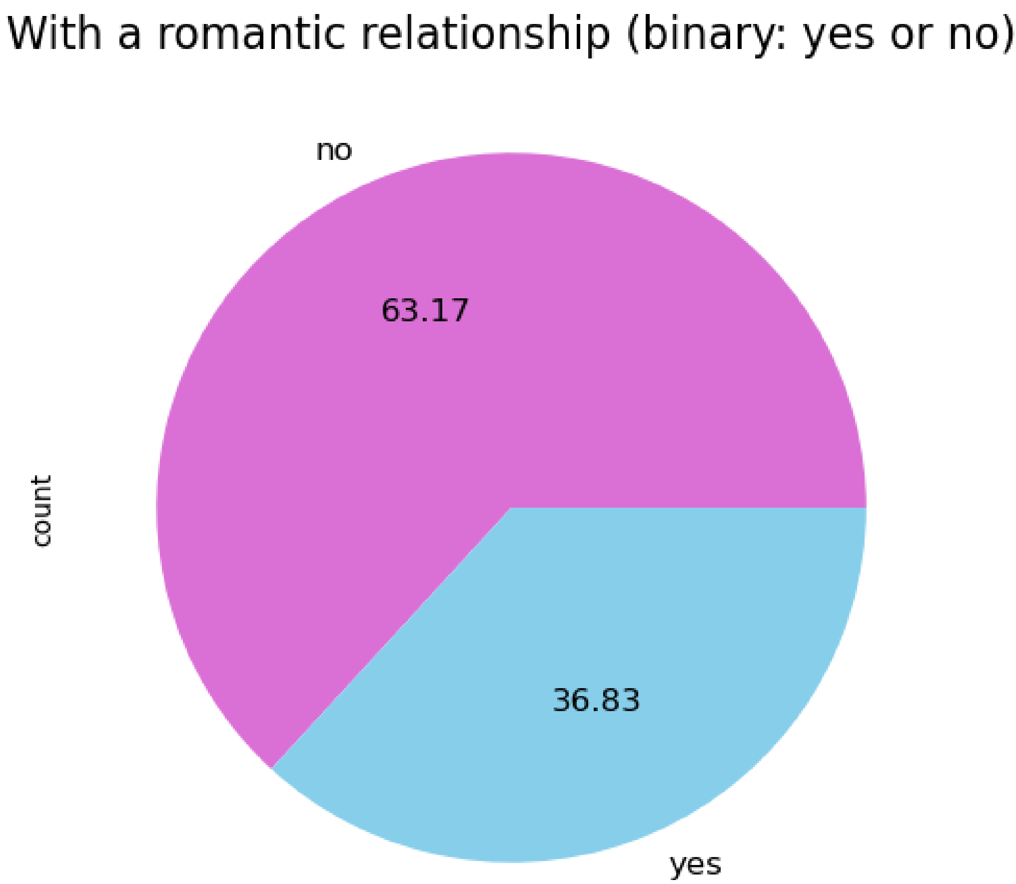
[43]:
```
df['internet'].value_counts().plot.pie(autopct='%.
↪2f',figsize=(8,6),fontsize=12,colors=['plum','wheat'])
plt.title("Internet access at home (binary: yes or no)",fontsize=16)
```

[43]:
```
Text(0.5, 1.0, 'Internet access at home (binary: yes or no)')
```

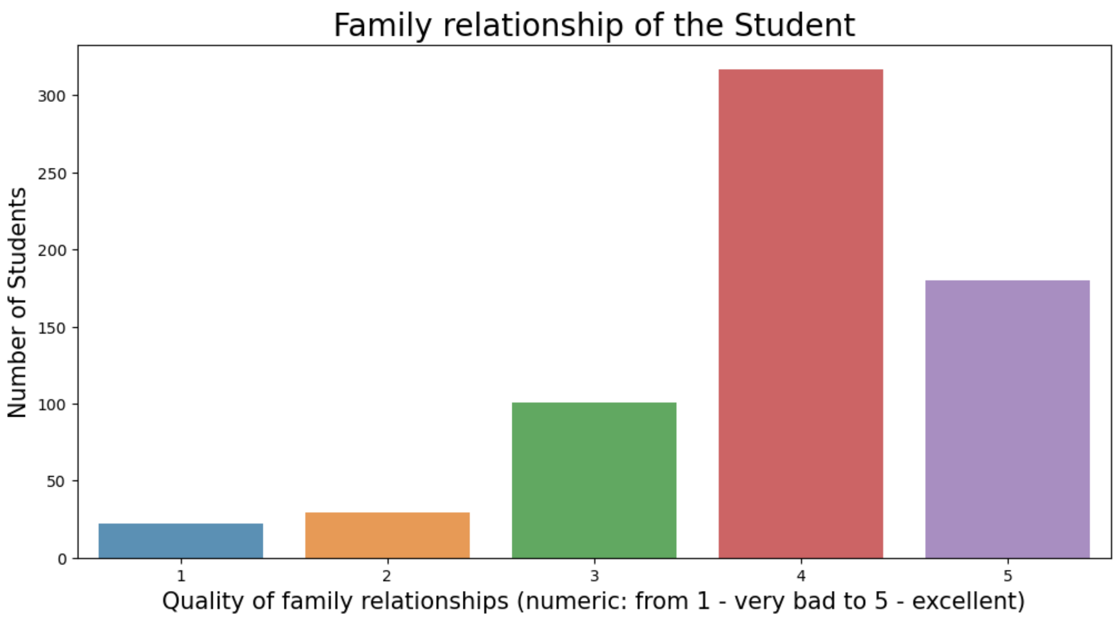



```
[44]: df['romantic'].value_counts().plot.pie(autopct='%.
      ↪2f',figsize=(8,6),fontsize=12,colors=['orchid','skyblue'])
      plt.title("With a romantic relationship (binary: yes or no)",fontsize=16)
```

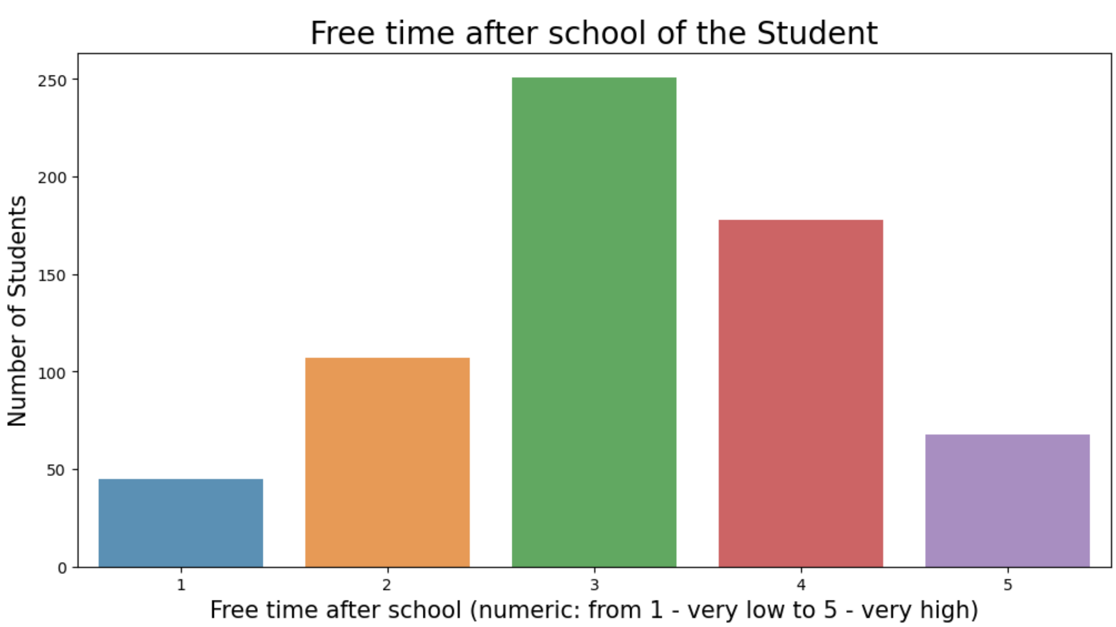
```
[44]: Text(0.5, 1.0, 'With a romantic relationship (binary: yes or no)')
```



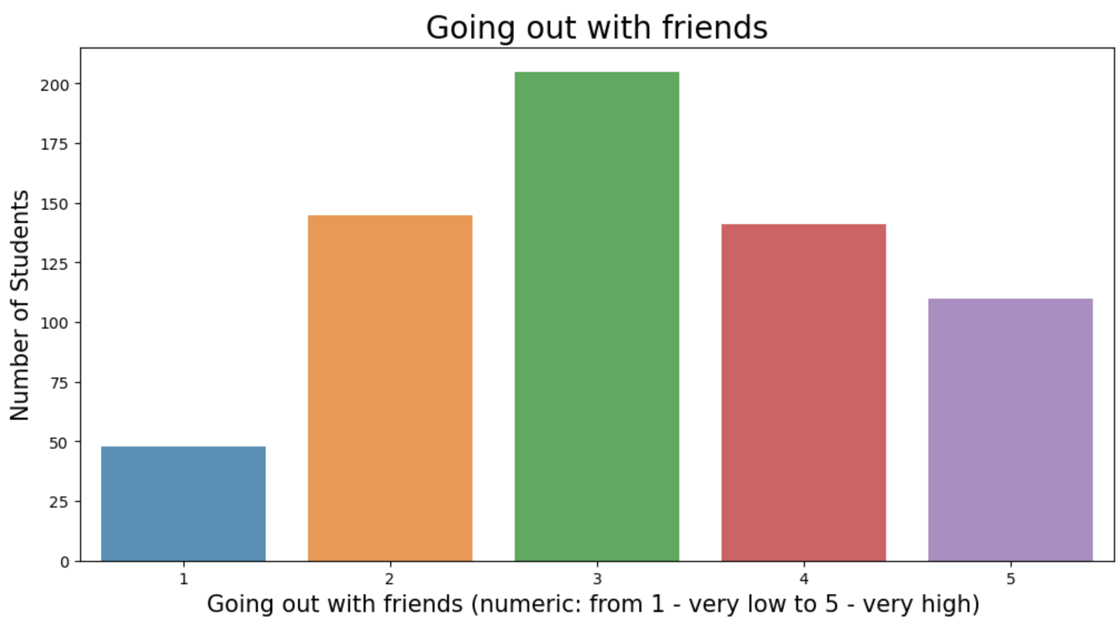
[45]:
```
count=df['famrel'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Family relationship of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Quality of family relationships (numeric: from 1 - very bad to 5 -␣
  ↪excellent)", fontsize=15)
plt.show()
```



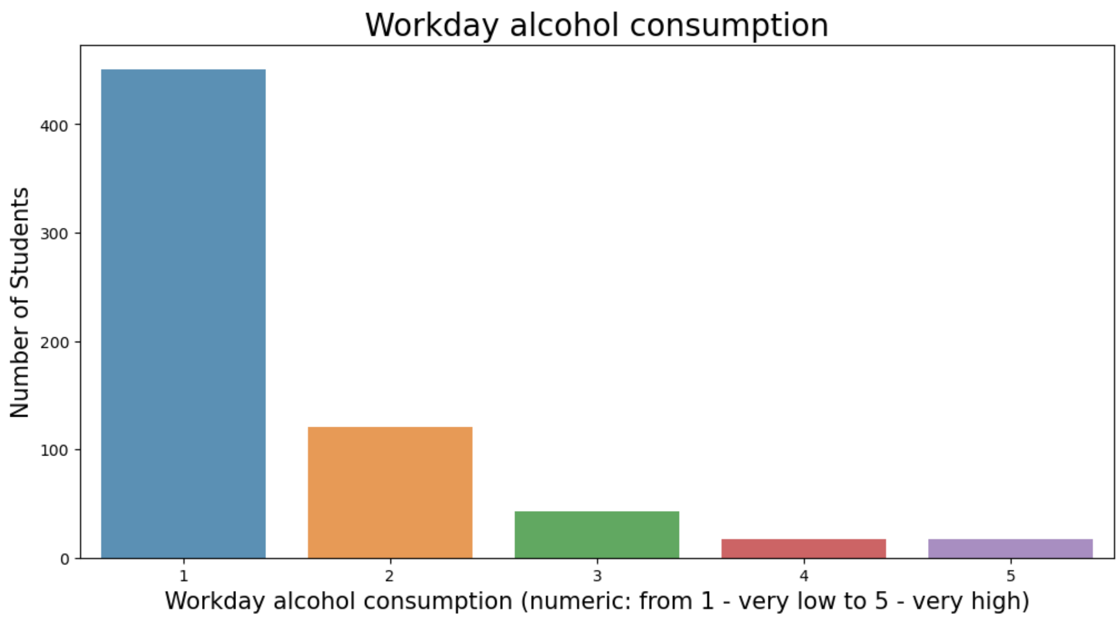
[46]:
```
count=df['freetime'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Free time after school of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Free time after school (numeric: from 1 - very low to 5 - very␣
↪high)", fontsize=15)
plt.show()
```



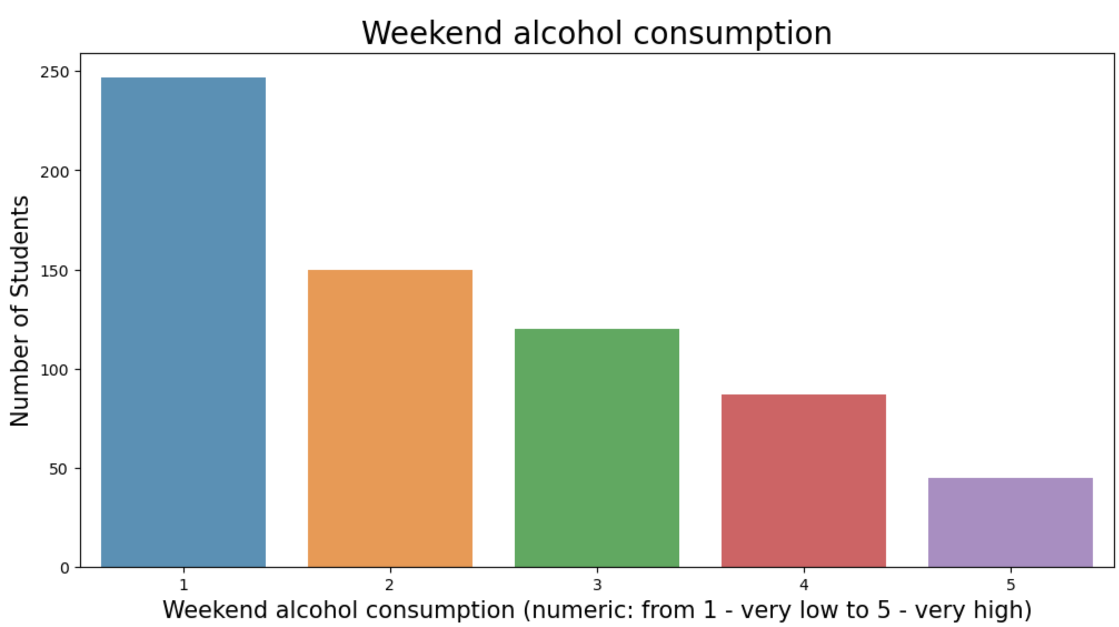
```
[47]: count=df['goout'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Going out with friends", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Going out with friends (numeric: from 1 - very low to 5 - very␣
 ↪high) ", fontsize=15)
plt.show()
```



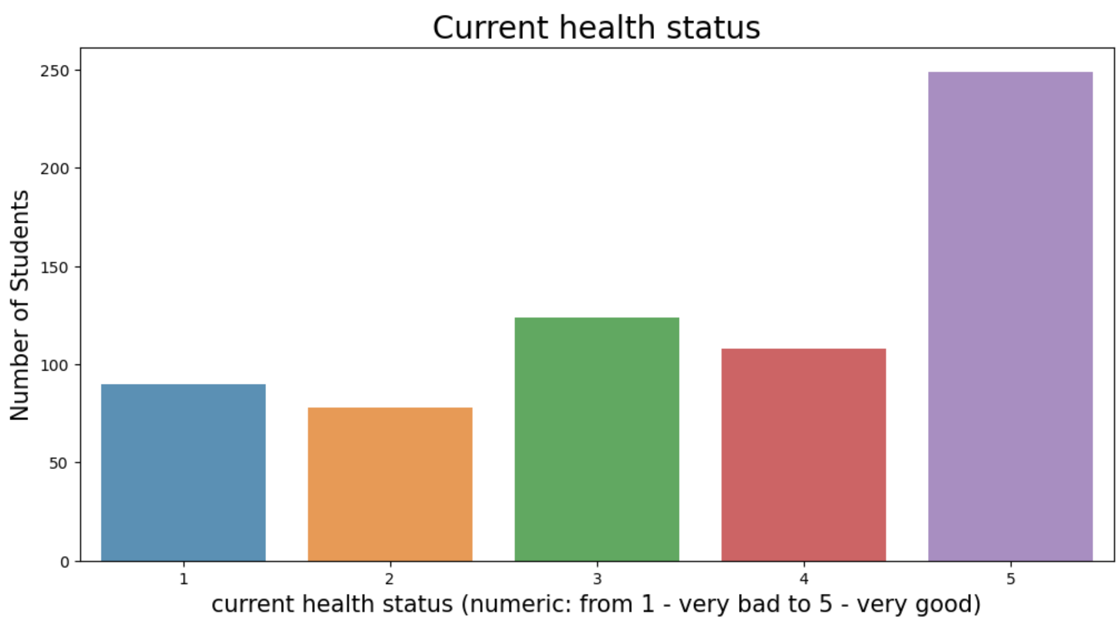

```
[48]: count=df['Dalc'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Workday alcohol consumption", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Workday alcohol consumption (numeric: from 1 - very low to 5 - very␣
 ↪high)", fontsize=15)
plt.show()
```



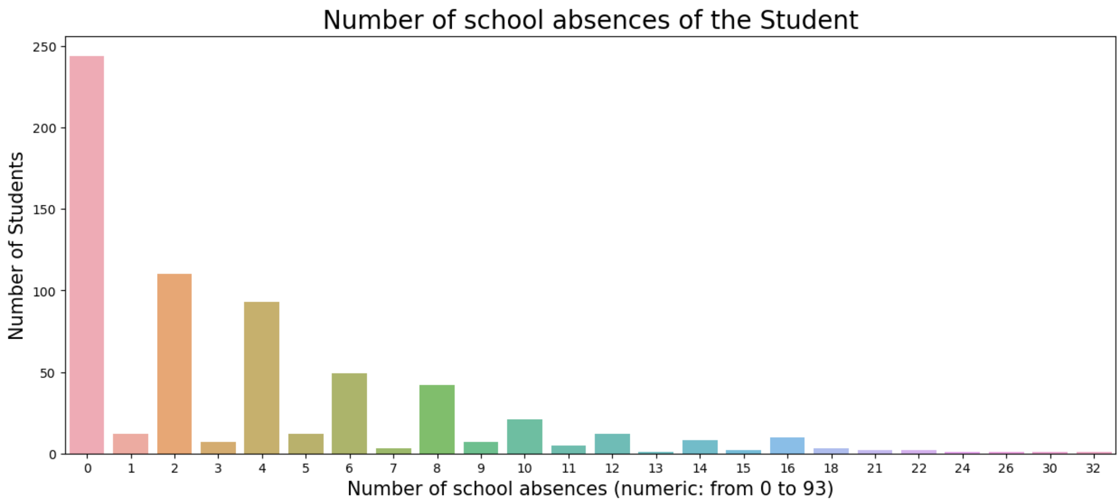
[49]:
```
count=df['Walc'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Weekend alcohol consumption", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Weekend alcohol consumption (numeric: from 1 - very low to 5 - very␣
↪high)", fontsize=15)
plt.show()
```



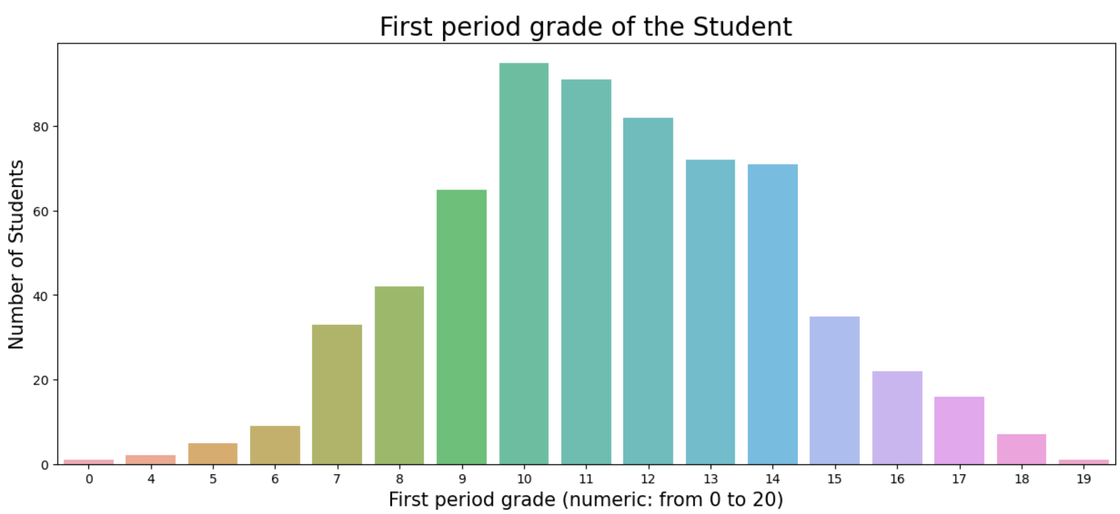
[50]:
```
count=df['health'].value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Current health status", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("current health status (numeric: from 1 - very bad to 5 - very␣
↪good)", fontsize=15)
plt.show()
```



[51]:
```
count=df['absences'].value_counts()
plt.figure(figsize=(15,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Number of school absences of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("Number of school absences (numeric: from 0 to 93)", fontsize=15)
plt.show()
```

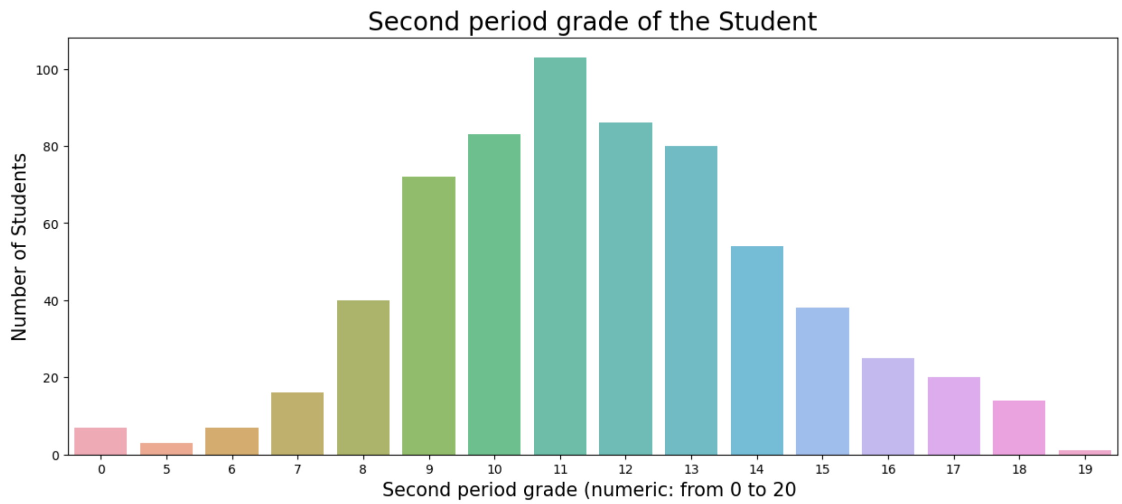


[52]:
```
count=df['G1'].value_counts()
plt.figure(figsize=(15,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("First period grade of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
plt.xlabel("First period grade (numeric: from 0 to 20)", fontsize=15)
plt.show()
```



[53]:
```
count=df['G2'].value_counts()
plt.figure(figsize=(15,6))
sns.barplot(x=count.index,y=count.values, alpha=0.8)
plt.title("Second period grade of the Student", fontsize=20)
plt.ylabel('Number of Students', fontsize=15)
```

```
plt.xlabel("Second period grade (numeric: from 0 to 20 ", fontsize=15)
plt.show()
```



1.3 Data training

[54]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64
 7   Fedu        649 non-null    int64
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64
 13  studytime   649 non-null    int64
 14  failures    649 non-null    int64
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
```

```
 18  activities  649 non-null   object
 19  nursery     649 non-null   object
 20  higher      649 non-null   object
 21  internet    649 non-null   object
 22  romantic    649 non-null   object
 23  famrel      649 non-null   int64
 24  freetime    649 non-null   int64
 25  goout       649 non-null   int64
 26  Dalc        649 non-null   int64
 27  Walc        649 non-null   int64
 28  health      649 non-null   int64
 29  absences    649 non-null   int64
 30  G1          649 non-null   int64
 31  G2          649 non-null   int64
 32  G3          649 non-null   int64
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

[55]:
```
prepareddata=df.drop(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus',␣
 ↪'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime','schoolsup',␣
 ↪'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'Dalc',
       'Walc'],axis=1)
prepareddata
```

[55]:
```
     studytime  failures  famrel  freetime  goout  health  absences  G1  G2  \
0            2         0       4         3      4       3         4   0  11
1            2         0       5         3      3       3         2   9  11
2            2         0       4         3      2       3         6  12  13
3            3         0       3         2      2       5         0  14  14
4            2         0       4         3      2       5         0  11  13
..         …         …       …        …      …       …         …  ..  ..
644          3         1       5         4      2       5         4  10  11
645          2         0       4         3      4       1         4  15  15
646          2         0       1         1      1       5         6  11  12
647          1         0       2         4      5       2         6  10  10
648          1         0       4         4      1       5         4  10  11

     G3
0    11
1    11
2    12
3    14
4    13
..   ..
644  10
```

```
645   16
646    9
647   10
648   11

[649 rows x 10 columns]
```

[58]:
```
prepareddata.columns
```

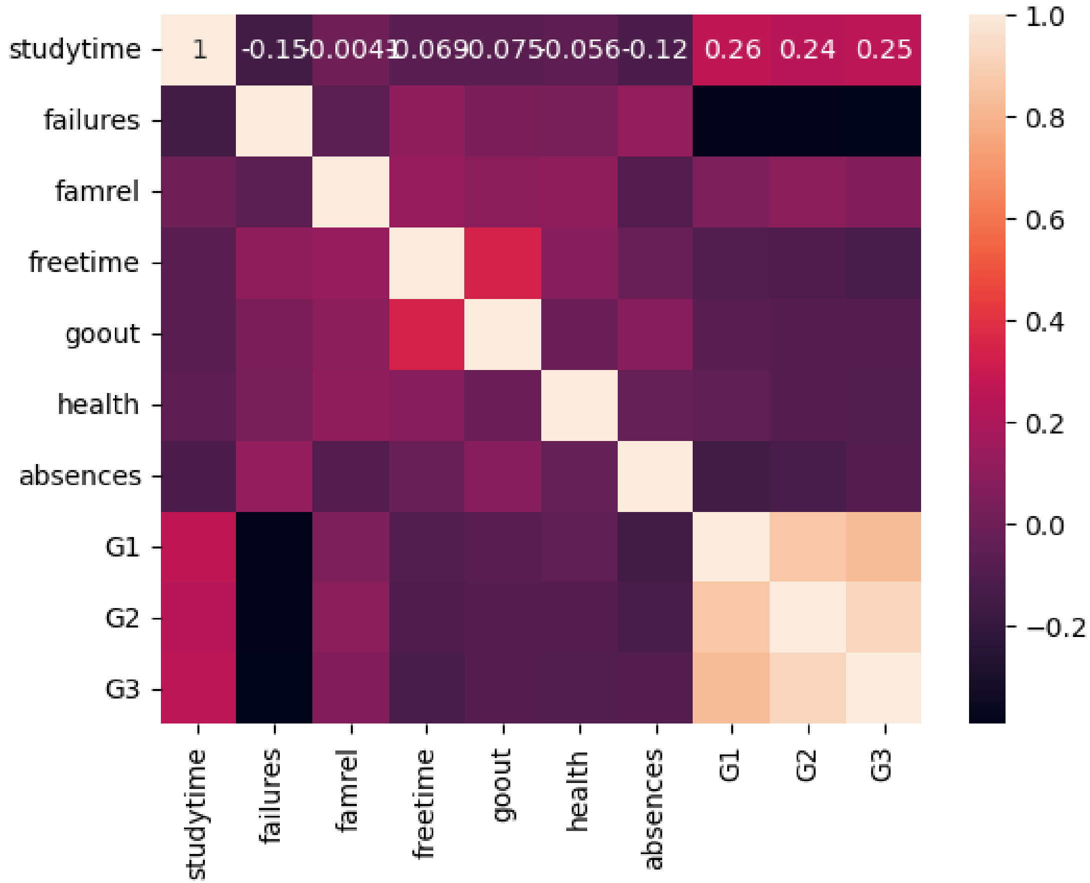
[58]:
```
Index(['studytime', 'failures', 'famrel', 'freetime', 'goout', 'health',
       'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

[59]:
```
prepareddata.describe()
```

[59]:
```
        studytime    failures      famrel    freetime       goout      health  \
count  649.000000  649.000000  649.000000  649.000000  649.000000  649.000000
mean     1.930663    0.221880    3.930663    3.180277    3.184900    3.536210
std      0.829510    0.593235    0.955717    1.051093    1.175766    1.446259
min      1.000000    0.000000    1.000000    1.000000    1.000000    1.000000
25%      1.000000    0.000000    4.000000    3.000000    2.000000    2.000000
50%      2.000000    0.000000    4.000000    3.000000    3.000000    4.000000
75%      2.000000    0.000000    5.000000    4.000000    4.000000    5.000000
max      4.000000    3.000000    5.000000    5.000000    5.000000    5.000000

         absences          G1          G2          G3
count  649.000000  649.000000  649.000000  649.000000
mean     3.659476   11.399076   11.570108   11.906009
std      4.640759    2.745265    2.913639    3.230656
min      0.000000    0.000000    0.000000    0.000000
25%      0.000000   10.000000   10.000000   10.000000
50%      2.000000   11.000000   11.000000   12.000000
75%      6.000000   13.000000   13.000000   14.000000
max     32.000000   19.000000   19.000000   19.000000
```

[60]:
```
corr = prepareddata.corr()
sns.heatmap(corr, annot=True)
```

[60]:
```
<Axes: >
```



[61]:
```
# Import train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split
# Here, X is the data which will have features that affect student's␣
↪performance and y will have our target G3i.e. final grade.
x=prepareddata[['studytime', 'failures', 'famrel', 'freetime', 'goout',␣
↪'health','absences', 'G1', 'G2']]
y=prepareddata['G3']
```

[62]:
```
# Split data into training data and testing data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
↪2,random_state=100)
#Ratio used for splitting training and testing data is 8:2 respectively
```

1.4 Model Creation

Linear Regression

[63]:
```
# Importing linear regression model
from sklearn.linear_model import LinearRegression
reg1 = LinearRegression()
```

[64]:
```
# Fitting data into the model.
reg1.fit(x_train, y_train)
```

[64]:
```
LinearRegression()
```

[65]:
```
# Making predictions
pred1 = reg1.predict(x_test)
```

[66]:
```
pred1
```

[66]:
```
array([ 5.87235294,  8.24733246, 18.43556003, 12.15775162, 12.19499069,
       12.25201614, 11.0576008 , 10.0228101 ,  0.29942162, 13.09502938,
        8.67100818, 17.11477321, 11.08549307, 15.04930522,  9.75161352,
       17.09938819, 11.36522308, 12.5731234 , 15.70622706, 15.35191265,
       12.82883321, 16.40206615, 11.39784336, 12.43826496,  9.41609947,
        8.02903748, 11.56774322, 16.96287392, 11.20935768, 11.03102494,
       13.08340762, 10.95002545, 15.00178358, 12.14632769, 11.20790142,
       10.14904698, 13.38459878, 11.60036395,  9.34121292,  8.22019063,
       14.56659911,  7.92479422, 11.70834767,  9.30857567, 10.75263676,
       11.03637517, 16.04058282, 12.31746341, 11.13818518, 18.37447505,
        9.96135993, 15.36801098,  7.908889  , 10.77182075, 11.33349434,
       15.72851747,  7.86836939, 10.30547261, 14.23652481, 11.23957972,
       12.24550484, 13.45812015, 18.5562363 , 10.65003663,  8.62766306,
        7.18210079,  8.32250228, 10.3251571 , 14.64973088, 11.08542604,
        9.58514625, 10.68350165, 10.42050837, 19.84970871, 11.70382209,
       12.37546005, 11.90179201,  9.09453135,  8.16940497, 11.36749753,
        9.00558104, 10.17479134, 12.50541526, 15.56782464, 15.92717194,
        7.85997799, 10.34593363, 12.64791236, 10.24522921, 11.44210122,
        9.25008693, 12.93676842,  7.87727081,  8.48444122,  8.96909559,
        9.16129426, 12.92074821,  8.44268094, 10.14418725,  9.55833729,
        7.56956835, 10.20514694, 11.24037833,  8.12665795, 10.10889216,
       10.75615731, 15.72275643,  9.40224785,  9.68250887,  9.68093705,
       12.45701824, 11.66300531, 13.05898612, 13.34339479, 15.77219285,
       15.13538728, 15.48828157, 14.91760339, 18.44649495, 14.89721082,
       11.70769632, 14.37288146, 14.57004418, 15.16778203, 11.7296011 ,
       10.95722264, 10.68882834, 11.95927448, 17.34647126, 15.35476269])
```

[67]:
```
print("Accuracy of the Linear Regression model comes to be: \n ")
print(reg1.score(x_train,y_train))
```

```
Accuracy of the Linear Regression model comes to be:

0.8493517628952961
```

Lasso Regression

[68]:
```
# Importing model
from sklearn.linear_model import Lasso
reg2 = Lasso()
```

[69]:
```
# Fitting data into the model.
reg2.fit(x_train, y_train)
```

[69]:
```
Lasso()
```

[70]:
```
# Making predictions
pred2 = reg2.predict(x_test)
```

[71]:
```
pred2
```

[71]:
```
array([ 6.14159442,  8.81947079, 17.76735226, 12.34671883, 12.34671883,
        12.28183815, 10.58309481, 10.51821413,  1.63521334, 13.26097118,
         9.40931972, 16.72333854, 11.30270511, 15.0245952 , 10.45333344,
        16.85309991, 11.43246648, 12.21695746, 15.08947589, 15.0245952 ,
        12.4764802 , 15.93884755, 11.49734716, 12.41159951,  9.66884246,
         8.62482874, 11.36758579, 16.00372824, 11.36758579, 11.36758579,
        13.1960905 , 11.36758579, 14.89483383, 12.28183815, 11.36758579,
        10.51821413, 13.32585187, 11.30270511,  9.73372314,  8.68970942,
        14.17522353,  8.62482874, 11.49734716,  9.53908109, 10.51821413,
        11.36758579, 15.7442055 , 12.34671883, 11.43246648, 17.76735226,
        10.45333344, 15.08947589,  7.90521844, 10.58309481, 11.36758579,
        15.08947589,  8.75459011, 10.51821413, 14.17522353, 11.36758579,
        12.28183815, 13.1960905 , 17.76735226, 10.51821413,  8.81947079,
         7.77545707,  9.47420041, 10.45333344, 14.17522353, 11.36758579,
         9.66884246, 10.58309481, 10.58309481, 18.68160461, 11.43246648,
        12.34671883, 11.43246648,  9.53908109,  8.68970942, 11.43246648,
         9.47420041, 10.32357207, 12.41159951, 15.0245952 , 15.0245952 ,
         8.62482874, 10.45333344, 12.4764802 , 10.58309481, 10.64797549,
         9.66884246, 12.41159951,  8.68970942,  9.53908109,  9.60396177,
         9.47420041, 13.1960905 ,  9.60396177, 10.58309481,  9.73372314,
         8.55994805, 10.51821413, 11.43246648,  8.62482874, 10.51821413,
        10.51821413, 15.0245952 ,  9.60396177,  9.73372314,  9.60396177,
        12.41159951, 11.49734716, 13.1960905 , 13.13120981, 15.08947589,
        15.0245952 , 15.08947589, 14.17522353, 17.70247157, 14.24010422,
        11.49734716, 14.11034285, 15.0245952 , 15.0245952 , 11.56222785,
        11.36758579, 10.51821413, 12.34671883, 16.85309991, 15.0245952 ])
```

[72]:
```
print("Accuracy of the Lasso Regression model comes to be: \n ")
print(reg2.score(x_train,y_train))
```

```
Accuracy of the Lasso Regression model comes to be:
```

0.8315785087125718

Decision Tree Regressor

[73]:
```
# Importing decision tree regressor
from sklearn.tree import DecisionTreeRegressor
reg3 = DecisionTreeRegressor()
```

[74]:
```
#Fitting data into the model.
reg3.fit(x_train, y_train)
```

[74]:
```
DecisionTreeRegressor()
```

[75]:
```
# Making predictions on Test data
pred3 = reg3.predict(x_test)
```

[76]:
```
pred3
```

[76]:
```
array([ 8. ,  9. , 18. , 12. , 13. , 13. , 11. , 11. ,  0. , 14. ,  9. ,
       17. , 10. , 15. , 11. , 18. , 14. , 12. , 17. , 16. , 13. , 17. ,
       11. , 12. ,  9. ,  8. , 12. , 17. , 12. , 11. , 13. , 11. , 15. ,
       12. , 11. , 10. , 14. , 11. , 10. ,  8. , 14. ,  7. , 11. ,  8. ,
       11. , 12. , 16. , 13. , 12. , 18. , 10. , 16. , 10. , 11. , 11. ,
       17. ,  7. , 11. , 15. , 11. , 13.5, 13. , 18. , 10. , 10. ,  8. ,
        8. , 11. , 14. , 14. , 10. , 10. , 11. , 18. , 12. , 14. , 12. ,
       10. ,  8. , 11.5, 10. , 11. , 14. , 16. , 16. ,  7. , 10. , 14. ,
       11. , 11. ,  9. , 12. ,  7. ,  8. , 11. ,  9. , 13. , 10. , 10. ,
        9. ,  7. , 11. , 11. ,  9. , 10. , 11. , 16. , 11. , 10. , 10. ,
       12. , 12. , 13. , 12. , 16. , 16. , 16. , 15. , 17. , 17. , 11. ,
       16. , 16. , 15. , 12. , 11. , 10. , 13. , 18. , 16. ])
```

[77]:
```
print("Accuracy of the Decision Tree Regressor  model comes to be: \n ")
print(reg3.score(x_train,y_train))
```

```
Accuracy of the Decision Tree Regressor  model comes to be:

0.9994471375430892
```

Random Forest Regressor

[78]:
```
#Importing random forest regressor
from sklearn.ensemble import RandomForestRegressor
reg4 = RandomForestRegressor(n_estimators=100)
```

[79]:
```
# Fitting data into the model.
reg4.fit(x_train, y_train)
```

[79]:
```
RandomForestRegressor()
```

[80]:
```
#making predictions.
pred4 = reg4.predict(x_test)
```

[81]:
```
pred4
```

[81]:
```
array([ 6.67      ,  8.43      , 17.6      , 12.61      , 12.73      ,
       12.55333333, 10.38      , 11.3      ,  0.84      , 13.06666667,
        9.25      , 16.74      , 10.74      , 15.45      , 10.11      ,
       17.66      , 11.48      , 12.77333333, 15.58      , 15.35      ,
       12.92      , 16.08      , 10.98      , 12.91      ,  9.23      ,
        8.19      , 11.63      , 15.86      , 11.91      , 11.54      ,
       12.78333333, 11.3      , 15.27      , 11.98      , 11.39      ,
       10.13      , 13.38      , 11.3      ,  9.97      ,  8.2       ,
       14.82      ,  8.02      , 11.3485    ,  9.21      , 10.78      ,
       12.17      , 15.56      , 12.79      , 12.02      , 17.77      ,
       10.08      , 16.04      ,  7.83      , 10.88      , 11.79      ,
       16.22      ,  6.56      , 10.46      , 14.84      , 11.5       ,
       12.40333333, 12.95      , 17.93      , 10.46      ,  7.44      ,
        7.79      ,  8.68      , 10.36      , 14.68      , 12.234     ,
        9.8       , 10.52      , 10.75      , 18.15      , 11.47166667,
       12.33      , 11.54      ,  9.66      ,  8.59      , 11.51333333,
        9.46      ,  9.64      , 12.92      , 15.58      , 15.75      ,
        7.82      ,  9.54      , 12.6       , 10.97      , 10.94      ,
        9.57      , 12.65      ,  7.91      ,  9.11      ,  9.83      ,
        9.18      , 13.52      ,  8.71      , 10.15      ,  9.61      ,
        7.68      , 10.39      , 11.1      ,  8.18      , 10.38      ,
       10.73      , 15.84      ,  9.96      , 10.39      , 10.14      ,
       13.02      , 11.4475    , 12.83      , 13.31      , 16.1       ,
       16.27      , 15.71      , 14.81      , 17.36      , 15.25      ,
       11.60333333, 14.73      , 15.47      , 15.47      , 11.655     ,
       11.245     ,  9.94      , 12.95      , 17.47      , 15.73      ])
```

[82]:
```
print("Accuracy of the Random Forest Regressor  model comes to be: \n ")
print(reg4.score(x_train,y_train))
```

```
Accuracy of the Random Forest Regressor  model comes to be:

0.9750680076898599
```

1.5 Performance Check

[83]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
print("Model\t\t\t          RootMeanSquareError  \t\t Accuracy of the model")
print("""Linear Regression       \t\t {:.4f} \t \t\t {:.4f}""".format(  np.
 ↪sqrt(mean_squared_error(y_test, pred1)), reg1.score(x_train,y_train)))
```

```
print("""Lasso Regression        \t\t {:.4f} \t \t\t {:.4f}""".format(  np.
 ↪sqrt(mean_squared_error(y_test, pred2)), reg2.score(x_train,y_train)))
print("""Decision Tree Regressor \t\t {:.4f} \t \t\t {:.4f}""".format(  np.
 ↪sqrt(mean_squared_error(y_test, pred3)), reg3.score(x_train,y_train)))
print("""Random Forest Regressor \t\t {:.4f} \t \t\t {:.4f}""".format(  np.
 ↪sqrt(mean_squared_error(y_test, pred4)), reg4.score(x_train,y_train)))
```

```
Model                              RootMeanSquareError            Accuracy of the
model
Linear Regression                         1.2218                         0.8494
Lasso Regression                          1.2391                         0.8316
Decision Tree Regressor                   1.4767                         0.9994
Random Forest Regressor                   1.2785                         0.9751
```

1.6 Conclusion

Accuracy of Decision Tree Regressor is higher than Linear Regression, Lasso Regression and Random Forest Regressor.

[ ]:
```

```