2023aiml554-assignment

July 14, 2024

1 Problem Statement:-

This dataset contains comprehensive information on 2,392 high school students, detailing their demographics, study habits, parental involvement, extracurricular activities, and academic performance. The target variable, GradeClass, classifies students' grades into distinct categories, providing a robust dataset for educational research, predictive modeling, and statistical analysis.

1.1 Libraries Import

[1]:
```
#Importing required packages
import numpy as np
import pandas as pd
import warnings as war
war.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sbn
import math
```

1.2 Load & Read dataSet

[2]:
```
# Defining dataSet csv file path
filepath="C:\\Users\\ASUS\\jupyterworkspace\\Assignment & Mini␣
 ↪Project\\Module_02_Feature Engineering\\Assignment\\Student_performance_data.
 ↪csv"
#Loading dataSet
dataRead=pd.read_csv(filepath)
#printing dataSet
dataRead
```

[2]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	\
0	1001	17.0	1	0	2	19.833723	
1	1002	18.0	0	0	1	15.408756	
2	1003	15.0	0	2	3	4.210570	
3	1004	17.0	1	0	3	10.028829	
4	1005	17.0	1	0	2	4.672495	
…	…	…	…	…	…	…	
2387	3388	18.0	1	0	3	10.680555	

```
2388       3389  17.0       0          0                  1         7.583217
2389       3390  16.0       1          0                  2         6.805500
2390       3391  16.0       1          1                  0        12.416653
2391       3392  16.0       1          0                  2        17.819907

      Absences  Tutoring  ParentalSupport  Extracurricular  Sports  Music  \
0          7.0       1.0              2.0                0     0.0    1.0
1          0.0       0.0              1.0                0     0.0    0.0
2         26.0       0.0              2.0                0     0.0    0.0
3         14.0       0.0              3.0                1     0.0    0.0
4         17.0       1.0              3.0                0     0.0    0.0
…          …         …                …                  …     …      …
2387       2.0       0.0              4.0                1     0.0    0.0
2388       4.0       1.0              4.0                0     1.0    0.0
2389      20.0       0.0              2.0                0     0.0    0.0
2390      17.0       0.0              2.0                0     1.0    1.0
2391      13.0       0.0              2.0                0     0.0    0.0

      Volunteering       GPA  GradeClass
0                0  2.929196           2
1                0  3.042915           1
2                0  0.112602           4
3                0  2.054218           3
4                0  1.288061           4
…              …       …          …
2387             0  3.455509           0
2388             0  3.279150           4
2389             1  1.142333           2
2390             0  1.803297           1
2391             1  2.140014           1

[2392 rows x 15 columns]
```

1.3 Task 1 : Identify the type of each of the attributes in the dataset.

1.3.1 DataTypes of each features/attributes

[3]:
```
# Checking descriptive statistic of the dataSet
dataRead.describe()
```

[3]:
```
         StudentID          Age       Gender    Ethnicity  ParentalEducation  \
count  2392.000000  2375.000000  2392.000000  2392.000000        2392.000000
mean   2196.500000    16.468632     0.510870     0.877508           1.746237
std     690.655244     1.124404     0.499986     1.028476           1.000411
min    1001.000000    15.000000     0.000000     0.000000           0.000000
25%    1598.750000    15.000000     0.000000     0.000000           1.000000
50%    2196.500000    16.000000     1.000000     0.000000           2.000000
```

```
75%    2794.250000    17.000000     1.000000     2.000000           2.000000
max    3392.000000    18.000000     1.000000     3.000000           4.000000

       StudyTimeWeekly     Absences     Tutoring  ParentalSupport  \
count      2358.000000  2380.000000  2367.000000      2378.000000
mean          9.761567    14.546639     0.301225         2.122372
std           5.658241     8.463651     0.458887         1.122941
min           0.001057     0.000000     0.000000         0.000000
25%           5.030526     7.000000     0.000000         1.000000
50%           9.653180    15.000000     0.000000         2.000000
75%          14.408107    22.000000     1.000000         3.000000
max          19.978094    29.000000     1.000000         4.000000

       Extracurricular       Sports        Music  Volunteering          GPA  \
count      2392.000000  2381.000000  2180.000000   2392.000000  2392.000000
mean          0.383361     0.303234     0.197248      0.157191     1.906186
std           0.486307     0.459752     0.398012      0.364057     0.915156
min           0.000000     0.000000     0.000000      0.000000     0.000000
25%           0.000000     0.000000     0.000000      0.000000     1.174803
50%           0.000000     0.000000     0.000000      0.000000     1.893393
75%           1.000000     1.000000     0.000000      0.000000     2.622216
max           1.000000     1.000000     1.000000      1.000000     4.000000

        GradeClass
count  2392.000000
mean      2.983696
std       1.233908
min       0.000000
25%       2.000000
50%       4.000000
75%       4.000000
max       4.000000
```

[4]:
```
#checking datatypes of each features/attributes for the dataSet
dataRead.dtypes
```

[4]:
```
StudentID              int64
Age                  float64
Gender                 int64
Ethnicity              int64
ParentalEducation      int64
StudyTimeWeekly      float64
Absences             float64
Tutoring             float64
ParentalSupport      float64
Extracurricular        int64
Sports               float64
```

```
Music                float64
Volunteering           int64
GPA                  float64
GradeClass             int64
dtype: object
```

1.3.2 Attributes/Features types

1.3.3 Below mentioning identification of each attributes/features from dataSet

StudentID : Nominal Discrete
Age : Ratio Continuous
Gender : Nominal Discrete
Ethinicity : Nominal Discrete
ParentalEducation : Ordinal Discrete
StudyTimeWeekly : Ratio Continuous
Absences : Interval Discrete
Tutoring : Nominal Discrete
ParentalSupport : Ordinal Discrete
Extracurricular : Nominal Discrete
Sports : Nominal Discrete
Music : Nominal Discrete
Volunteering : Nominal Discrete
GPA : Ratio Continuous
GradeClass : Ordinal Discrete

1.4 Task 2: Name the attribute(s) which have missing data. How many data points are missing? How will you handle missing data

1.4.1 Attributes name of missing data

[5]:
```
#Finding total no. of missing values for attributes specific
missingValue_Count=dataRead.isnull().sum()
print(missingValue_Count)
```

```
StudentID              0
Age                   17
Gender                 0
Ethnicity              0
ParentalEducation      0
StudyTimeWeekly       34
Absences              12
Tutoring              25
ParentalSupport       14
Extracurricular        0
Sports                11
Music                212
Volunteering           0
```

```
GPA                    0
GradeClass             0
dtype: int64
```

[6]:
```
#finding missing values attribues with counts
missingValue_attributes=missingValue_Count[missingValue_Count.
  ↪where(missingValue_Count.values>0).notnull()]
print(missingValue_attributes)
#Finding the attributes's key which have missing values
print("Below is the list of  missing values attributes:- ")
print(missingValue_attributes.keys())
```

```
Age                  17
StudyTimeWeekly      34
Absences             12
Tutoring             25
ParentalSupport      14
Sports               11
Music               212
dtype: int64
Below is the list of  missing values attributes:-
Index(['Age', 'StudyTimeWeekly', 'Absences', 'Tutoring', 'ParentalSupport',
       'Sports', 'Music'],
      dtype='object')
```

[7]:
```
#Finding total no. of missing values for dataSet
total_missingvalue_Count=missingValue_Count.sum()
print("total no. of missing values is :- {} ".format(total_missingvalue_Count))
```

```
total no. of missing values is :- 325
```

1.4.2 Logic to handle missing values

Handling & imputing missing values for "Age" attribute

[8]:
```
#calculating mean of "Age" attribute
print("Age mean :{}".format(math.floor(dataRead.Age.mean())))
#calculating median of "Age" attribute
print("Age median :{}".format(math.floor(dataRead.Age.median())))
#calculating mode of "Age" attribute
print("Age mode :{}".format(math.floor(dataRead.Age.mode()[0])))
#calculating value counts of "Age" attribute
print("Age value counts:{}".format(dataRead.Age.value_counts()))
```

```
Age mean :16
Age median :16
Age mode :15
Age value counts:Age
15.0    626
```

```
16.0    589
17.0    581
18.0    579
Name: count, dtype: int64
```

Observation:- "Age" is a numercial variable. Mean and median are almost same and value counts are also almost same, we can fill missing values with mean

[9]:
```
# Imputing missing values by mean for "Age" attribute
age_mean = round(dataRead.Age.mean(),0)
dataRead.Age.fillna(age_mean, inplace=True)
```

Handling & imputing missing values for "StudyTimeWeekly" attribute

[10]:
```
#calculating mean of "StudyTimeWeekly" attribute
print("StudyTimeWeekly mean :{}".format(dataRead.StudyTimeWeekly.mean()))
#calculating median of "StudyTimeWeekly" attribute
print("StudyTimeWeekly median :{}".format(dataRead.StudyTimeWeekly.median()))
#calculating mode of "StudyTimeWeekly" attribute
print("StudyTimeWeekly mode :{}".format(dataRead.StudyTimeWeekly.mode()[0]))
#calculating value counts of "StudyTimeWeekly" attribute
print("StudyTimeWeekly value counts:{}".format(dataRead.StudyTimeWeekly.
 ↪value_counts()))
```

```
StudyTimeWeekly mean :9.76156738078414
StudyTimeWeekly median :9.653180382
StudyTimeWeekly mode :0.001056539
StudyTimeWeekly value counts:StudyTimeWeekly
19.833723    1
8.745633     1
15.457479    1
6.779399     1
2.861628     1
            ..
4.190384     1
9.106575     1
18.168751    1
2.089606     1
17.819907    1
Name: count, Length: 2358, dtype: int64
```

Observation:- "StudyTimeWeekly" is a numercial variable. Mean and median are almost same and value counts are unique, we can fill missing values with mean

[11]:
```
# Imputing missing values by mean for "StudyTimeWeekly" attribute
StudyTimeWeekly_mean = dataRead['StudyTimeWeekly'].mean()
dataRead['StudyTimeWeekly'].fillna(StudyTimeWeekly_mean, inplace=True)
```

Handling & imputing missing values for "Absences" attribute

[12]:
```
#calculating mean of "Absences" attribute
print("Absences mean :{}".format(dataRead.Absences.mean()))
#calculating median of "Absences" attribute
print("Absences median :{}".format(dataRead.Absences.median()))
#calculating mode of "Absences" attribute
print("Absences mode :{}".format(dataRead.Absences.mode()[0]))
#calculating value counts of "Absences" attribute
print("Absences value counts:{}".format(dataRead.Absences.value_counts()))
```

```
Absences mean :14.546638655462186
Absences median :15.0
Absences mode :13.0
Absences value counts:Absences
13.0    93
12.0    92
25.0    91
19.0    90
8.0     90
20.0    88
15.0    87
6.0     87
23.0    86
7.0     86
1.0     84
21.0    83
14.0    83
27.0    83
5.0     82
17.0    81
24.0    81
26.0    80
3.0     79
16.0    79
18.0    79
11.0    71
22.0    71
29.0    69
4.0     69
2.0     68
0.0     67
9.0     66
10.0    61
28.0    54
Name: count, dtype: int64
```

Observation:- "Absences" is a numercial variable. Mean and median are almost same and most of the values are have same number of distribution,, we can fill missing values

with mean

[13]:
```
# Imputing missing values by mean for "Absences" attribute
Absences_mean = dataRead['Absences'].mean()
dataRead['Absences'].fillna(Absences_mean, inplace=True)
```

Handlin & imputing missing values for "Tutoring" attribute

[14]:
```
#calculating mode of "Tutoring" attribute
print("Tutoring mode :{}".format(dataRead.Tutoring.mode()[0]))
#calculating value counts of "Tutoring" attribute
print("Tutoring value counts:{}".format(dataRead.Tutoring.value_counts()))
```

```
Tutoring mode :0.0
Tutoring value counts:Tutoring
0.0    1654
1.0     713
Name: count, dtype: int64
```

Observation:- "Tutoring" is a categorical nominal variable.we can fill missing values with mode

[15]:
```
# Imputing missing values by mode for "Tutoring" attribute
Tutoring_mode = dataRead['Tutoring'].mode()[0]
dataRead['Tutoring'].fillna(Tutoring_mode, inplace=True)
```

Handlin & imputing missing values for "ParentalSupport" attribute

[16]:
```
#calculating mode of "ParentalSupport" attribute
print("ParentalSupport mode :{}".format(dataRead.ParentalSupport.mode()[0]))
#calculating value counts of "ParentalSupport" attribute
print("ParentalSupport value counts:{}".format(dataRead.ParentalSupport.
  ↪value_counts()))
```

```
ParentalSupport mode :2.0
ParentalSupport value counts:ParentalSupport
2.0    737
3.0    692
1.0    485
4.0    253
0.0    211
Name: count, dtype: int64
```

Observation:- "ParentalSupport" is a categorical ordinal variable.we can fill missing values with mode

[17]:
```
# Imputing missing values by mode for "ParentalSupport" attribute
ParentalSupport_mode = dataRead['ParentalSupport'].mode()[0]
dataRead['ParentalSupport'].fillna(ParentalSupport_mode, inplace=True)
```

Handlin & imputing missing values for "Sports" attribute

[18]:
```
#calculating mode of "Sports" attribute
print("Sports mode :{}".format(dataRead.Sports.mode()[0]))
#calculating value counts of "Sports" attribute
print("Sports value counts:{}".format(dataRead.Sports.value_counts()))
```

```
Sports mode :0.0
Sports value counts:Sports
0.0    1659
1.0     722
Name: count, dtype: int64
```

Observation:- "Sports" is a categorical nominal variable.we can fill missing values with mode

[19]:
```
# Imputing missing values by mode for "Sports" attribute
Sports_mode = dataRead['Sports'].mode()[0]
dataRead['Sports'].fillna(Sports_mode, inplace=True)
```

Handlin & imputing missing values for "Music" attribute

[20]:
```
#calculating mode of "Music" attribute
print("Music mode :{}".format(dataRead.Sports.mode()[0]))
#calculating value counts of "Music" attribute
print("Music value counts:{}".format(dataRead.Music.value_counts()))
```

```
Music mode :0.0
Music value counts:Music
0.0    1750
1.0     430
Name: count, dtype: int64
```

Observation:- "Music" is a categorical nominal variable.we can fill missing values with mode

[21]:
```
# Imputing missing values by mode for "Music" attribute
Music_mode = dataRead['Music'].mode()[0]
dataRead['Music'].fillna(Music_mode, inplace=True)
```

1.5 Task 3: Give the reasons for each column that how you are handling the data.

1- "Age" is a numercial variable. Mean and median are almost same and value counts are also almost same, we can fill missing values with mean.

2- "StudyTimeWeekly" is a numercial variable. Mean and median are almost same and value counts are unique, we can fill missing values with mean

3- "Absences" is a numercial variable. Mean and median are almost same and most of the values are have same number of distribution,, we can fill

missing values with mean

4- “Tutoring” is a categorical nominal variable.we can fill missing values with mode.

5- “ParentalSupport” is a categorical ordinal variable.we can fill missing values with mode

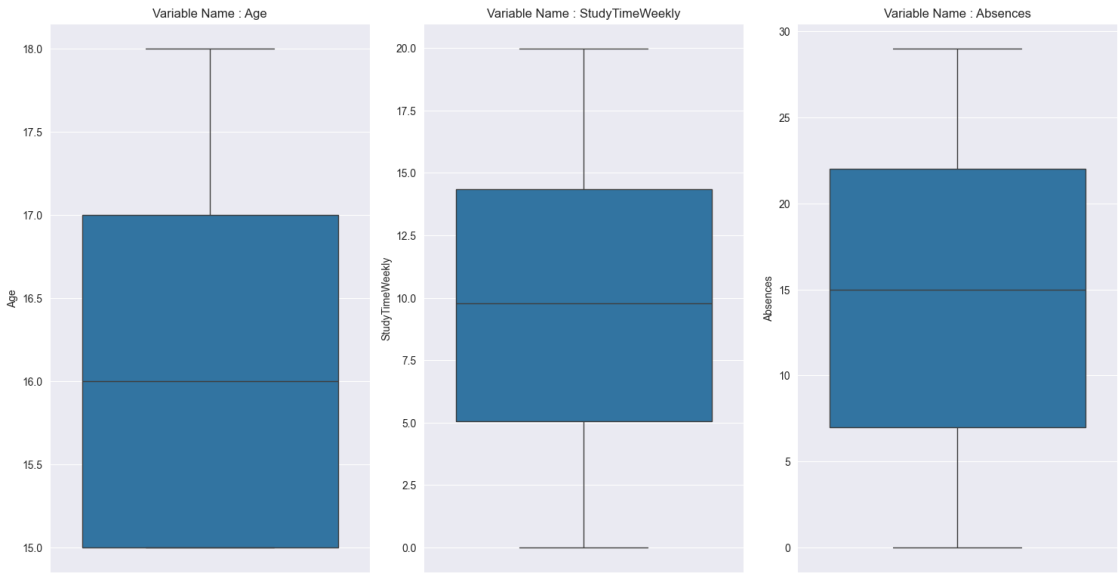
6- “Sports” is a categorical nominal variable.we can fill missing values with mode

7- “Music” is a categorical nominal variable.we can fill missing values with mode

1.5.1 Outlier Check for numerical variables

[22]:
```
# taking list of all numerocal attributes
numerical_attributes = ['Age', 'StudyTimeWeekly', 'Absences']
dataRead_new = dataRead[numerical_attributes]
# Set the parameters that control the general style of the plots.
sbn.set_style("darkgrid")
# Selecting all numerical attributes columns names
cols = dataRead_new.columns
#Plotting Box plot for each numerical attributes
plt.figure(figsize=(15, 30) )
for count, item in enumerate(cols, 1):
    plt.subplot(4, 3, count)
    sbn.boxplot(dataRead_new[item])
    plt.title(f"Variable Name : {item}")

plt.tight_layout()
plt.show()
```



Observation:- From the above box plots it shows that there are no outliers in the numerical attributes data

1.6 Task 4: Perform normalization of all numeric attributes

1.6.1 Min Max Scaler is used to perform normalization only for three numerical varaibles which are Age, StudyTimeweekly and Absences, as GPA is a target varaible mentioned in the assignment question , hence normalization has not been applied on it

[23]:
```
#importing MinMaxScaler library
from sklearn.preprocessing import MinMaxScaler
# taking list of all numerocal attributes
numerical_attributes_list = ['Age', 'StudyTimeWeekly', 'Absences']
scalling=MinMaxScaler()
dataRead[numerical_attributes_list] = scalling.
 ↪fit_transform(dataRead[numerical_attributes_list])
dataRead.describe()
```

[23]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	\
count	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	
mean	2196.500000	0.488434	0.510870	0.877508	1.746237	
std	690.655244	0.373697	0.499986	1.028476	1.000411	
min	1001.000000	0.000000	0.000000	0.000000	0.000000	
25%	1598.750000	0.000000	0.000000	0.000000	1.000000	
50%	2196.500000	0.333333	1.000000	0.000000	2.000000	
75%	2794.250000	0.666667	1.000000	2.000000	2.000000	
max	3392.000000	1.000000	1.000000	3.000000	4.000000	

	StudyTimeWeekly	Absences	Tutoring	ParentalSupport	\
count	2392.000000	2392.000000	2392.000000	2392.000000	
mean	0.488587	0.501608	0.298077	2.121656	
std	0.281216	0.291117	0.457509	1.119688	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.252881	0.241379	0.000000	1.000000	
50%	0.488587	0.517241	0.000000	2.000000	
75%	0.718476	0.758621	1.000000	3.000000	
max	1.000000	1.000000	1.000000	4.000000	

	Extracurricular	Sports	Music	Volunteering	GPA	\
count	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000	
mean	0.383361	0.301839	0.179766	0.157191	1.906186	
std	0.486307	0.459152	0.384073	0.364057	0.915156	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	1.174803	
50%	0.000000	0.000000	0.000000	0.000000	1.893393	
75%	1.000000	1.000000	0.000000	0.000000	2.622216	
max	1.000000	1.000000	1.000000	1.000000	4.000000	

```
         GradeClass
count  2392.000000
mean      2.983696
std       1.233908
min       0.000000
25%       2.000000
50%       4.000000
75%       4.000000
max       4.000000
```

1.7 Task 5: Print the average GPA for each grade respectively.

[24]:
```
#grouping all Gradeclass attribute w.r.t GPA target feature
average_gpa_eachgradeclass=dataRead.groupby("GradeClass")["GPA"].mean()
print(average_gpa_eachgradeclass)
```

```
GradeClass
0    3.102942
1    3.001673
2    2.659742
3    2.215545
4    1.208041
Name: GPA, dtype: float64
```

1.8 Task 6: Which grade is highly involved in Sports

[25]:
```
# crosstab to check %counts by sports and gradeclass
data_one=pd.crosstab(dataRead['GradeClass'],dataRead['Sports'],normalize=True)
data_two=data_one.applymap(lambda x : str(math.ceil(x*10000)/100) + "%")
data_two
```

[25]:

Sports	0.0	1.0
GradeClass		
0	2.89%	1.59%
1	7.45%	3.81%
2	11.75%	4.6%
3	12.09%	5.23%
4	35.67%	14.97%

Observation:-Grade 4 is highly involved in Sports as it contains 14.97% of data

1.9 Task 7: Calculate the proximity measure between (choose the measuring technique based on the column)

1. Tutoring and GPA
2. Studytime and GPA

3. Studytime and absences

7.1 Tutoring and GPA As Tutoring is nomial binary and GPA is continuous variable , the proximity measures will follow mix attribute approach

[26]:
```
from scipy.spatial.distance import cdist
# Tutoring Dissimilarity
tutor = ['Tutoring']
dataRead_new = dataRead[tutor]
tutor_dis=cdist(dataRead_new ,dataRead_new,metric='jaccard')

# GPA Dissimilarity
# defining a numby array with 0 with same data points
gpa_dis = np.zeros((dataRead.shape[0], dataRead.shape[0]))
# saving all the gpa values in a numpy array
gpa_val = dataRead['GPA'].values
# calculating max and min of gpa and their difference
gpa_val_max = dataRead['GPA'].max()
gpa_val_min = dataRead['GPA'].min()
gpa_max_min_diff = dataRead['GPA'].max() - dataRead['GPA'].min()
for i in range(dataRead.shape[0]):
    for j in range(dataRead.shape[0]):
        gpa_dis[i, j] = round(abs(gpa_val[i] - gpa_val[j]) / gpa_max_min_diff ,␣
↪5)


# combining two Dissimilarity matrix
tutor_gpa_dissimilarity = (tutor_dis + gpa_dis) / 2
print(tutor_gpa_dissimilarity)
```

```
[[0.       0.514215 0.852075 … 0.72336  0.640735 0.59865 ]
 [0.514215 0.       0.36629  … 0.237575 0.15495  0.112865]
 [0.852075 0.36629  0.       … 0.128715 0.211335 0.253425]
 …
 [0.72336  0.237575 0.128715 … 0.       0.08262  0.12471 ]
 [0.640735 0.15495  0.211335 … 0.08262  0.       0.04209 ]
 [0.59865  0.112865 0.253425 … 0.12471  0.04209  0.      ]]
```

7.2 Studytime and GPA As both variables are numeric , euclidean distance is calulated between objects for proximity measures

[27]:
```
from scipy.spatial.distance import cdist
# calculating dissimiliarity using euclidean distance
stu_gpa = ['StudyTimeWeekly', 'GPA']
dataRead_new = dataRead[stu_gpa]
dist_euc = cdist(dataRead_new, dataRead_new, metric='euclidean')
print(dist_euc)
```

```
[[0.         0.24898894 2.92315051 … 1.90215428 1.18553652 0.79559395]
 [0.24898894 0.         2.98344622 … 1.94876305 1.24863371 0.91093232]
 [2.92315051 2.98344622 0.         … 1.03789115 1.73988064 2.13880765]
 …
 [1.90215428 1.94876305 1.03789115 … 0.         0.71816913 1.13989384]
 [1.18553652 1.24863371 1.73988064 … 0.71816913 0.         0.43189606]
 [0.79559395 0.91093232 2.13880765 … 1.13989384 0.43189606 0.        ]]
```

7.3 Studytime and Absences As both variables are numeric , euclidean distance is calulated between objects for proximity measures

[28]:
```
# calculating dissimiliarity using euclidean distance
stu_gpa2 = ['StudyTimeWeekly', 'Absences']
dataRead_new2 = dataRead[stu_gpa2]
dist_euc2 = cdist(dataRead_new2, dataRead_new2, metric='euclidean')
print(dist_euc2)
```

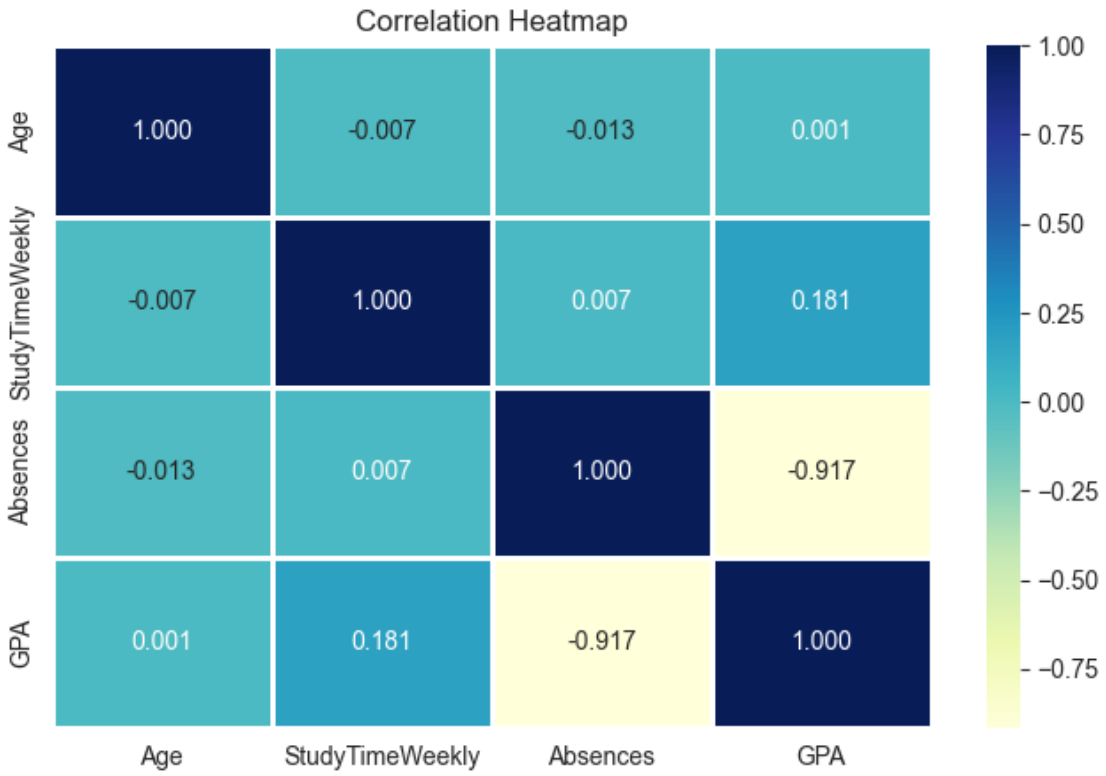
```
[[0.         0.3276086  1.02022634 … 0.79136832 0.50670972 0.23014807]
 [0.3276086  0.         1.0573668  … 0.81307438 0.60503861 0.46424003]
 [1.02022634 1.0573668  0.         … 0.24429298 0.51483069 0.8155069 ]
 …
 [0.79136832 0.81307438 0.24429298 … 0.         0.29932457 0.60187587]
 [0.50670972 0.60503861 0.51483069 … 0.29932457 0.         0.30361285]
 [0.23014807 0.46424003 0.8155069  … 0.60187587 0.30361285 0.        ]]
```

1.10 Task 8: Perform the feature selection considering GPA as the target variable and the rest of the columns as independent variables using the filter method. (use at least 3 methods)

1st Filter Method - Pearson Correaltion Method only for numerical variable

[29]:
```
# figure size of the plot
plt.figure(figsize=(8, 5))
# listing all the numerical variables
numerical_var_list = ['Age','StudyTimeWeekly', 'Absences','GPA']
dataRead_new= dataRead[numerical_var_list]
# Using Seaborn to create a heatmap
sbn.heatmap(dataRead_new.corr(method='pearson'), annot=True, fmt='.3f',␣
 ↪cmap='YlGnBu', linewidths=1)

plt.title('Correlation Heatmap')
plt.show()
# pearson correaltion table
dataRead_new.corr(method='pearson')
```



[29]:

	Age	StudyTimeWeekly	Absences	GPA
Age	1.000000	-0.006979	-0.013086	0.000996
StudyTimeWeekly	-0.006979	1.000000	0.007204	0.181166
Absences	-0.013086	0.007204	1.000000	-0.916764
GPA	0.000996	0.181166	-0.916764	1.000000

For other filering methods, GPA has to be discretized and Min of GPA is 0 and max of GPA is 4, then max-min = 4,then bins will be 4 and all other numerical variables are binned as well to calculate variable importance. As Age variable contains only 4 discrete values , Binning is not needed for the Age field

[30]:
```
# Binning of GPA
dataRead['GPA_Binned'] = pd.cut(dataRead['GPA'], bins=4, labels=False)
print(f'Max of GPA by Bin : {dataRead.groupby("GPA_Binned")["GPA"].max()}')
print(f'Minimum of GPA by Bin : {dataRead.groupby("GPA_Binned")["GPA"].min()}')

# Binning of StudyTimeWeekly
dataRead['STW_Binned'] = pd.cut(dataRead['StudyTimeWeekly'], bins=10,␣
 ↪labels=False)
print(f'Max of StudyTimeWeekly by Bin : {dataRead.
 ↪groupby("STW_Binned")["StudyTimeWeekly"].max()}')
```

```
print(f'Minimum of StudyTimeWeekly by Bin : {dataRead.
  ↪groupby("STW_Binned")["StudyTimeWeekly"].min()}')

# Binning of Absences
dataRead['Abs_Binned'] = pd.cut(dataRead['Absences'], bins=10, labels=False)
print(f'Max of Absences by Bin : {dataRead.groupby("Abs_Binned")["Absences"].
  ↪max()}')
print(f'Minimum of Absences by Bin : {dataRead.
  ↪groupby("Abs_Binned")["Absences"].min()}')
```

```
Max of GPA by Bin : GPA_Binned
0    0.999089
1    1.999572
2    2.999544
3    4.000000
Name: GPA, dtype: float64
Minimum of GPA by Bin : GPA_Binned
0    0.000000
1    1.000776
2    2.000084
3    3.004233
Name: GPA, dtype: float64
Max of StudyTimeWeekly by Bin : STW_Binned
0    0.099994
1    0.199557
2    0.298963
3    0.399446
4    0.499557
5    0.600000
6    0.699386
7    0.798807
8    0.899619
9    1.000000
Name: StudyTimeWeekly, dtype: float64
Minimum of StudyTimeWeekly by Bin : STW_Binned
0    0.000000
1    0.100593
2    0.200050
3    0.300948
4    0.400562
5    0.500140
6    0.600067
7    0.701894
8    0.801231
9    0.900231
Name: StudyTimeWeekly, dtype: float64
Max of Absences by Bin : Abs_Binned
```

```
0    0.068966
1    0.172414
2    0.275862
3    0.379310
4    0.482759
5    0.586207
6    0.689655
7    0.793103
8    0.896552
9    1.000000
Name: Absences, dtype: float64
Minimum of Absences by Bin : Abs_Binned
0    0.000000
1    0.103448
2    0.206897
3    0.310345
4    0.413793
5    0.501608
6    0.620690
7    0.724138
8    0.827586
9    0.931034
Name: Absences, dtype: float64
```

2nd Filter Method - Mutual Information Classifier Method for Categorical variables

[31]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# X & Y intialization
categorical_var_list =␣
 ↪['Age','Abs_Binned','STW_Binned','Gender','Ethnicity','ParentalEducation','Tutoring','Paren
X= dataRead[categorical_var_list]
Y = dataRead["GPA_Binned"]
# mutual info function fit
mut_info = SelectKBest(score_func=mutual_info_classif,k=5)
mut_info.fit_transform(X,Y)
# dataframe creation to contain mutual info score with variable names
mut_info_score = pd.DataFrame(mut_info.scores_,index=X.
 ↪columns,columns=['Mutual_Info_Score'])
# sorting mutual infor score in descending order
mut_info_score.sort_values(by=['Mutual_Info_Score'],ascending=False)
```

[31]:

	Mutual_Info_Score
GradeClass	0.737462
Abs_Binned	0.681660
Extracurricular	0.017072
STW_Binned	0.013107
Gender	0.012981

```
Tutoring                    0.010830
Music                       0.008806
Age                         0.005327
Ethnicity                   0.000000
ParentalEducation           0.000000
ParentalSupport             0.000000
Sports                      0.000000
Volunteering                0.000000
```

3rd Filter Method : Chi Square Test among categorical variables

[32]:
```
from scipy.stats import chi2_contingency

# categorical variable list define
categorical_var_list =␣
 ↪['Age','Abs_Binned','STW_Binned','Gender','Ethnicity','ParentalEducation','Tutoring','Paren
list1 = []

# calculation p_value for chi sqaure test for all categorical variable present␣
 ↪in the list above
for column in categorical_var_list:
    contingency_table = pd.crosstab(dataRead[column], dataRead['GPA_Binned'])
    chi2, p, degrees_of_freedom, expected_result =␣
 ↪chi2_contingency(contingency_table)
    list1.append(round(p,6))

# dataframe creation to contain variable names and their correspondi p_vales␣
 ↪for chi sqaure
df2 = pd.DataFrame()
df2['Column_Name'] = categorical_var_list
df2['Chi_Sq_P_Value'] = list1

df2.sort_values(by=['Chi_Sq_P_Value'])
```

[32]:

	Column_Name	Chi_Sq_P_Value
1	Abs_Binned	0.000000
6	Tutoring	0.000000
7	ParentalSupport	0.000000
12	GradeClass	0.000000
2	STW_Binned	0.000001
8	Extracurricular	0.000071
9	Sports	0.000770
10	Music	0.001805
5	ParentalEducation	0.178146
0	Age	0.248155
11	Volunteering	0.428389
4	Ethnicity	0.514743

```
3              Gender        0.900460
```

4th Filter Method - Anova Method for categorical variables using F_Classif

[33]:
```
from sklearn.feature_selection import f_classif,SelectKBest
# Anova model fit
model = SelectKBest(score_func=f_classif,k=10)
model.fit(X,Y)
# model score calculation along with p_values
df_score = pd.DataFrame(model.scores_)
df_pvalue = pd.DataFrame(np.round(model.pvalues_,3))
df_col = pd.DataFrame(X.columns)
df_new = pd.concat([df_col,df_score,df_pvalue],axis=1)
# Assigning the column names
df_new.columns = ["Col_Names","Anova Score","Anova P-Value"]
df_new.sort_values(by=['Anova Score'],ascending = False)
# print(df_new.nlargest(10,columns="Anova Score"))
```

[33]:

	Col_Names	Anova Score	Anova P-Value
1	Abs_Binned	2741.224189	0.000
12	GradeClass	1882.101515	0.000
7	ParentalSupport	28.442936	0.000
2	STW_Binned	22.703090	0.000
6	Tutoring	21.956430	0.000
8	Extracurricular	7.330401	0.000
9	Sports	5.636696	0.001
10	Music	5.027663	0.002
4	Ethnicity	1.856574	0.135
11	Volunteering	0.923005	0.429
5	ParentalEducation	0.685961	0.561
3	Gender	0.193841	0.901
0	Age	0.034341	0.991

1.10.1 Task 9: Perform a comparison between various feature selection methods and report the best 5 features among each technique.

9.1. Pearson Correlation :

- Pearson correaltion method has been applied on only numerical features and there are only 3 independent numerical variables - 1- Absences, 2- StudyTimeweekly 3- Age. Target variable is GPA.
- The correaltion coeffienct between Absences and GPA is super strong -0.91. we can say there is negetive strong relationship between Absences and GPA. The correaltion between StudytimeWeekly and GPA is below .02 and same goes for the field Age as well.

9.2. Mutual Information :

- Mutual Information Classfier is used to perform this filter method for feature selection

- StudyTimeWeekly and Absences have also been binned for Mutual Information method
- Age has not been converted to a binning field as the values of Age are discrete and their distribution are almost same for all the four vales - 15,16,17 & 18
- GPA, the target variable is also discretized as a categorical variable is needed for mututal information classifier method
- We can see the top 5 varaibles who have got the most mutual information with GPA_Binned (GPA Binned Field) variable are -
  1. GradeClass
  2. Abs_Binned (Absences Binned Field)
  3. Tutoring
  4. Sports
  5. Extracurricular

9.3. Chi Square :

- As Chi Square needs the target of dependent variable a categorical field , hence GPA field has been discretized and converted to categorical to perform chi square
- StudyTimeWeekly and Absences have also been binned for chi square
- Age has not been converted to a binning field as the values of Age are discrete and their distribution are almost same for all the four vales - 15,16,17 & 18
- After performing the Chi Square test, based on the P-Values for every features in ascending order, we can say that top 5 features will be
  1. Abs_Binned (Absences Binned Field)
  2. Tutoring
  3. ParentalSupprot
  4. GradeClass
  5. STW_Binned (StudyTimeWeekly Binned variable)
- In Chi Square we usually take those features whose p values are less than 0.05 and if p values are less than 0.05 we reject the null hypothesis.

9.4. Anova :

- Anova method can be also used for feature selection
- F_classif method has been used here
- Discretized has been done on StudytimeWeekly, Absences and GPA like in Mutual Information Score
- top 5 variables which have the most F stats score are 1. Abs_Binned (Absences Binned Field) 2. GradeClass 3. ParentalSupport 4. STW_Binned (StudyTimeWeekly Binned variable) 5. Tutoring

1.11 Task 10: Write down the understanding the most important feature in words

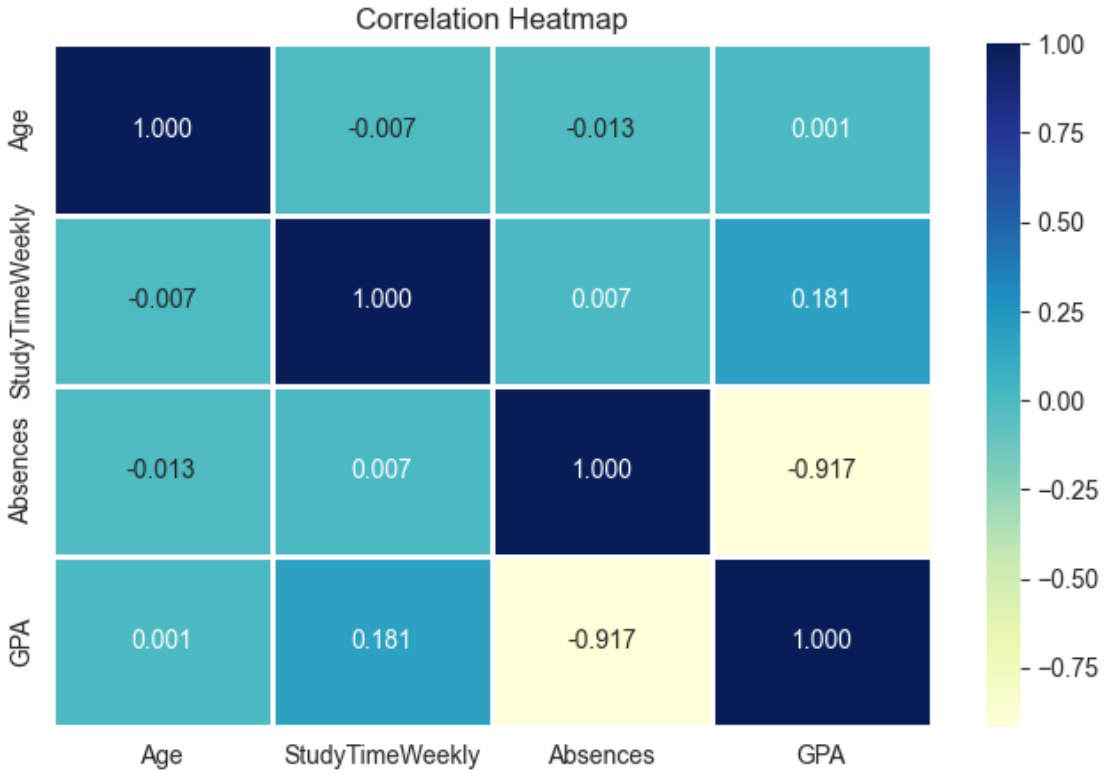
from Upper filter methods, it can be conlcuded that most important features are those which have a strong relationship with target variable.

```
- Absences field is numerial field and has a stong negative relationship with GPA. - 0.91, this
- GradeClass is another most important features and we can see it is present in all top 5 featu
- Abs_Binned is also present in all filter methods and Absence has a strong relationship with G
```

- We can say that most important features using all 4 methods are Abseneces and GradeClass

1.12 Task 11: Plot the correlation plot between the independent feature and the target variable.

[34]:
```
# figure size of the plot
plt.figure(figsize=(8, 5))
# listing all the numerical variables
numerical_var_list = ['Age','StudyTimeWeekly', 'Absences','GPA']
df_new= dataRead[numerical_var_list]
# Using Seaborn to create a heatmap
sbn.heatmap(df_new.corr(method='pearson'), annot=True, fmt='.3f',␣
 ↪cmap='YlGnBu', linewidths=1)
plt.title('Correlation Heatmap')
plt.show()
# pearson correaltion table
df_new.corr(method='pearson')
```



[34]:

	Age	StudyTimeWeekly	Absences	GPA
Age	1.000000	-0.006979	-0.013086	0.000996
StudyTimeWeekly	-0.006979	1.000000	0.007204	0.181166
Absences	-0.013086	0.007204	1.000000	-0.916764

```
GPA              0.000996         0.181166 -0.916764  1.000000
```

1.13 12. Write down the understanding of feature correlation in words.

Pearson Correlation :

* Pearson correaltion method has been applied on only numerical features and there are only 3 i
  1- Absences, 2- StudyTimeweekly 3- Age. Target variable is GPA.
* The correaltion coeffienct between Absences and GPA is super strong -0.91. we can say there
* The more the Absences will be , the less GPA will be
* The correaltion between StudytimeWeekly and GPA is below .02 and same goes for the field Age