Student Performance Data Set Project

Nuriye Merve TATLIDİL – 150212002

1-Introduction

What can affect a student's education life? What can a family provide for a high school student to succeed in their lessons? Do long studies at the desk bring benefits or extra-curricular activities? As data scientists, we need to ask a variety of questions for the issues we focus on.We should use domain information, statistics and algorithmic programming together in our projects.

2- General View of the Data

1) school - student's school (binary: "GP" - Gabriel Pereira or "MS" - Mousinho da Silveira)

2) sex - student's sex (binary: "F" - female or "M" - male)

3) age - student's age (numeric: from 15 to 22)

4) address - student's home address type (binary: "U" - urban or "R" - rural)

5) famsize - family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3)

6) Pstatus - parent's cohabitation status (binary: "T" - living together or "A" - apart)

7) Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)

8) Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)

9) Mjob - mother's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")

10) Fjob - father's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")

11) reason - reason to choose this school (nominal: close to "home", school "reputation", "course" preference or "other")

12) guardian - student's guardian (nominal: "mother", "father" or "other")

13) 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)

14) studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)

15) failures - number of past class failures (numeric: n if 1<=n<3, else 4)

16) schoolsup - extra educational support (binary: yes or no)

17) famsup - family educational support (binary: yes or no)

18) paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)

19) activities - extra-curricular activities (binary: yes or no)

20) nursery - attended nursery school (binary: yes or no)

21) higher - wants to take higher education (binary: yes or no)

22) internet - Internet access at home (binary: yes or no)

23) romantic - with a romantic relationship (binary: yes or no)

24) famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)

25) freetime - free time after school (numeric: from 1 - very low to 5 - very high)

26) goout - going out with friends (numeric: from 1 - very low to 5 - very high)

27) Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)

28) Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)

29) health - current health status (numeric: from 1 - very bad to 5 - very good)

30) absences - number of school absences (numeric: from 0 to 93)

31) G1 - first period grade (numeric: from 0 to 20)

31) G2 - second period grade (numeric: from 0 to 20)

32) G3 - final grade (numeric: from 0 to 20, output target)

*The G3 attribute in the data set shows the final grades of the students.I've made a classification between notes so I can easily apply Classification algorithms.Accordingly :

15-20 : A

10-15 : B

5-10 : C

0-5 : D

3-Importing Library

Since I will use the dataset as a dataframe, I first imported the pandas library. Then I added the dataset to my project. I will make some changes on the attributes of my dataset, so I saved the dataset with the first version name because I want to show the first and final version of the dataset.

```
import pandas as pd
df = pd.read_csv("../data/student-mat-firstversion.csv", sep=';')
df
#Firstly I imported the pandas library and then I read the comma separated excel file
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	letter_grade
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	3	4	1	1	3	6	5	6	6	C
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	3	3	1	1	3	4	5	5	6	C
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	3	2	2	3	3	10	7	8	10	B
3	GP	F	15	U	GT3	T	4	2	health	services	...	2	2	1	1	5	2	15	14	15	A
4	GP	F	16	U	GT3	T	3	3	other	other	...	3	2	1	2	5	4	6	10	10	B
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	4	4	5	4	11	9	9	9	C
391	MS	M	17	U	LE3	T	3	1	services	services	...	4	5	3	4	2	3	14	16	16	A
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	3	3	3	3	3	10	8	7	C
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	1	3	4	5	0	11	12	10	B
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	2	3	3	3	5	5	8	9	9	C

395 rows × 34 columns

I showed some important values of the attributes in the data set. I applied the transposition because I thought it looked more understandable when I received the transposition.

```
In [8]: df.describe().T #I showed some important values of the dataset
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
age	395.0	16.696203	1.276043	15.0	16.0	17.0	18.0	22.0
Medu	395.0	2.749367	1.094735	0.0	2.0	3.0	4.0	4.0
Fedu	395.0	2.521519	1.088201	0.0	2.0	2.0	3.0	4.0
traveltime	395.0	1.448101	0.697505	1.0	1.0	1.0	2.0	4.0
studytime	395.0	2.035443	0.839240	1.0	1.0	2.0	2.0	4.0
failures	395.0	0.334177	0.743651	0.0	0.0	0.0	0.0	3.0
famrel	395.0	3.944304	0.896659	1.0	4.0	4.0	5.0	5.0
freetime	395.0	3.235443	0.998862	1.0	3.0	3.0	4.0	5.0
goout	395.0	3.108861	1.113278	1.0	2.0	3.0	4.0	5.0
Dalc	395.0	1.481013	0.890741	1.0	1.0	1.0	2.0	5.0
Walc	395.0	2.291139	1.287897	1.0	1.0	2.0	3.0	5.0
health	395.0	3.554430	1.390303	1.0	3.0	4.0	5.0	5.0
absences	395.0	5.708861	8.003096	0.0	0.0	4.0	8.0	75.0
G1	395.0	10.908861	3.319195	3.0	8.0	11.0	13.0	19.0
G2	395.0	10.713924	3.761505	0.0	9.0	11.0	13.0	19.0
G3	395.0	10.415190	4.581443	0.0	8.0	11.0	14.0	20.0

It is important for the application of algorithms that the types of our data are categorical or numerical.For this reason, I looked at how many numerical and categorical variables there are because we would convert categorical variables into numerical variables in the next steps.We can also see the size of the data set in this way.

```
df.info() #I showed info data type, column and row number, memory usage information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 34 columns):
school          395 non-null object
sex             395 non-null object
age             395 non-null int64
address         395 non-null object
famsize         395 non-null object
Pstatus         395 non-null object
Medu            395 non-null int64
Fedu            395 non-null int64
Mjob            395 non-null object
Fjob            395 non-null object
reason          395 non-null object
guardian        395 non-null object
traveltime      395 non-null int64
studytime       395 non-null int64
failures        395 non-null int64
schoolsup       395 non-null object
famsup          395 non-null object
paid            395 non-null object
activities      395 non-null object
nursery         395 non-null object
higher          395 non-null object
internet        395 non-null object
romantic        395 non-null object
famrel          395 non-null int64
freetime        395 non-null int64
goout           395 non-null int64
Dalc            395 non-null int64
Walc            395 non-null int64
health          395 non-null int64
absences        395 non-null int64
G1              395 non-null int64
G2              395 non-null int64
G3              395 non-null int64
letter_grade    395 non-null object
dtypes: int64(16), object(18)
memory usage: 105.0+ KB
```

4-Missing Data

There are many steps in the data preprocessing phase. One of them is data cleaning. In this phase, we check the missing, noisy and outlier data.

First, I check for missing data.

1-Is there any missing value?

```
: df.isnull().values.any()
```

```
: False
```

I didn't take action because there is no missing value

5- Data Transformation

One – Hot Transformation

I applied one-hot transformation to attributes that have more than 2 values.because I don't have to give numbers like 0.1,2,3 to the data when it's like this.If I had given numbers like 0.1,2,3, I could have made some values more important.

```
7]: clmns = ['reason','Mjob','Fjob','guardian']
df2 = pd.concat([df, pd.get_dummies(df[clmns])], axis=1).drop(clmns, axis = 1)
df2
#I made one-hot transformation because 'reason', 'Mjob', 'Fjob', 'guardian' attributes have more than 2 variables
#In one-hot transformation, the values of the attributes are regenerated separately. I deleted the old values to avoid
#confusion.
```

7]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	...	Mjob_services	Mjob_teacher	Fjob_at_home	Fjob_health	Fjob_othe
0	GP	F	18	U	GT3	A	4	4	2	2	...	0	0	0	0	[illegible]
1	GP	F	17	U	GT3	T	1	1	1	2	...	0	0	0	0	[illegible]
2	GP	F	15	U	LE3	T	1	1	1	2	...	0	0	0	0	[illegible]
3	GP	F	15	U	GT3	T	4	2	1	3	...	0	0	0	0	[illegible]
4	GP	F	16	U	GT3	T	3	3	1	2	...	0	0	0	0	[illegible]
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	1	2	...	1	0	0	0	[illegible]
391	MS	M	17	U	LE3	T	3	1	2	1	...	1	0	0	0	[illegible]
392	MS	M	21	R	GT3	T	1	1	1	1	...	0	0	0	0	[illegible]
393	MS	M	18	R	LE3	T	3	2	3	1	...	1	0	0	0	[illegible]
394	MS	M	19	U	LE3	T	1	1	1	1	...	0	0	1	0	[illegible]

395 rows × 44 columns

0 – 1 Transformation

I have converted 0-1 to attributes that have 2 values in the data set.

```
from sklearn.preprocessing import LabelEncoder
lbe = LabelEncoder()
clms = ['school','sex','address','famsize','Pstatus','schoolsup','famsup','paid','activities','nursery','higher','internet','rom
for i in clms:
    df2[i] = lbe.fit_transform(df2[i])
df2
#I convert 0-1 with the help of clms list structure and a loop
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	...	Mjob_services	Mjob_teacher	Fjob_at_home	Fjob_health	Fjob_othe
0	0	0	18	1	0	0	4	4	2	2	...	0	0	0	0	
1	0	0	17	1	0	1	1	1	1	2	...	0	0	0	0	
2	0	0	15	1	1	1	1	1	1	2	...	0	0	0	0	
3	0	0	15	1	0	1	4	2	1	3	...	0	0	0	0	
4	0	0	16	1	0	1	3	3	1	2	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	1	1	20	1	1	0	2	2	1	2	...	1	0	0	0	
391	1	1	17	1	1	1	3	1	2	1	...	1	0	0	0	
392	1	1	21	0	0	1	1	1	1	1	...	0	0	0	0	
393	1	1	18	0	1	1	3	2	3	1	...	1	0	0	0	
394	1	1	19	1	1	1	1	1	1	1	...	0	0	1	0	

395 rows × 44 columns

school : GP = 0 MS = 1

sex : F = 0 M = 1

address : U = 1 R = 0

famsize : GT3 = 0 LE3 = 1

Pstatus : A = 0 T = 1

schoolsup : Yes = 1 No = 0

famsup : Yes = 1 No = 0

paid : Yes = 1 No = 0

activities : Yes = 1 No = 0

nursery : Yes = 1 No = 0

higher : Yes = 1 No = 0

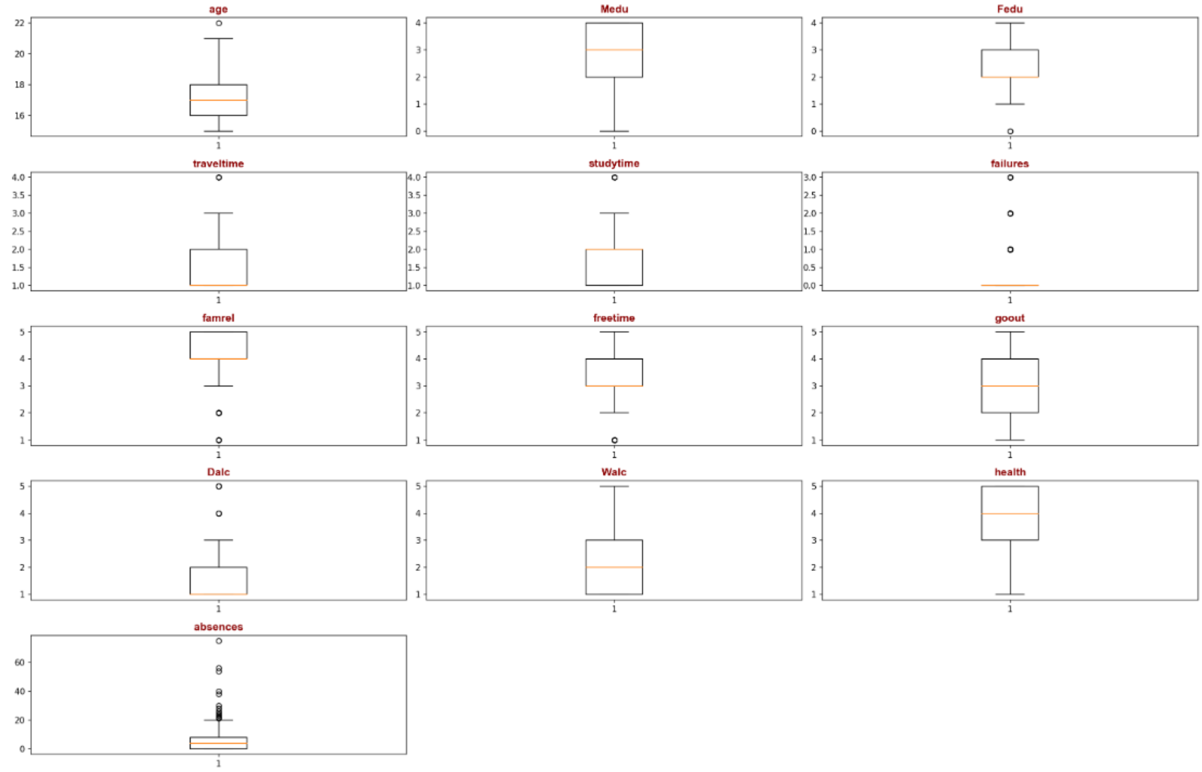
internet : Yes = 1 No = 0

romantic : Yes = 1 No = 0

6-Outlier Data

It is important to find solutions to outlier data because outlier data can cause the results to go in the wrong direction.I have visualized whether there is an outlier value in the data with the boxplot.

```
import matplotlib.pyplot as plt
baslik_font={'family':'arial','color':'darkred','weight':'bold','size':13}
eksen_font={'family':'arial','color':'darkblue','weight':'bold','size':10}
plt.figure(figsize=(20,13),dpi=150)
fill_list=['age','Medu','Fedu','traveltime','studytime','failures','famrel','freetime','goout','Dalc','Walc','health','absences'
for i,col in enumerate(fill_list):
    plt.subplot(5,3,i+1)
    plt.boxplot(col, data=df2)
    plt.title(col,fontdict=baslik_font)
plt.tight_layout()
plt.show()
#Let's visualize outlier data
```

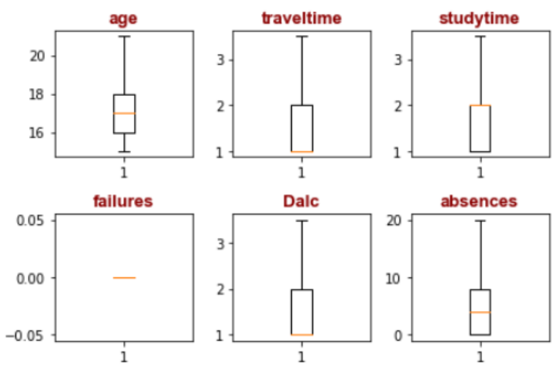


We can do many solutions for Outlier data.The method of suppression made sense to me.Because with the suppression method, I get the data closer to the lowest or highest limit value.

Suppression method

```
c = ['age','traveltime','studytime','failures','Dalc','absences']
for col in c :
    df2_col = df2[col]
    Q1 = df2_col.quantile(0.25) #first interquartile range
    Q3 = df2_col.quantile(0.75) #third interquartile range
    IQR = Q3-Q1   #IQR tells how far the middle values spread.And this is its formula
    low_limit = Q1 - 1.5*IQR #low limit formula
    high_limit = Q3 + 1.5*IQR #high limit formula
    outliers_col_higher = (df2_col > high_limit) #outlier data greater than the upper limit
    df2_col[outliers_col_higher] = high_limit
```

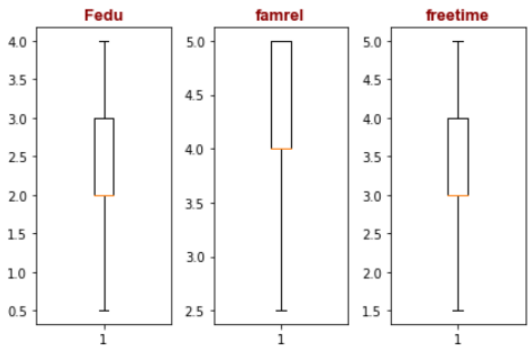
```
for i,col in enumerate(c):
    plt.subplot(2,3,i+1)
    plt.boxplot(col, data=df2)
    plt.title(col,fontdict=baslik_font)
plt.tight_layout()
plt.show()
#let's visualize new data
```



I have done the attibutes separately, which I will bring closer to the highest and lowest limits.

```
]: col = ['Fedu','famrel','freetime']
   for c in col :
       df2_c = df2[c]
       Q1 = df2_c.quantile(0.25) #first interquartile range
       Q3 = df2_c.quantile(0.75) #third interquartile range
       IQR = Q3-Q1   #IQR tells how far the middle values spread.And this is its formula
       low_limit = Q1 - 1.5*IQR #low limit formula
       high_limit = Q3 + 1.5*IQR #high limit formula
       outliers_c_lower = (df2_c < low_limit) #outlier data greater than the upper limit
       df2_c[outliers_c_lower] = low_limit
```

```
]: for i,colum in enumerate(col):
       plt.subplot(1,3,i+1)
       plt.boxplot(colum, data=df2)
       plt.title(colum,fontdict=baslik_font)
   plt.tight_layout()
   plt.show()
   #let's visualize new data
```



ALGORITHMS

I import some important packages and libraries to evaluate algorithms and their results. I assign the "letter_grade" attribute to the y value, which I will use as the tag value. The other attributes remain as x.

I divide the data set into 70% education and 30% test. I'm doing "stratify = y" to make sure the class rates are maintained.

```
3]: from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
    from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.svm import SVC
```

```
4]: y = df2["letter_grade"]
    x = df2.drop(["letter_grade"], axis=1)
```

```
5]: x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                        test_size=0.30,
                                                        stratify=y)
    #I have separated the data set as 70% train and 30% test
```

```
6]: x_test
```

6]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	...	Mjob_services	Mjob_teacher	Fjob_at_home	Fjob_health	Fjob_othe
235	0	1	16	1	0	1	3	2.0	2.0	3.0	...	0	0	0	0	
58	0	1	15	1	1	1	1	2.0	1.0	2.0	...	0	0	1	0	
373	1	0	17	0	0	1	1	2.0	1.0	1.0	...	0	0	0	0	
385	1	0	18	0	0	1	2	2.0	2.0	3.0	...	0	0	0	0	
344	0	0	18	1	0	1	2	3.0	1.0	3.0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
301	0	1	17	1	1	1	4	4.0	2.0	1.0	...	0	0	0	0	
353	1	1	19	0	0	1	1	1.0	3.0	1.0	...	0	0	0	0	
378	1	0	18	1	0	1	3	3.0	1.0	2.0	...	0	0	0	0	
172	0	1	17	1	1	1	4	4.0	1.0	2.0	...	0	1	0	0	
178	0	1	16	0	0	1	4	2.0	1.0	1.0	...	0	1	0	0	

119 rows × 46 columns

1-NAIVE BAYES

I worked a lot on the sample code given in the project guide. I did the primer calculation and the evidence formula according to my own dataset. I wanted to keep them in the dataframe as there are 46 attributes and I have 4 classes. I succeeded this part.

p_x_C1 = (1 / (math.sqrt(2 * math.pi * std_C1**2))) * math.exp(-1 * (test_x[0][0] - m_C1)**2 / (2 * std_C1**2))

Because the loop had to return for both 119 test data, for the letter value, and for 46 attributes. I couldn't do it here. I didn't want to missing the assignment. So I learned how he did it from Umut and did this part like him.

```
Xtrain = pd.concat([x_train, y_train], axis=1)
Xtrain
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	...	Mjob_teacher	Fjob_at_home	Fjob_health	Fjob_other	Fjob_services
56	0	0	15	1	0	0	4	3.0	1.0	2.0	...	0	0	0	0	1
4	0	0	16	1	0	1	3	3.0	1.0	2.0	...	0	0	0	1	0
15	0	0	16	1	0	1	4	4.0	1.0	1.0	...	0	0	0	1	0
382	1	1	17	1	0	1	2	3.0	2.0	2.0	...	0	0	0	0	1
13	0	1	15	1	0	1	4	3.0	2.0	2.0	...	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
285	0	1	17	1	0	1	1	1.0	1.0	2.0	...	0	0	0	1	0
254	0	1	17	0	0	1	2	1.0	1.0	1.0	...	0	0	0	1	0
258	0	1	18	1	0	1	2	1.0	1.0	2.0	...	0	0	0	1	0
148	0	1	16	1	0	1	4	4.0	1.0	1.0	...	1	0	0	0	0
59	0	0	16	1	0	1	4	2.0	1.0	2.0	...	0	0	0	1	0

276 rows × 47 columns

```
## assume Gaussian (Normal) Distribution
## Bayes Theorem: Posterior = (Likelihood x Prior) / Evidence
## Bayes Theorem: p_C_x = (P_x_C x p_C) / p_x
Xtrain_A = Xtrain[Xtrain['letter_grade'] == 'A']
Xtrain_B = Xtrain[Xtrain['letter_grade'] == 'B']
Xtrain_C = Xtrain[Xtrain['letter_grade'] == 'C']
Xtrain_D = Xtrain[Xtrain['letter_grade'] == 'D']
## Calculate prior probabilities: p_C1, p_C2, p_C3, p_C4 - C1:A C2:B C3:C C4:D
p_C1 = Xtrain_A.shape[0] / Xtrain.shape[0]
print(p_C1)
p_C2 = Xtrain_B.shape[0] / Xtrain.shape[0]
print(p_C2)
p_C3 = Xtrain_C.shape[0] / Xtrain.shape[0]
print(p_C3)
p_C4 = Xtrain_D.shape[0] / Xtrain.shape[0]
print(p_C4)
```

```
0.18478260869565216
0.4855072463768116
0.2318840579710145
0.09782608695652174
```

[24]:
```python
m_C1 = Xtrain[Xtrain['letter_grade']== 'A'].iloc[:,0:46].mean()
std_C1 =Xtrain[Xtrain['letter_grade']== 'A'].iloc[:,0:46].std()
print(m_C1)
print(std_C1)
m_C2 = Xtrain[Xtrain['letter_grade']== 'B'].iloc[:,0:46].mean()
std_C2 =Xtrain[Xtrain['letter_grade']== 'B'].iloc[:,0:46].std()
print(m_C2)
print(std_C2)
m_C3 = Xtrain[Xtrain['letter_grade']== 'C'].iloc[:,0:46].mean()
std_C3 =Xtrain[Xtrain['letter_grade']== 'C'].iloc[:,0:46].std()
print(m_C3)
print(std_C3)
m_C4 = Xtrain[Xtrain['letter_grade']== 'D'].iloc[:,0:46].mean()
std_C4 =Xtrain[Xtrain['letter_grade']== 'D'].iloc[:,0:46].std()
print(m_C4)
print(std_C4)
```

```
schoolsup            0.119403
famsup               0.567164
paid                 0.470149
activities           0.514925
nursery              0.753731
higher               0.955224
internet             0.813433
romantic             0.335821
famrel               4.033582
freetime             3.167910
goout                3.000000
Dalc                 1.414179
Walc                 2.320896
health               3.477612
absences             5.462687
G1                  11.231343
G2                  11.291045
G3                  11.552239
reason_course        0.358209
reason_home          0.313433
```

8]:
```python
import numpy as np
X_test_arr = x_test.to_numpy()
import math
pred = []
for test in X_test_arr:


    p_x_C1 = (1 / ((2 * math.pi * std_C1**2)**0.5)) * 2.7182**(-1 * (test - m_C1)**2 / (2 * std_C1**2))
    ##print(p_x_C1)

    p_x_C2 = (1 / ((2 * math.pi * std_C2**2)**0.5)) * 2.7182**(-1 * (test - m_C2)**2 / (2 * std_C2**2))
    ##print(p_x_C2)

    p_x_C3 = (1 / ((2 * math.pi * std_C3**2)**0.5)) * 2.7182**(-1 * (test - m_C3)**2 / (2 * std_C3**2))
    ##print(p_x_C3)

    p_x_C4 = (1 / ((2 * math.pi * std_C4**2)**0.5)) * 2.7182**(-1 * (test - m_C4)**2 / (2 * std_C4**2))
    ##print(p_x_C4)

    ## Calculate evidence: p_x_C1 * p_C1 + p_x_C2 * p_C2 toplam olasılık .prod() içindeki tüm sayıları çarp

    p_x = p_x_C1.prod() * p_C1 + p_x_C2.prod() * p_C2 + p_x_C3.prod() * p_C3 + p_x_C4.prod() * p_C4
    p_x

    ## Calculate posterior probabilities olasılıklar çarpımlarıyla hesaplanıyor
    p_C1_x = p_x_C1.prod() * p_C1 / p_x
    p_C2_x = p_x_C2.prod() * p_C2 / p_x
    p_C3_x = p_x_C3.prod() * p_C3 / p_x
    p_C4_x = p_x_C4.prod() * p_C4 / p_x

    if(p_C1_x > p_C2_x):

        if(p_C1_x > p_C3_x):

            if(p_C1_x> p_C4_x):
                pred.append('A')
    if(p_C2_x > p_C1_x):
```

```
                pred.append('A')
    if(p_C2_x > p_C1_x):

        if(p_C2_x > p_C3_x):

            if(p_C2_x> p_C4_x):
                pred.append('B')
    if(p_C3_x > p_C1_x):

        if(p_C3_x > p_C2_x):

            if(p_C3_x> p_C4_x):
                pred.append('C')
    if(p_C4_x > p_C1_x):

        if(p_C4_x > p_C2_x):

            if(p_C4_x> p_C3_x):
                pred.append('D')
print(pred)
```

```
['C', 'C', 'C', 'A', 'B', 'B', 'C', 'C', 'B', 'B', 'B', 'D', 'B', 'C', 'B', 'A', 'B', 'A', 'B', 'B', 'C', 'B', 'B', 'A', 'B',
'A', 'B', 'A', 'C', 'D', 'B', 'B', 'A', 'C', 'D', 'B', 'B', 'C', 'C', 'B', 'B', 'B', 'B', 'A', 'C', 'B', 'A', 'C', 'A', 'C',
'B', 'B', 'B', 'D', 'B', 'D', 'A', 'B', 'A', 'A', 'B', 'C', 'C', 'D', 'B', 'D', 'B', 'C', 'A', 'C', 'D', 'A', 'B', 'A', 'C',
'A', 'C', 'D', 'B', 'A', 'B', 'A', 'B', 'C', 'A', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'C', 'A', 'C', 'B', 'C', 'B', 'B',
'B', 'D', 'B', 'D', 'C', 'B', 'A', 'A', 'D', 'C', 'B', 'C', 'B', 'A', 'B', 'C', 'A', 'B', 'B']
```

```
#accuracy
r = 0
s = 0
for f, b in zip(pred, y_test):
    if(f == b):
        r += 1
    else:
        s+=1
print("Our accuracy",r/(r+s))
```

```
Our accuracy 0.8235294117647058
```

2-NAIVE BAYES (With Sklearn)

Here I look at the results using the sklearn naive bayes package.

```
n [28]: nb = GaussianNB()
        nb_model = nb.fit(x_train, y_train)
        nb_model
        # I built the naive bayes model here.

ut[28]: GaussianNB(priors=None, var_smoothing=1e-09)

n [33]: nb_model.predict(x_test)[0:119]
        #I'm performing the prediction process

ut[33]: array(['C', 'B', 'C', 'C', 'A', 'B', 'D', 'A', 'A', 'B', 'B', 'C', 'A',
               'A', 'B', 'C', 'C', 'B', 'B', 'C', 'A', 'C', 'B', 'D', 'A', 'A',
               'B', 'B', 'A', 'B', 'B', 'C', 'B', 'A', 'C', 'C', 'A', 'C', 'B',
               'C', 'B', 'B', 'B', 'B', 'C', 'D', 'B', 'B', 'B', 'B', 'B', 'C',
               'B', 'C', 'B', 'B', 'B', 'C', 'D', 'A', 'A', 'D', 'C', 'A', 'B',
               'B', 'B', 'C', 'C', 'B', 'B', 'B', 'A', 'B', 'C', 'B', 'C', 'C',
               'B', 'B', 'B', 'B', 'C', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'B',
               'B', 'D', 'C', 'C', 'A', 'A', 'D', 'C', 'A', 'A', 'C', 'A', 'A',
               'C', 'C', 'C', 'B', 'B', 'D', 'A', 'C', 'A', 'B', 'D', 'C', 'A',
               'B', 'A'], dtype='<U1')

n [34]: nb_model.predict_proba(x_test)[0:119]
        #I found the probability values.left row are probabilities of 0. right row are probabilities of 1.
```

```
]: array([[0.00000000e+00, 7.73472163e-06, 9.99992265e-01, 0.00000000e+00],
          [1.62644193e-05, 9.98828926e-01, 1.15480949e-03, 0.00000000e+00],
          [2.00560480e-17, 9.81398068e-02, 9.01860193e-01, 0.00000000e+00],
          [1.46808317e-13, 4.54762093e-01, 5.45237907e-01, 0.00000000e+00],
          [9.24035968e-01, 7.59640288e-02, 3.29664422e-09, 0.00000000e+00],
          [8.32433530e-06, 9.99991666e-01, 9.67798500e-09, 0.00000000e+00],
          [0.00000000e+00, 2.94495669e-48, 4.01267739e-32, 1.00000000e+00],
          [9.99988795e-01, 1.12045740e-05, 3.85426682e-20, 0.00000000e+00],
          [1.00000000e+00, 2.67552732e-12, 1.47109994e-37, 0.00000000e+00],
          [3.74384929e-18, 5.06581872e-01, 4.93418128e-01, 0.00000000e+00],
          [2.48253544e-03, 9.97517179e-01, 2.85264211e-07, 0.00000000e+00],
          [6.87812923e-28, 2.81784608e-02, 9.71821539e-01, 0.00000000e+00],
          [9.78840624e-01, 2.11592575e-02, 1.18156890e-07, 0.00000000e+00],
          [9.99999999e-01, 5.83865003e-10, 1.25218220e-33, 0.00000000e+00],
          [5.65184607e-04, 9.99427019e-01, 7.79685087e-06, 0.00000000e+00],
          [3.34511691e-32, 4.36938173e-08, 9.99999956e-01, 0.00000000e+00],
          [5.85002659e-38, 2.71807172e-07, 9.99999728e-01, 0.00000000e+00],
          [1.41223403e-05, 9.99985369e-01, 5.08211756e-07, 0.00000000e+00],
          [5.84651552e-04, 9.99408765e-01, 6.58296518e-06, 0.00000000e+00],
          [4.99338799e-15, 8.45688739e-02, 9.15431126e-01, 0.00000000e+00]
```

```
]: #If I want to calculate the test error:
   y_pred = nb_model.predict(x_test)
```

```
]: accuracy_score(y_test, y_pred)
   #I calculated the accuracy score
```

```
]: 0.8319327731092437
```

3-ALGORITHMS WITH K FOLD CROSS VALIDATION

I imported StratifiedKFold, which is written in the project guide. And I applied the KNN algorithm first.

```
34]: from sklearn.model_selection import StratifiedKFold
     Y= df2["letter_grade"]
     X = df2.drop(["letter_grade"], axis=1)
     #I imported libraries
```

I import some important packages and libraries to evaluate algorithms and their results. I assign the "letter_grade" attribute to the Y value, which I will use as the tag value. The other attributes remain as X.

1--KNN

Here the algorithm uses minkowski distance for n = 5 by default. I applied the algorithm with StratifiedKFold. As there are 5 iterations, I saved these results in a list and found the average of accuracy. Finally, I visualized each iteration.

35]:
```python
#I create an object to apply knn. k fold cross validation I keep every iteration in a list where it will be applied 5 times.
knn = KNeighborsClassifier()
accuracy_1 = []
skf1 = StratifiedKFold(n_splits = 5,random_state = None)
skf1.get_n_splits(X,Y)
for train_index, test_index in skf1.split(X,Y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

    knn_model = knn.fit(X_train, Y_train)
    prediction_1 =knn.predict(X_test)
    score_1 = accuracy_score(prediction_1,Y_test)
    accuracy_1.append(score_1)

print(accuracy_1)
```

```
[0.8271604938271605, 0.8875, 0.8987341772151899, 0.8974358974358975, 0.8701298701298701]
```

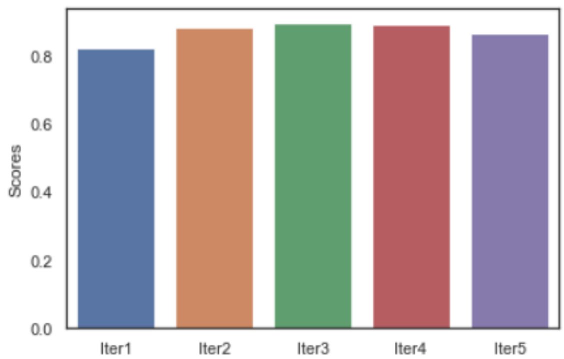
36]:
```python
#I average the results
total1=0
kmean1=0
for b in range(0,len(accuracy_1)):
    total1+=accuracy_1[b]
    kmean1=(total1)/len(accuracy_1)


print('Ortalama : ',kmean1)
```

```
Ortalama :  0.8761920877216236
```

59]:
```python
#I visualized the results of 5 iterations
scores = pd.DataFrame(accuracy_1,columns=['Scores'])

sns.set(style="white", rc={"lines.linewidth": 3})
sns.barplot(x=['Iter1','Iter2','Iter3','Iter4','Iter5'],y="Scores",data=scores)
plt.show()
sns.set()
```



I added the model tuning step to find the values that give the best results. Here I applied all the numbers from 1 to 11 as the value of n and found the value of the n which gives the best result. Finally, I calculated the success of this value.

[37]:
```
knn_params = {"n_neighbors": np.arange(1,11)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, knn_params, cv=5)
knn_cv.fit(X_train, Y_train)
```

```
C:\Users\ntatl\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when tes
t-set sizes are unequal.
  DeprecationWarning)
```

[37]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[38]:
```
print("En iyi skor:" + str(knn_cv.best_score_))
print("En iyi parametreler: " + str(knn_cv.best_params_))
```

```
En iyi skor:0.8836477987421384
En iyi parametreler: {'n_neighbors': 8}
```

[39]:
```
knn = KNeighborsClassifier(8)
knn_tuned = knn.fit(X_train, Y_train)
```

[40]:
```
knn_tuned.score(X_test, Y_test)
```

[40]:
```
0.8961038961038961
```

[41]:
```
y_pred = knn_tuned.predict(X_test)
```

[42]:
```
accuracy_score(Y_test, y_pred)
```

[42]:
```
0.8961038961038961
```

2--Naive Bayes

I have applied the naive bayes algorithm twice before. The difference this time is StratifiedKFold. As there are 5 iterations, I saved these results in a list and found the average of accuracy. Finally, I visualized each iteration.

3]:
```
#I create an object to apply bayes. k fold cross validation I keep every iteration in a list where it will be applied 5 times.
nb2 = GaussianNB()
accuracy_2 = []

skf2 = StratifiedKFold(n_splits = 5,random_state = None)
skf2.get_n_splits(X,Y)
for train_index, test_index in skf2.split(X,Y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

    nb2_model = nb2.fit(X_train, Y_train)
    prediction_2 =nb2.predict(X_test)
    score_2 = accuracy_score(prediction_2,Y_test)
    accuracy_2.append(score_2)

print(accuracy_2)
```

```
[0.8024691358024691, 0.7625, 0.8354430379746836, 0.7307692307692307, 0.8311688311688312]
```

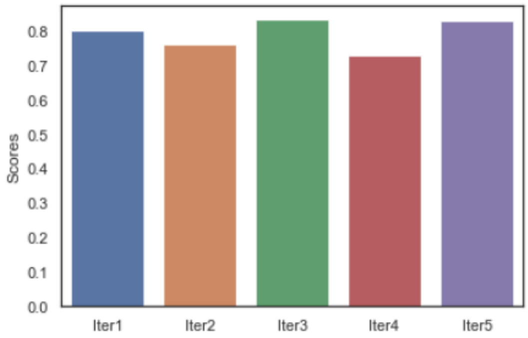
4]:
```
#I average the results
total2=0
kmean2=0
for a in range(0,len(accuracy_2)):
    total2+=accuracy_2[a]
    kmean2=(total2)/len(accuracy_2)


print('Ortalama : ',kmean2)
```

```
Ortalama :  0.792470047143043
```

0]:
```
#I visualized the results of 5 iterations
scores = pd.DataFrame(accuracy_2,columns=['Scores'])

sns.set(style="white", rc={"lines.linewidth": 3})
sns.barplot(x=['Iter1','Iter2','Iter3','Iter4','Iter5'],y="Scores",data=scores)
plt.show()
sns.set()
```



3—SVC

Algorithm C = 1 is applied by default. I applied the algorithm with StratifiedKFold. As there are 5 iterations, I saved these results in a list and found the average of accuracy. Finally, I visualized each iteration.

5]:
```
#I create an object to apply svm. k fold cross validation I keep every iteration in a list where it will be applied 5 times.
svm_model = SVC(kernel = "linear")

accuracy_3 = []

skf3 = StratifiedKFold(n_splits = 5,random_state = None)
skf3.get_n_splits(X,Y)
for train_index, test_index in skf3.split(X,Y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

    svm_model = SVC(kernel = "linear").fit(X_train, Y_train)
    # I did the model building process
    prediction_3 =svm_model.predict(X_test)
    score_3 = accuracy_score(prediction_3,Y_test)
    accuracy_3.append(score_3)

print(accuracy_3)
```

```
[0.9876543209876543, 1.0, 0.9873417721518988, 1.0, 1.0]
```

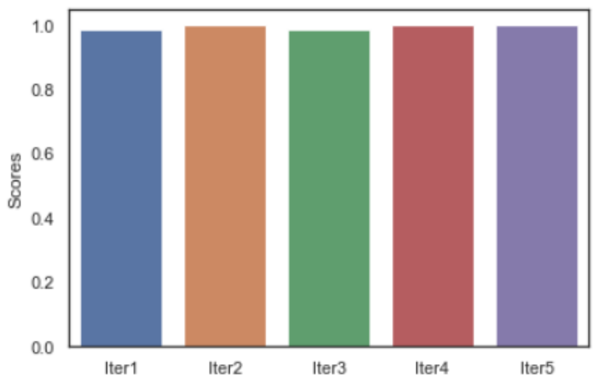
6]:
```
#I average the results
total3=0
kmean3=0
for d in range(0,len(accuracy_3)):
    total3+=accuracy_3[d]
    kmean3=(total3)/len(accuracy_3)


print('Ortalama : ',kmean3)
```

```
Ortalama :  0.9949992186279106
```

1]:
```
#I visualized the results of 5 iterations
scores = pd.DataFrame(accuracy_3,columns=['Scores'])

sns.set(style="white", rc={"lines.linewidth": 3})
sns.barplot(x=['Iter1','Iter2','Iter3','Iter4','Iter5'],y="Scores",data=scores)
plt.show()
sns.set()
```



I added the model tuning step to find the values that give the best results. Here I applied all the numbers from 1 to 10 as the C value and found the best C value. Finally, I calculated the success of this value.

```
[47]: svc_params = {"C": np.arange(1,10)}

      svc = SVC(kernel = "linear")

      svc_cv_model = GridSearchCV(svc,svc_params,
                                  cv = 5,
                                  n_jobs = -1,
                                  verbose = 2 )

      svc_cv_model.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    2.9s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    2.9s finished
```

```
[47]: GridSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                                 decision_function_shape='ovr', degree=3,
                                 gamma='auto_deprecated', kernel='linear',
                                 max_iter=-1, probability=False, random_state=None,
                                 shrinking=True, tol=0.001, verbose=False),
                   iid='warn', n_jobs=-1,
                   param_grid={'C': array([1, 2, 3, 4, 5, 6, 7, 8, 9])},
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                   scoring=None, verbose=2)
```

```
[48]: print("En iyi parametreler: " + str(svc_cv_model.best_params_))
```

```
En iyi parametreler: {'C': 1}
```

```
[49]: svc_tuned = SVC(kernel = "linear", C = 1).fit(X_train, Y_train)
```

```
[50]: y_pred = svc_tuned.predict(X_test)
      accuracy_score(Y_test, y_pred)
```

```
[50]: 1.0
```

4—Random Forest

I applied the algorithm with StratifiedKFold. As there are 5 iterations, I saved these results in a list and found the average of accuracy. Finally, I visualized each iteration.

```
#I create an object to apply knn. k fold cross validation I keep every iteration in a list where it will be applied 5 times.
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
accuracy_4 = []
```

```
skf4 = StratifiedKFold(n_splits = 5,random_state = None)
skf4.get_n_splits(X,Y)
for train_index, test_index in skf3.split(X,Y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

    rf_model = RandomForestClassifier().fit(X_train, Y_train)
    # I did the model building process
    prediction_4 =rf_model.predict(X_test)
    score_4 = accuracy_score(prediction_4,Y_test)
    accuracy_4.append(score_4)

print(accuracy_4)
```

```
[0.9382716049382716, 1.0, 0.9367088607594937, 0.9358974358974359, 0.8961038961038961]
```

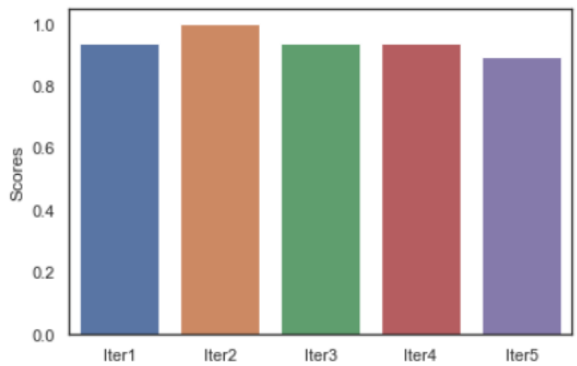
```
#I average the results
total4=0
kmean4=0
for h in range(0,len(accuracy_4)):
    total4+=accuracy_4[h]
    kmean4=(total4)/len(accuracy_4)


print('Ortalama : ',kmean4)
```

```
Ortalama :  0.9413963595398194
```

```
#I visualized the results of 5 iterations
scores = pd.DataFrame(accuracy_4,columns=['Scores'])

sns.set(style="white", rc={"lines.linewidth": 3})
sns.barplot(x=['Iter1','Iter2','Iter3','Iter4','Iter5'],y="Scores",data=scores)
plt.show()
sns.set()
```



I applied different states of values "max_depth", "max_features", "n_estimators", "min_samples_split". I found the best results of these values.

[54]:
```
rf_model
```

[54]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[55]:
```
rf_params = {"max_depth": [2,5,8,10],
            "max_features": [2,5,8],
            "n_estimators": [10,500,1000],
            "min_samples_split": [2,5,10]}
```

[56]:
```
rf_model = RandomForestClassifier()

rf_cv_model = GridSearchCV(rf_model,
                           rf_params,
                           cv = 5,
                           n_jobs = -1,
                           verbose = 2)
```

[57]:
```
rf_cv_model.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 169 tasks      | elapsed:    9.7s
[Parallel(n_jobs=-1)]: Done 372 tasks      | elapsed:   25.0s
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed:   39.4s finished
C:\Users\ntatl\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when tes
t-set sizes are unequal.
  DeprecationWarning)
```

57]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 5, 8, 10], 'max_features': [2, 5, 8],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [10, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

58]:
```
print("En iyi parametreler: " + str(rf_cv_model.best_params_))
```

```
En iyi parametreler: {'max_depth': 10, 'max_features': 8, 'min_samples_split': 5, 'n_estimators': 10}
```

59]:
```
rf_tuned = RandomForestClassifier(max_depth = 5,
                                  max_features = 8,
                                  min_samples_split = 5,
                                  n_estimators = 500)

rf_tuned.fit(X_train, Y_train)
```

59]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=5, max_features=8, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

60]:
```
y_pred = rf_tuned.predict(X_test)
accuracy_score(Y_test, y_pred)
```

60]:
```
0.987012987012987
```

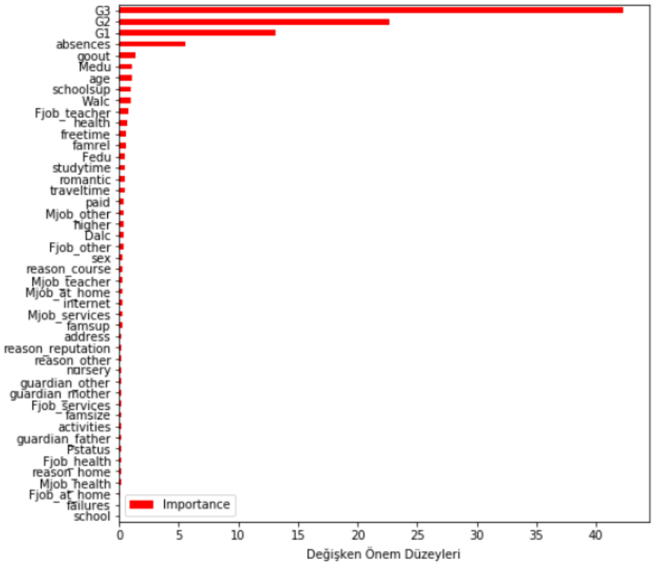
I created a chart to see which values are the most effective in the algorithm and how effective they are.

```
51]: Importance = pd.DataFrame({"Importance": rf_tuned.feature_importances_*100},
                              index = X_train.columns)
```

```
52]: Importance.sort_values(by = "Importance",
                           axis = 0,
                           ascending = True).plot(kind ="barh", color = "r",figsize=(8,8))

     plt.xlabel("Değişken Önem Düzeyleri")
```

```
52]: Text(0.5, 0, 'Değişken Önem Düzeyleri')
```



Evaluation of Results

Firstlı I compared the results I found first :

	KNN	Naive Bayes	SVM	Random Forest
Accuracy Rate	%87	%79	%99.9	%94

I compared the best parameters of the algorithms I used with StratifiedKFold with their accuracy rates.

74]:
```python
#In the model tuning section, I found the best results of the algorithms. Here I evaluated the best results of each algorithm.
modeller = [
    knn_tuned,
    nb2_model,
    svc_tuned,
    rf_tuned]

for model in modeller:
    isimler = model.__class__.__name__
    y_pred = model.predict(X_test)
    dogruluk = accuracy_score(Y_test, y_pred)
    print("-"*28)
    print(isimler + ":" )
    print("Accuracy: {:.4%}".format(dogruluk))
```

```
----------------------------
KNeighborsClassifier:
Accuracy: 89.6104%
----------------------------
GaussianNB:
Accuracy: 83.1169%
----------------------------
SVC:
Accuracy: 100.0000%
----------------------------
RandomForestClassifier:
Accuracy: 98.7013%
```

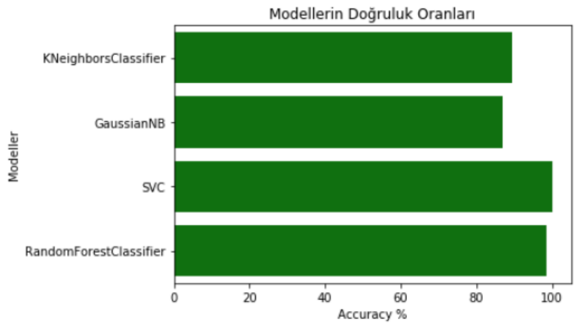
If I want to show this with a chart:

```python
#In the model tuning section, I found the best results of the algorithms, where I visualized the best results of each algorithm.
sonuc = []

sonuclar = pd.DataFrame(columns= ["Modeller","Accuracy"])

for model in modeller:
    isimler = model.__class__.__name__
    y_pred = model.predict(X_test)
    dogruluk = accuracy_score(Y_test, y_pred)
    sonuc = pd.DataFrame([[isimler, dogruluk*100]], columns= ["Modeller","Accuracy"])
    sonuclar = sonuclar.append(sonuc)


sns.barplot(x= 'Accuracy', y = 'Modeller', data=sonuclar, color="g")
plt.xlabel('Accuracy %')
plt.title('Modellerin Doğruluk Oranları');
```



If I want to compare Naive Bayes accuracy rates:

	Naive Bayes	Naive Bayes (sklearn)	Naive Bayes (StratifiedKFold)
Accuracy Rate	%82.3	%84	%83.1

The first version of the algorithms or the model tuned (most successful version) showed me that the most successful algorithm in this data set is the SVM algorithm.