```
import pandas as pd

df=pd.read_csv("/content/student-mat.csv",delimiter=';')
```

Data Description

```
Attributes for both student-mat.csv (Math course) and student-por.csv (Portuguese language course) da
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 â€“ 5th to 9th g
8 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th g
9 Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrati
10 Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrat
11 reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' pre
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
14 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 -
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)

# these grades are related with the course subject, Math or Portuguese:
```

```
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)
```

Converting String data to int using label encoder

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()

cols = ['address','famsize','school', 'sex','Mjob','Fjob','schoolsup','famsup','paid','Pstatu
```

Label Encoder

```
df[cols] = df[cols].apply(LabelEncoder().fit_transform)
```

Info of data after conversion

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      395 non-null    int64
 1   sex         395 non-null    int64
 2   age         395 non-null    int64
 3   address     395 non-null    int64
 4   famsize     395 non-null    int64
 5   Pstatus     395 non-null    int64
 6   Medu        395 non-null    int64
 7   Fedu        395 non-null    int64
 8   Mjob        395 non-null    int64
 9   Fjob        395 non-null    int64
 10  reason      395 non-null    int64
 11  guardian    395 non-null    int64
 12  traveltime  395 non-null    int64
 13  studytime   395 non-null    int64
 14  failures    395 non-null    int64
 15  schoolsup   395 non-null    int64
 16  famsup      395 non-null    int64
 17  paid        395 non-null    int64
 18  activities  395 non-null    int64
```

```
 19  nursery     395 non-null    int64
 20  higher      395 non-null    int64
 21  internet    395 non-null    int64
 22  romantic    395 non-null    int64
 23  famrel      395 non-null    int64
 24  freetime    395 non-null    int64
 25  goout       395 non-null    int64
 26  Dalc        395 non-null    int64
 27  Walc        395 non-null    int64
 28  health      395 non-null    int64
 29  absences    395 non-null    int64
 30  G1          395 non-null    int64
 31  G2          395 non-null    int64
 32  G3          395 non-null    int64
dtypes: int64(33)
memory usage: 102.0 KB
```

```
df
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	f
0	0	0	18	1	0	0	4	4	0	4	...	4	
1	0	0	17	1	0	1	1	1	0	2	...	5	
2	0	0	15	1	1	1	1	1	0	2	...	4	
3	0	0	15	1	0	1	4	2	1	3	...	3	
4	0	0	16	1	0	1	3	3	2	2	...	4	
...	...	...	...	...	...	...	...	...	...	...	...	...	
390	1	1	20	1	1	0	2	2	3	3	...	5	
391	1	1	17	1	1	1	3	1	3	3	...	2	
392	1	1	21	0	0	1	1	1	2	2	...	5	
393	1	1	18	0	1	1	3	2	3	2	...	4	
394	1	1	19	1	1	1	1	1	2	0	...	3	

395 rows × 33 columns

Rename function

```
df.rename(columns = {'famsize':'Fsize'}, inplace = True)
```

total null items

```
counting=df.isnull().count()
```

Total COlumn and its names

```
df.columns
```

```
Index(['school', 'sex', 'age', 'address', 'Fsize', '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')
```

Shape

```
df.shape
```

```
(395, 33)
```

Dropping all null item and making it inplace =True(premanentely)

```
df.dropna(inplace=True)
```

```
df
```

	school	sex	age	address	Fsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	0.0	0.0	0.428571	1.0	0.0	0.0	1.00	1.00	0.00	1.00	...	0.75
1	0.0	0.0	0.285714	1.0	0.0	1.0	0.25	0.25	0.00	0.50	...	1.00
2	0.0	0.0	0.000000	1.0	1.0	1.0	0.25	0.25	0.00	0.50	...	0.75
3	0.0	0.0	0.000000	1.0	0.0	1.0	1.00	0.50	0.25	0.75	...	0.50
4	0.0	0.0	0.142857	1.0	0.0	1.0	0.75	0.75	0.50	0.50	...	0.75
...	...	...	...	...	...	...	...	...	...	...	...	...
390	1.0	1.0	0.714286	1.0	1.0	0.0	0.50	0.50	0.75	0.75	...	1.00
391	1.0	1.0	0.285714	1.0	1.0	1.0	0.75	0.25	0.75	0.75	...	0.25
392	1.0	1.0	0.857143	0.0	0.0	1.0	0.25	0.25	0.50	0.50	...	1.00
393	1.0	1.0	0.428571	0.0	1.0	1.0	0.75	0.50	0.75	0.50	...	0.75
394	1.0	1.0	0.571429	1.0	1.0	1.0	0.25	0.25	0.50	0.00	...	0.50

395 rows × 33 columns

```
counting
```

```
school        395
sex           395
age           395
address       395
Fsize         395
Pstatus       395
Medu          395
Fedu          395
Mjob          395
Fjob          395
reason        395
guardian      395
traveltime    395
studytime     395
failures      395
schoolsup     395
famsup        395
paid          395
activities    395
nursery       395
higher        395
internet      395
romantic      395
famrel        395
freetime      395
goout         395
Dalc          395
Walc          395
health        395
absences      395
G1            395
G2            395
G3            395
dtype: int64
```

```
df.shape
```

```
(395, 33)
```

```
df.head(2)
```

	school	sex	age	address	Fsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel

```
df.tail(2)
```

	school	sex	age	address	Fsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
393	1.0	1.0	0.428571	0.0	1.0	1.0	0.75	0.50	0.75	0.5	...	0.75
394	1.0	1.0	0.571429	1.0	1.0	1.0	0.25	0.25	0.50	0.0	...	0.50

2 rows × 33 columns

Correlation

```
corr = df.corr()

# Displaying dataframe as an heatmap
# with diverging colourmap as coolwarm
corr.style.background_gradient(cmap ='binary')
```

	school	sex	age	address	Fsize	Pstatus	Medu	
school	1.000000	-0.012286	0.377610	-0.279797	0.064866	0.045923	-0.133333	-0
sex	-0.012286	1.000000	-0.028606	-0.028504	0.089862	0.023443	0.078228	0
age	0.377610	-0.028606	1.000000	-0.146722	0.037847	0.029598	-0.163658	-0
address	-0.279797	-0.028504	-0.146722	1.000000	0.072472	-0.042572	0.138804	0
Fsize	0.064866	0.089862	0.037847	0.072472	1.000000	-0.149612	-0.043068	-0
Pstatus	0.045923	0.023443	0.029598	-0.042572	-0.149612	1.000000	-0.123565	-0
Medu	-0.133333	0.078228	-0.163658	0.138804	-0.043068	-0.123565	1.000000	0
Fedu	-0.079807	0.034878	-0.163438	0.072178	-0.058879	-0.088730	0.623455	1
Mjob	-0.056670	0.191421	-0.069104	0.108818	0.071386	-0.054476	0.454805	0
Fjob	0.018974	0.084957	-0.035191	-0.001911	-0.084448	0.033892	0.157920	0
reason	-0.090010	-0.095867	-0.015337	-0.047891	-0.019212	-0.003553	0.117653	0
guardian	0.011109	-0.071343	0.290485	-0.044512	-0.002718	-0.108715	-0.023918	-0
traveltime	0.242308	0.059722	0.070641	-0.328096	0.063493	0.028265	-0.171639	-0
studytime	-0.090681	-0.306268	-0.004140	-0.020912	-0.073595	0.024294	0.064944	-0
failures	0.059804	0.044436	0.243665	-0.078578	-0.015769	-0.003339	-0.236680	-0
schoolsup	-0.139789	-0.138271	-0.251811	0.024712	-0.028642	-0.042238	-0.036029	0

conclusions from corr()

It is found that scoring good in anyone of the exam will result in good marks in all exams
exams:G1,G2,G3

Father education and mother education is slightly positively correlated

if the person drinks during weekend there are chances that he might drink everyday

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
df['school']=labelencoder.fit_transform(df['school'])
df.head(1)
```

	school	sex	age	address	Fsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	
0	0	0.0	0.428571	1.0	0.0	0.0	1.0	1.0	0.0	1.0	...	0.75	

1 rows × 33 columns

```
df.skew(axis = 0, skipna = True)
```

```
school        2.400519
sex           0.106886
age           0.466270
address      -1.337481
Fsize         0.936623
Pstatus      -2.607984
Medu         -0.318381
Fedu         -0.031672
Mjob         -0.335188
Fjob         -0.362303
reason        0.409568
guardian     -0.111546
traveltime    1.607029
studytime     0.632142
failures      2.387026
schoolsup     2.220534
famsup       -0.464291
paid          0.168315
activities   -0.035584
nursery      -1.466570
higher       -4.114829
internet     -1.791595
romantic      0.705766
famrel       -0.951882
freetime     -0.163351
goout         0.116502
Dalc          2.190762
Walc          0.611960
health       -0.494604
absences      3.671579
G1            0.240613
G2           -0.431645
G3           -0.732672
dtype: float64
```

```
#function which return return of min-max eq
def norm(item):
    return (item - item.min())/(item.max() - item.min())
#apply norm function to each item in dataframe
df = df.apply(norm)
df
```

	school	sex	age	address	Fsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	0.0	0.0	0.428571	1.0	0.0	0.0	1.00	1.00	0.00	1.00	...	0.75
1	0.0	0.0	0.285714	1.0	0.0	1.0	0.25	0.25	0.00	0.50	...	1.00
2	0.0	0.0	0.000000	1.0	1.0	1.0	0.25	0.25	0.00	0.50	...	0.75
3	0.0	0.0	0.000000	1.0	0.0	1.0	1.00	0.50	0.25	0.75	...	0.50
4	0.0	0.0	0.142857	1.0	0.0	1.0	0.75	0.75	0.50	0.50	...	0.75
...	...	...	...	...	...	...	...	...	...	...	...	...
390	1.0	1.0	0.714286	1.0	1.0	0.0	0.50	0.50	0.75	0.75	...	1.00

G1

School reports

grouby functions

grouping by schools

'GP' - Gabriel Pereira 'MS' - Mousinho da Silveira

G1 G2 G3 are the grade exam conducted at two schools and we are obtaining the mean median and min values for those

**COMPARING THE SCHOOLS and GRADES **

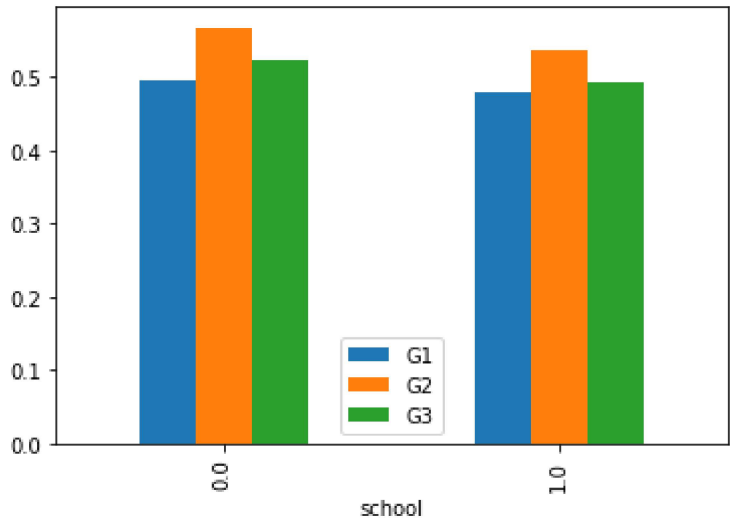
```
g1_school=df[['G1','G2','G3']].groupby(df['school']).mean()
g1_school_max=df[['G1','G2','G3']].groupby(df['school']).max()
g1_school_min=df[['G1','G2','G3']].groupby(df['school']).min()
```

```
print("Average:\n\n{} \n\nMAX\n\n{} \n\nMIN\n\n{}".format(g1_school,g1_school_max,g1_school_m
```

```
Average:

              G1        G2        G3
school
0.0     0.496239  0.567486  0.524499
1.0     0.479620  0.536613  0.492391

MAX

          G1        G2    G3
```

```
school
0.0     1.0  1.000000  1.00
1.0     1.0  0.947368  0.95

MIN

            G1        G2   G3
school
0.0     0.0000  0.000000  0.0
1.0     0.1875  0.263158  0.0
```

```
g1_school=df[['G1','G2','G3']].groupby(df['school']).mean().plot(kind='bar')
```



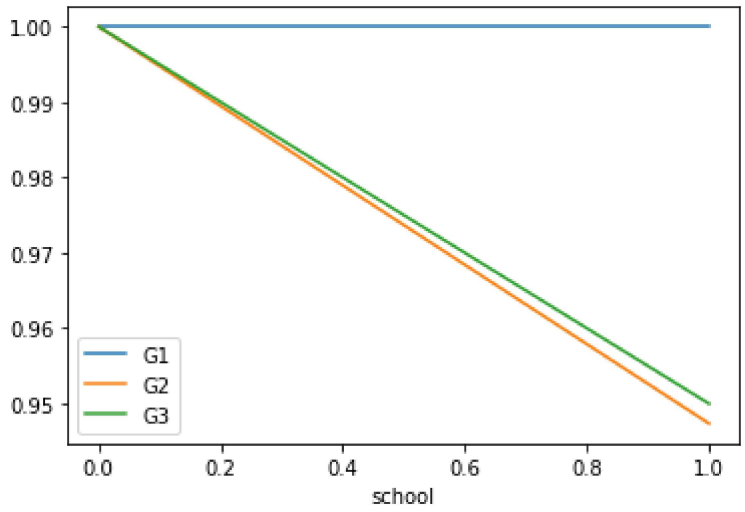
G2 every school has high marks

G1 is the lowest

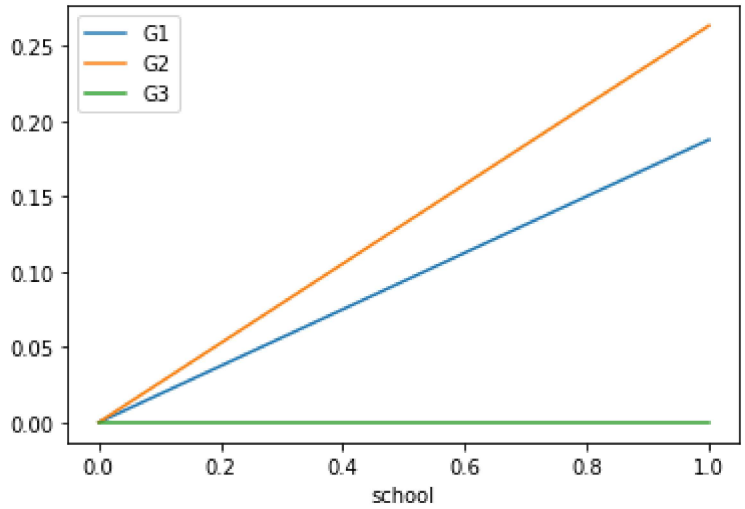
school 0 has high performance than school 1

```
g1_school=df[['G1','G2','G3']].groupby(df['school']).mean().plot(kind='line')
```

```
g1_school=df[['G1','G2','G3']].groupby(df['school']).max().plot()
```



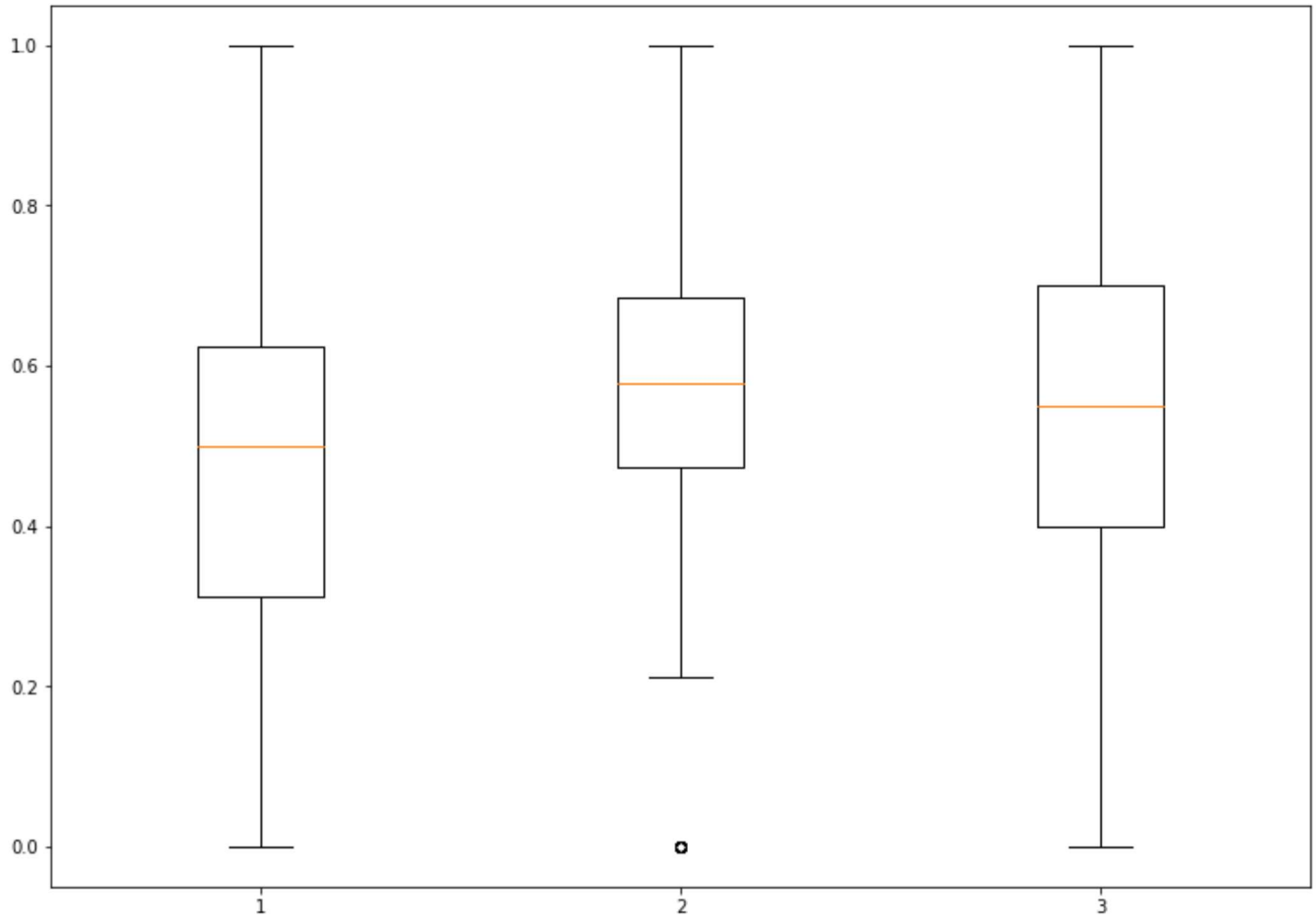

```
g1_school=df[['G1','G2','G3']].groupby(df['school']).min().plot()
```



Five number summary G3 has the most extremes

```
import matplotlib.pyplot as plt

g1_school=df['G1']
g2_school=df['G2']
g3_school=df['G3']
data = [g1_school,g2_school,g3_school]
# Creating axes instance
fig = plt.figure(figsize =(10, 7))
ax = fig.add_axes([0, 0, 1, 1])
# Creating plot
plt.boxplot(data)
```

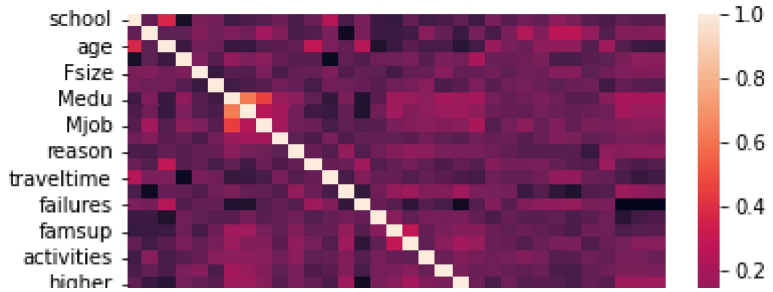
```
# show plot
plt.show()
```



*box plot and its five number summary *

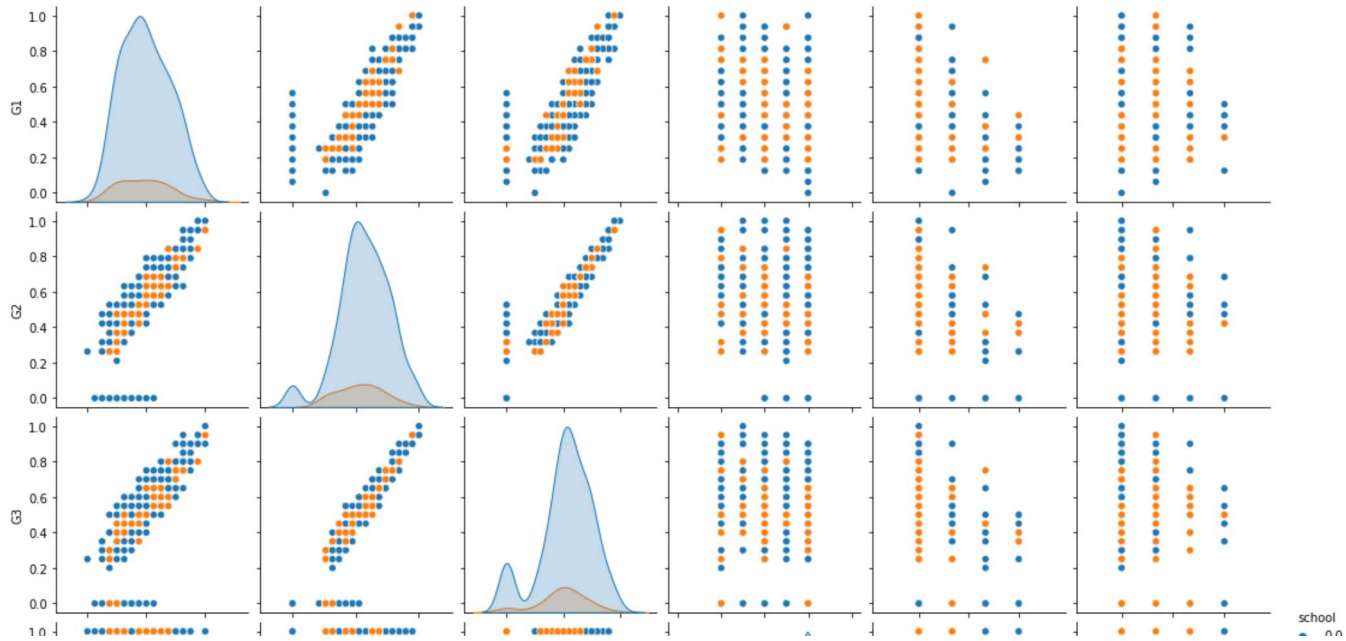
g1 - median is around 0.5 max is 1 min is 0 g2 - median is around 0.6 max is less than 1 greater than 0.9 min is 0.2 g3 - median is around 0.6 max is 1 min is 0

```
import seaborn as sb
dataplot=sb.heatmap(df.corr())
plt.show()
```




```
col=['school','G1','G2','G3','health','failures','traveltime']
sb.pairplot(df[col],hue="school")
```

```
<seaborn.axisgrid.PairGrid at 0x7f2baf255950>
```



inference :

> Indented block

---

when failures are less grades are more

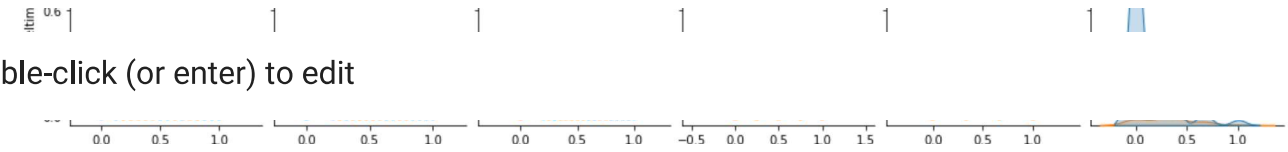
---

g1 ,g2 , g3 are related to each other

---

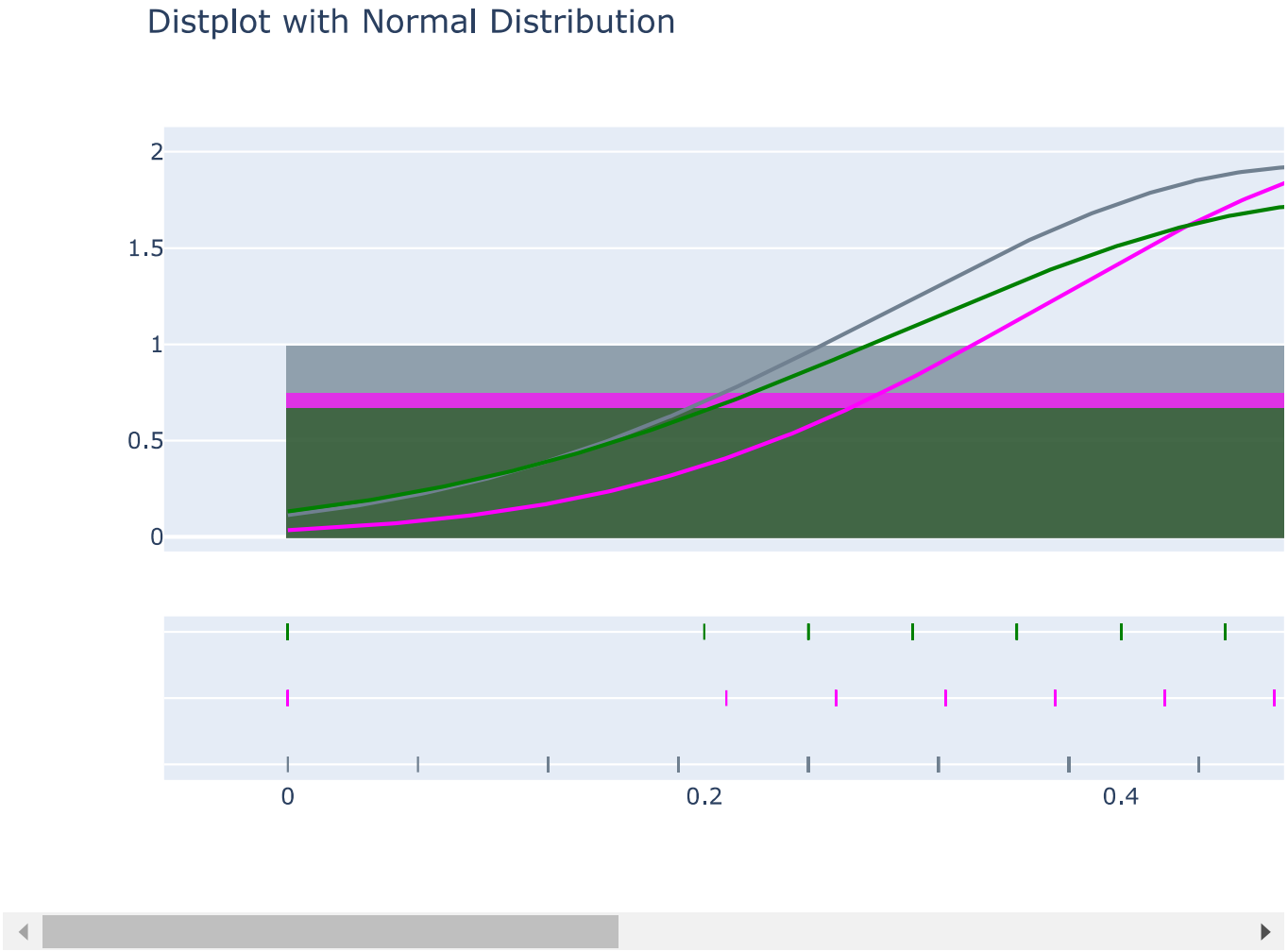
when travel time increases grades decreses but when its in limit then grades remain the same

---

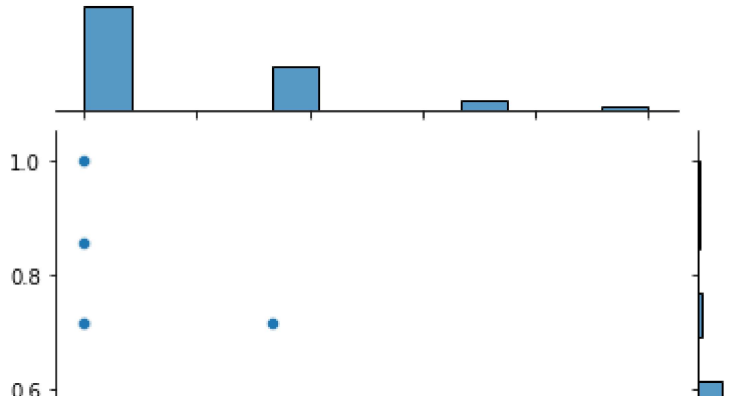


Double-click (or enter) to edit

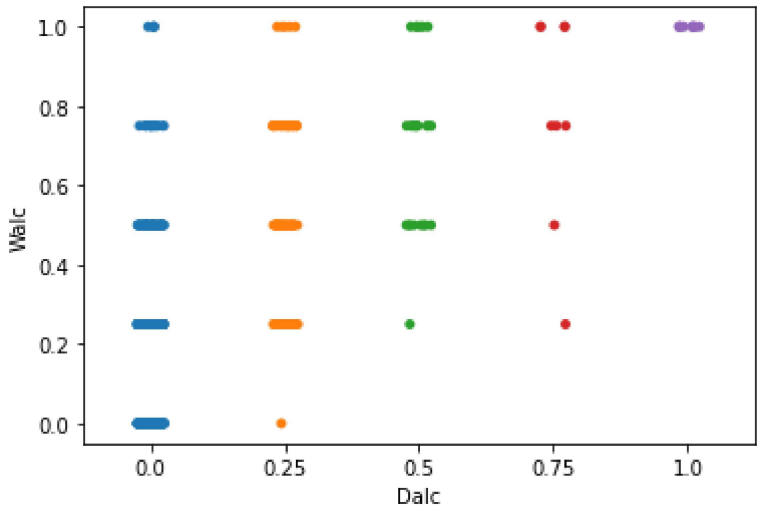
```
import plotly.figure_factory as ff
x1 = df['G1']
x2 = df['G2']
x3 = df['G3']

group_labels = ['g1', 'g2 ','g3']

colors = ['slategray', 'magenta','green']

# Create distplot with curve_type set to 'normal'
fig = ff.create_distplot([x1, x2,x3], group_labels, bin_size=.5,
                         curve_type='normal', # override default 'kde'
                         colors=colors)

# Add title
fig.update_layout(title_text='Distplot with Normal Distribution')
fig.show()
```




```
# importing required packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.jointplot(x = "traveltime", y = "age",kind = "scatter", data = df)
# show the plot
plt.show()
```




```
sb.stripplot(x="Dalc", y="Walc", data=df)

plt.show()
```




```
num_var = df['health']
num_var = pd.Series(num_var, name = "health")

# Plot histogram
sns.histplot(data = num_var, kde = True,stat = "probability")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2bafbd3a10>
```

```
fig = plt.figure(figsize=(10, 5))
g2=df['G2']
g1=df['G1']
# creating the bar plot
plt.barh(g2,g1, color='maroon')

plt.xlabel("G2")
plt.ylabel("G1")
plt.title("G1 VS g2")
plt.show()
```

KNN

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
df.info()
```

```
#Extracting Independent and dependent Variable
x= df.iloc[:, [30,32]].values
y= df.iloc[:, 0].values

# Splitting the dataset into training and test set.
X_train, X_test, Y_train, Y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
sc_X=StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.fit_transform(X_test)
```

```
import math
math.sqrt(len(Y_test))
```

When p =1, Minkowski = Manhattan

When p =2 , Minkowski = Euclidean

When p =3 , Minkowski = Chebyshev

```
classifier= KNeighborsClassifier(n_neighbors=3,p=2)
classifier.fit(X_train, Y_train)
```

```
classifier2= KNeighborsClassifier(n_neighbors=3,p=1)
classifier2.fit(X_train, Y_train)
```

```
classifier3= KNeighborsClassifier(n_neighbors=3,p=3)
classifier3.fit(X_train, Y_train)
```

```
print("{}".format(classifier))
print("{}".format(classifier2))
print("{}".format(classifier3))
```

```
#Predicting the test set result
Y_pred= classifier.predict(X_test)
Y_pred2= classifier2.predict(X_test)
Y_pred3= classifier3.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(Y_test, Y_pred)
cm2= confusion_matrix(Y_test, Y_pred2)
cm3= confusion_matrix(Y_test, Y_pred3)
```

```
cm3
```

```
cm2
```

```
cm
```

COnfusion matrix have 7% wrongly predicted 92% correctly predicted accuracy

```
acc=accuracy_score(Y_test, Y_pred)
acc
```

```
cl=classification_report(Y_test, Y_pred)


cl


from matplotlib.colors import ListedColormap
import numpy as nm
import matplotlib.pyplot as mtp
x_set, y_set = X_train, Y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shap
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('g score')
mtp.ylabel('')
mtp.legend()
mtp.show()



#Extracting Independent and dependent Variable
x1= df.iloc[:, [32,31]].values
y1= df.iloc[:, 0].values

# Splitting the dataset into training and test set.
X_train1, X_test1, Y_train1, Y_test1= train_test_split(x1, y1, test_size= 0.25, random_state=
sc_X=StandardScaler()
X_train1=sc_X.fit_transform(X_train1)
X_test1=sc_X.fit_transform(X_test1)


from sklearn.naive_bayes import GaussianNB
clas = GaussianNB()
clas.fit(X_train1, Y_train1)


    # Predicting the Test set results
y_pred1 = clas.predict(X_test1)


    # Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test1, y_pred1)
cm
```

```
    # Visualising the Training set results
import matplotlib.pyplot as mtp
import numpy as nm
from matplotlib.colors import ListedColormap
x_set, y_set = X_train1, Y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
               c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('g1 score')
mtp.ylabel('G score')
mtp.legend()
mtp.show()
```

```
    # Visualising the Training set results
import matplotlib.pyplot as mtp
import numpy as nm
from matplotlib.colors import ListedColormap
x_set, y_set = X_test, Y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
               c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('G1 score')
mtp.ylabel('G2 score')
mtp.legend()
mtp.show()
```

Clustering

kmeans clustering

```
df.head()
```

```
df.columns
```

```
df.info()
```

```
x = df.iloc[:, [30,31]].values
```

```
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 7):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 7), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()


#training the K-means model on a dataset
kmeans = KMeans(n_clusters=2, init='k-means++', random_state= 0)
y_predict= kmeans.fit_predict(x)
```

optimal no of clusters 3

```
y_predict
```

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluste
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'red', label = 'Cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yello
mtp.title('Clustering')
mtp.xlabel('G1 score')
mtp.ylabel('G2 Score')
mtp.legend()
mtp.show()
```

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=3, init='k-means++', random_state= 0)
y_predict= kmeans.fit_predict(x)
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluste
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yello
```

```
mtp.title('Clustering')
mtp.xlabel('G1 score')
mtp.ylabel('G2 Score')
mtp.legend()
mtp.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-f8714d13a3e1> in <module>
      1 #training the K-means model on a dataset
----> 2 kmeans = KMeans(n_clusters=3, init='k-means++', random_state= 0)
      3 y_predict= kmeans.fit_predict(x)
      4 #visulaizing the clusters
      5 mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue',
label = 'Cluster 1') #for first cluster

NameError: name 'KMeans' is not defined
```

SEARCH STACK OVERFLOW

As G1 increases then G2 also increases

green - need to improve student

blue - average student

red - good and best students

Hierarchical clustering