Notebook

March 10, 2023

1 Student Grade Analysis & Prediction

Objective: Prediction of the final grade of Portugese high school students

Data Set Information The data used is from a Portuguese secondary school. The data includes academic and personal characteristics of the students as well as final grades. The task is to predict the final grade from the student information. (Regression) Link to dataset

Citation: P. Cortez and A. Silva. Using Data Mining to Predict Secondary School Student Performance. In A. Brito and J. Teixeira Eds., Proceedings of 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008) pp. 5-12, Porto, Portugal, April, 2008, EUROSIS, ISBN 978-9077381-39-7.Web Link

1.1 Import Libraries

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

1.2 The Data

Let's start by reading in the student-mat.csv file into a pandas dataframe.

```
stud= pd.read_csv("gs://iitj_bigdataproject_pgd_dev/Pub/student-mat.csv/*.csv")␣
 ↪    # Read the dataset
```

```
print('Total number of students:',len(stud))
```

```
Total number of students: 395
```

```
stud['G3'].describe()
```

```
count    395.000000
mean      10.415190
std        4.581443
min        0.000000
```

```
25%       8.000000
50%      11.000000
75%      14.000000
max      20.000000
Name: G3, dtype: float64
```

```
stud.info()    # Information on dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 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
dtypes: int64(16), object(17)
memory usage: 101.9+ KB
```

```python
stud.columns     # Dataset Columns
```

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

```python
stud.describe()     # Dataset description
```

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

           famrel    freetime       goout        Dalc        Walc      health  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean     3.944304    3.235443    3.108861    1.481013    2.291139    3.554430
std      0.896659    0.998862    1.113278    0.890741    1.287897    1.390303
min      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
25%      4.000000    3.000000    2.000000    1.000000    1.000000    3.000000
50%      4.000000    3.000000    3.000000    1.000000    2.000000    4.000000
75%      5.000000    4.000000    4.000000    2.000000    3.000000    5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

         absences          G1          G2          G3
count  395.000000  395.000000  395.000000  395.000000
mean     5.708861   10.908861   10.713924   10.415190
std      8.003096    3.319195    3.761505    4.581443
min      0.000000    3.000000    0.000000    0.000000
25%      0.000000    8.000000    9.000000    8.000000
50%      4.000000   11.000000   11.000000   11.000000
75%      8.000000   13.000000   13.000000   14.000000
max     75.000000   19.000000   19.000000   20.000000
```

```python
stud.head()     # First 5 values of dataset
```

```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  …  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
```

```
2     GP   F   15        U      LE3        T      1      1  at_home      other  ...
3     GP   F   15        U      GT3        T      4      2   health   services  ...
4     GP   F   16        U      GT3        T      3      3    other      other  ...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

```
stud.tail()    # Last 5 values of dataset
```

```
    school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
390     MS   M   20       U     LE3       A     2     2  services  services
391     MS   M   17       U     LE3       T     3     1  services  services
392     MS   M   21       R     GT3       T     1     1     other     other
393     MS   M   18       R     LE3       T     3     2  services     other
394     MS   M   19       U     LE3       T     1     1     other   at_home

     ... famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
390  ...      5        5      4     4     5      4       11   9   9   9
391  ...      2        4      5     3     4      2        3  14  16  16
392  ...      5        5      3     3     3      3        3  10   8   7
393  ...      4        4      1     3     4      5        0  11  12  10
394  ...      3        2      3     3     3      5        5   8   9   9

[5 rows x 33 columns]
```

```
stud.isnull().any()    # To check any null values present in dataset
```

```
school        False
sex           False
age           False
address       False
famsize       False
Pstatus       False
Medu          False
Fedu          False
Mjob          False
Fjob          False
reason        False
guardian      False
traveltime    False
```

```
studytime     False
failures      False
schoolsup     False
famsup        False
paid          False
activities    False
nursery       False
higher        False
internet      False
romantic      False
famrel        False
freetime      False
goout         False
Dalc          False
Walc          False
health        False
absences      False
G1            False
G2            False
G3            False
dtype: bool
```

```
import cufflinks as cf
cf.go_offline()
```

```
stud.iplot()    # Plot for the all attributes
```

```
stud.iplot(kind='scatter',x='age',y='G3',mode='markers',size=8)    # Plot for␣
↪age vs G3
```

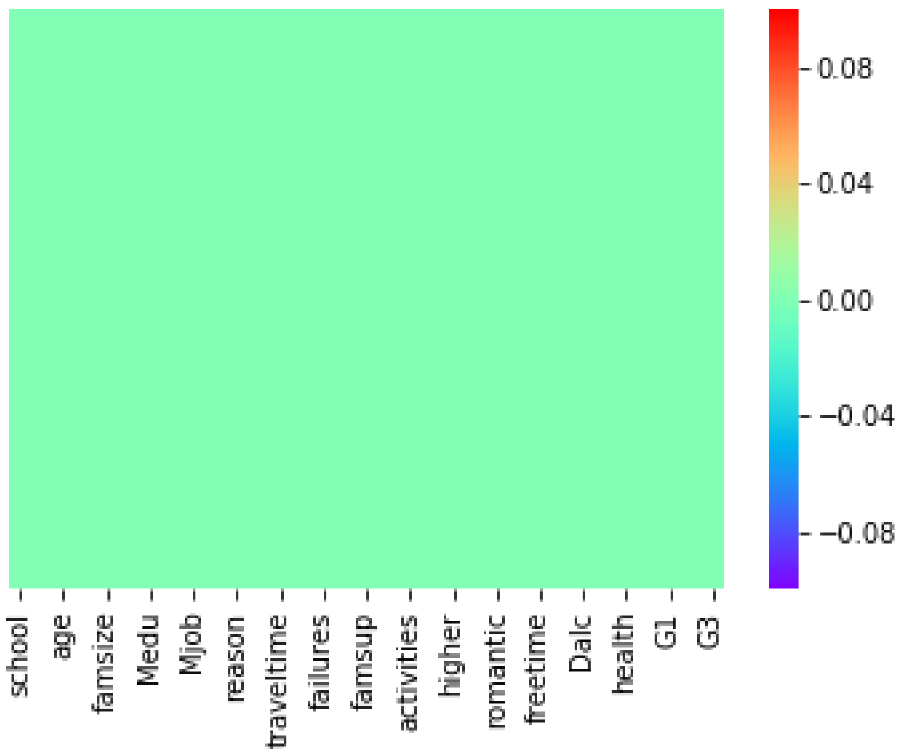
```
stud.iplot(kind='box')
```

```
stud['G3'].iplot(kind='hist',bins=100,color='blue')
```

2 Data Visualization

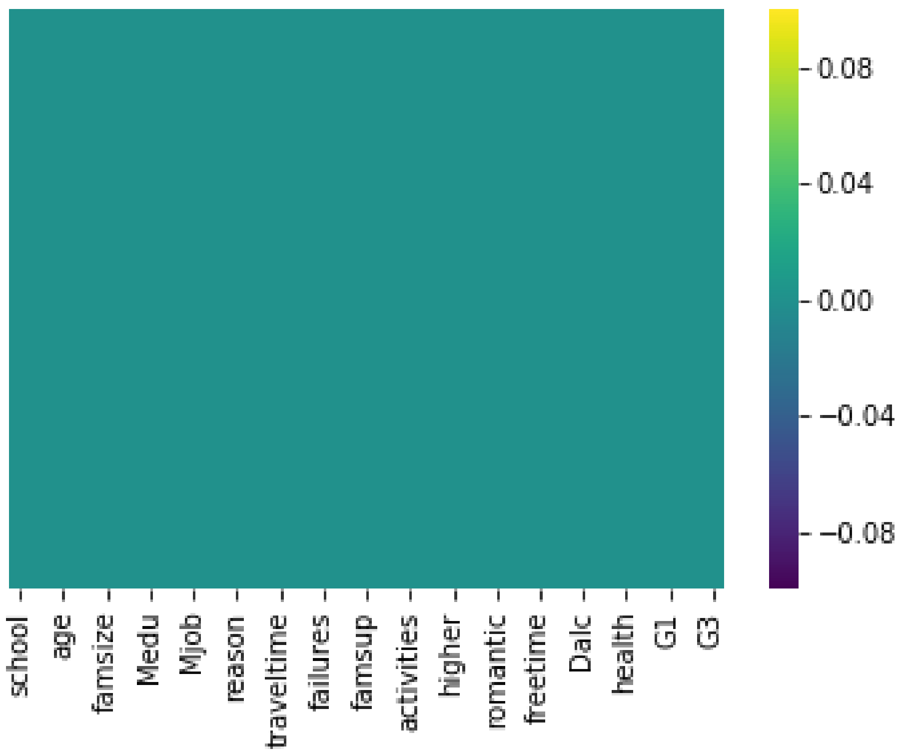
```
sns.heatmap(stud.isnull(),cmap="rainbow",yticklabels=False)    # To check any␣
↪null values present in dataset pictorially
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2041744c438>
```




```
sns.heatmap(stud.isnull(),cmap="viridis",yticklabels=False)     # Map color -
↪viridis
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20414b1c2b0>
```



- There are no null values in the given dataset

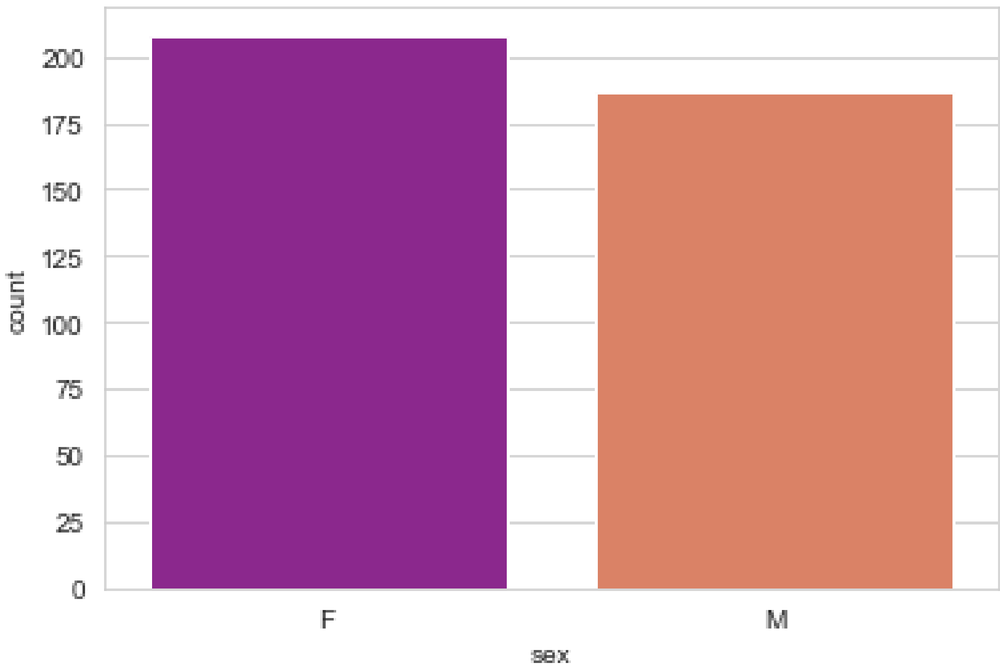
2.1 Student's Sex

```
f_stud = len(stud[stud['sex'] == 'F'])    # Number of female students
print('Number of female students:',f_stud)
m_stud = len(stud[stud['sex'] == 'M'])    # Number of male students
print('Number of male students:',m_stud)
```

```
Number of female students: 208
Number of male students: 187
```

```
sns.set_style('whitegrid')    # male & female student representaion on countplot
sns.countplot(x='sex',data=stud,palette='plasma')
```

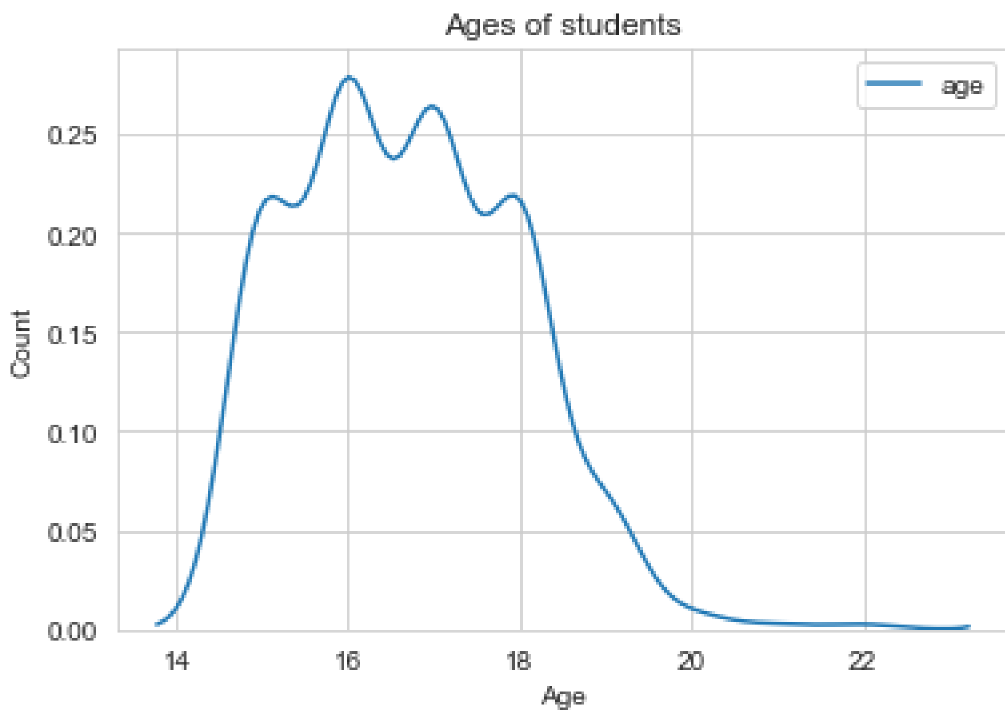
```
<matplotlib.axes._subplots.AxesSubplot at 0x204149de630>
```



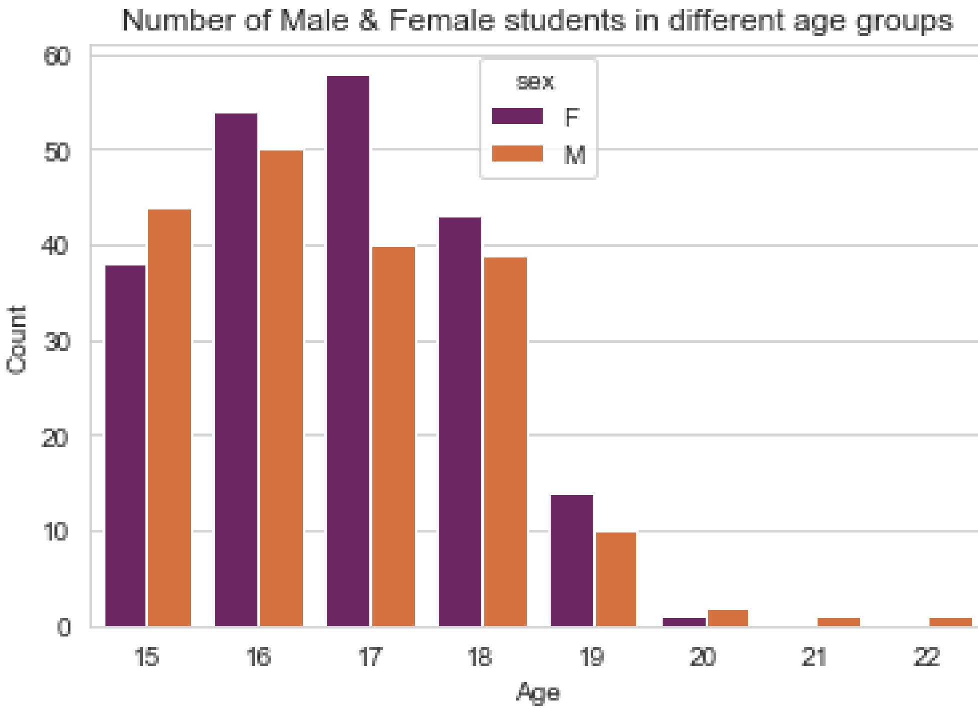
- The gender distribution is pretty even.

3 Age of Students

```
b = sns.kdeplot(stud['age'])    # Kernel Density Estimations
b.axes.set_title('Ages of students')
b.set_xlabel('Age')
b.set_ylabel('Count')
plt.show()
```




```
b = sns.countplot(x='age',hue='sex', data=stud, palette='inferno')
b.axes.set_title('Number of Male & Female students in different age groups')
b.set_xlabel("Age")
b.set_ylabel("Count")
plt.show()
```



- The student age seems to be ranging from 15-19, where gender distribution is pretty even in each age group.
- The age group above 19 may be outliers, year back students or droupouts.

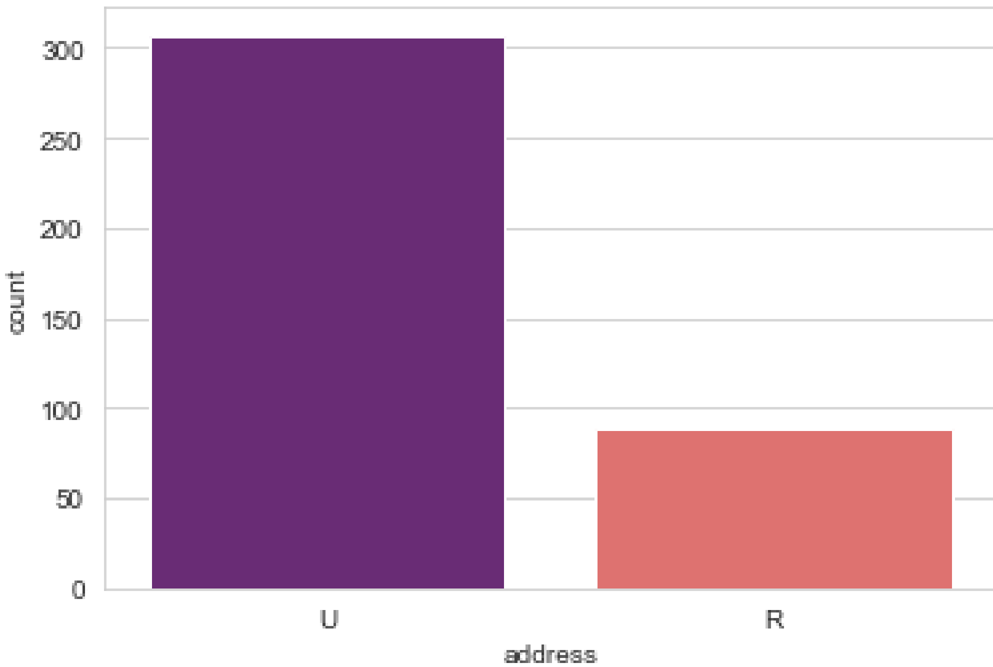
4 Students from Urban & Rural Areas

```
u_stud = len(stud[stud['address'] == 'U'])    # Number of urban areas students
print('Number of Urban students:',u_stud)
r_stud = len(stud[stud['address'] == 'R'])    # Number of rural areas students
print('Number of Rural students:',r_stud)
```

```
Number of Urban students: 307
Number of Rural students: 88
```

```
sns.set_style('whitegrid')
sns.countplot(x='address',data=stud,palette='magma')    # urban & rural␣
 ↪representaion on countplot
```

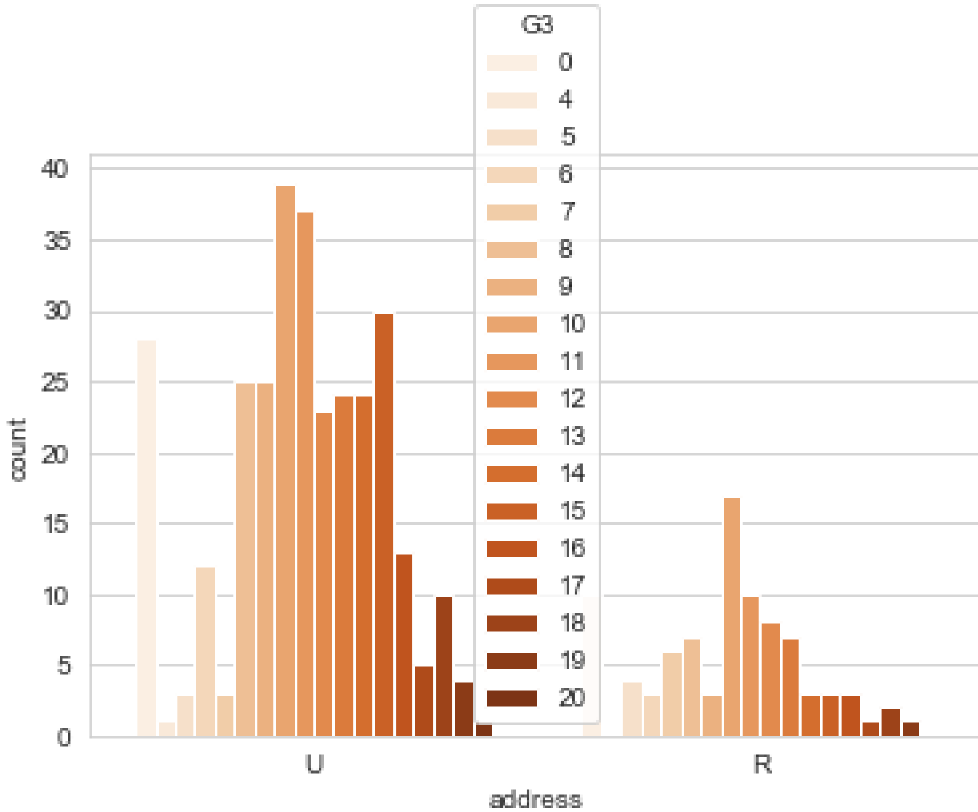
```
<matplotlib.axes._subplots.AxesSubplot at 0x2041470feb8>
```



- Approximately 77.72% students come from urban region and 22.28% from rural region.

```
sns.countplot(x='address',hue='G3',data=stud,palette='Oranges')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x204158e1710>
```

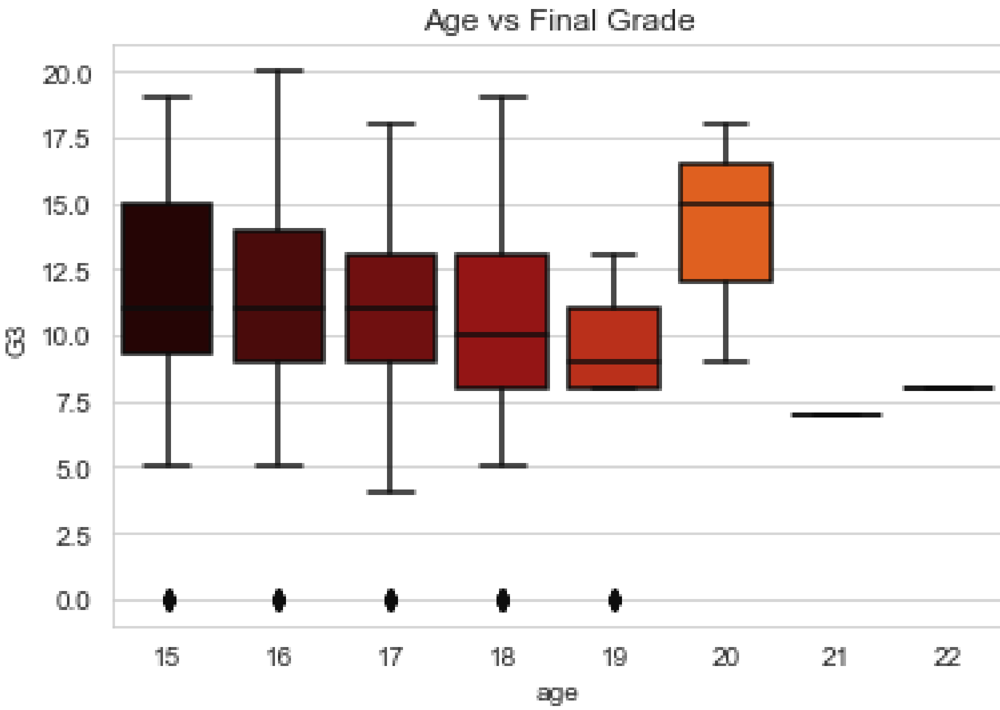


5 EDA - Exploratory Data Analysis

5.0.1 1. Does age affect final grade?

```
b= sns.boxplot(x='age', y='G3',data=stud,palette='gist_heat')
b.axes.set_title('Age vs Final Grade')
```

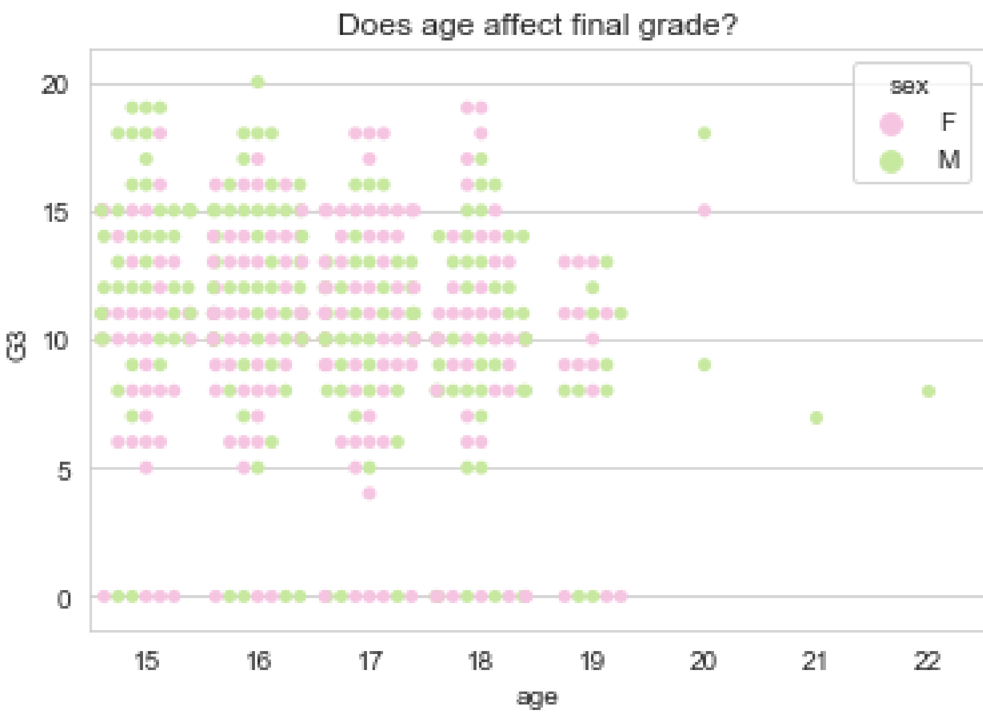
```
Text(0.5, 1.0, 'Age vs Final Grade')
```



- Plotting the distribution rather than statistics would help us better understand the data.
- The above plot shows that the median grades of the three age groups(15,16,17) are similar. Note the skewness of age group 19. (may be due to sample size). Age group 20 seems to score highest grades among all.

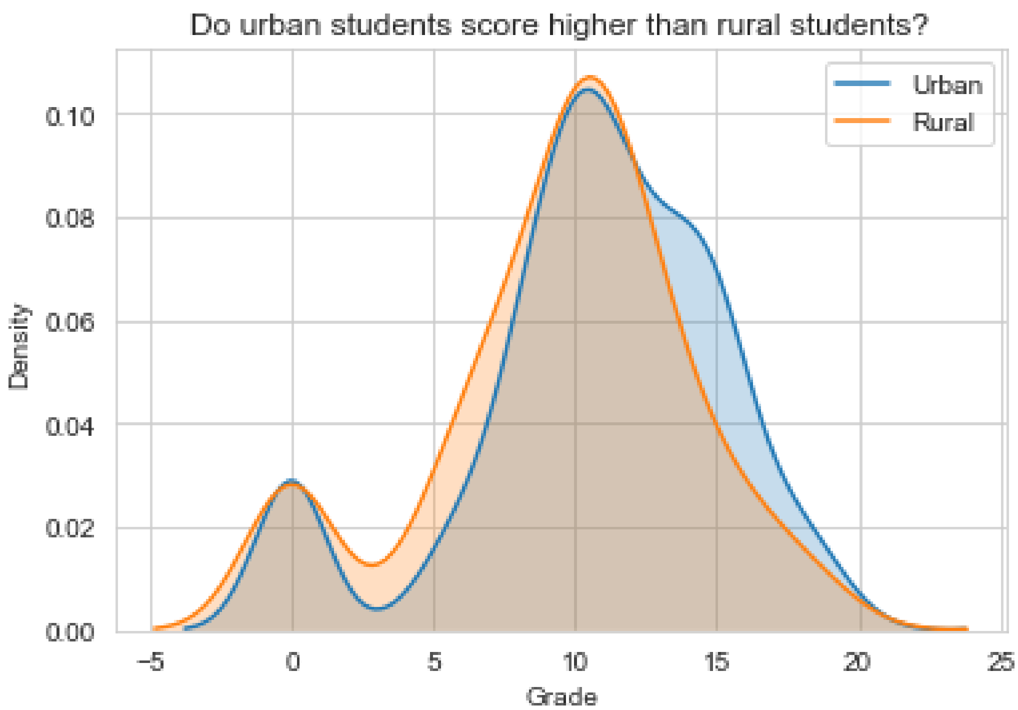
```
b = sns.swarmplot(x='age', y='G3',hue='sex', data=stud,palette='PiYG')
b.axes.set_title('Does age affect final grade?')
```

```
Text(0.5, 1.0, 'Does age affect final grade?')
```



5.1 2. Do urban students perform better than rural students?

```
# Grade distribution by address
sns.kdeplot(stud.loc[stud['address'] == 'U', 'G3'], label='Urban', shade = True)
sns.kdeplot(stud.loc[stud['address'] == 'R', 'G3'], label='Rural', shade = True)
plt.title('Do urban students score higher than rural students?')
plt.xlabel('Grade');
plt.ylabel('Density')
plt.show()
```



- The above graph clearly shows there is not much difference between the grades based on location.

```
stud.corr()['G3'].sort_values()
```

```
failures     -0.360415
age          -0.161579
goout        -0.132791
traveltime   -0.117142
health       -0.061335
Dalc         -0.054660
Walc         -0.051939
freetime      0.011307
absences      0.034247
famrel        0.051363
studytime     0.097820
Fedu          0.152457
Medu          0.217147
G1            0.801468
G2            0.904868
G3            1.000000
Name: G3, dtype: float64
```

5.2 Encoding categorical variables using LabelEncoder()

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
stud.iloc[:,0]=le.fit_transform(stud.iloc[:,0])
stud.iloc[:,1]=le.fit_transform(stud.iloc[:,1])
stud.iloc[:,3]=le.fit_transform(stud.iloc[:,3])
stud.iloc[:,4]=le.fit_transform(stud.iloc[:,4])
stud.iloc[:,5]=le.fit_transform(stud.iloc[:,5])
stud.iloc[:,8]=le.fit_transform(stud.iloc[:,8])
stud.iloc[:,9]=le.fit_transform(stud.iloc[:,9])
stud.iloc[:,10]=le.fit_transform(stud.iloc[:,10])
stud.iloc[:,11]=le.fit_transform(stud.iloc[:,11])
stud.iloc[:,15]=le.fit_transform(stud.iloc[:,15])
stud.iloc[:,16]=le.fit_transform(stud.iloc[:,16])
stud.iloc[:,17]=le.fit_transform(stud.iloc[:,17])
stud.iloc[:,18]=le.fit_transform(stud.iloc[:,18])
stud.iloc[:,19]=le.fit_transform(stud.iloc[:,19])
stud.iloc[:,20]=le.fit_transform(stud.iloc[:,20])
stud.iloc[:,21]=le.fit_transform(stud.iloc[:,21])
stud.iloc[:,22]=le.fit_transform(stud.iloc[:,22])
```

```
stud.head()
```

```
   school  sex  age  address  famsize  Pstatus  Medu  Fedu  Mjob  Fjob  …  \
0       0    0   18        1        0        0     4     4     0     4  …
1       0    0   17        1        0        1     1     1     0     2  …
2       0    0   15        1        1        1     1     1     0     2  …
3       0    0   15        1        0        1     4     2     1     3  …
4       0    0   16        1        0        1     3     3     2     2  …

   famrel  freetime  goout  Dalc  Walc  health  absences  G1  G2  G3
0       4         3      4     1     1       3         6   5   6   6
1       5         3      3     1     1       3         4   5   5   6
2       4         3      2     2     3       3        10   7   8  10
3       3         2      2     1     1       5         2  15  14  15
4       4         3      2     1     2       5         4   6  10  10

[5 rows x 33 columns]
```

```
stud.tail()
```

```
     school  sex  age  address  famsize  Pstatus  Medu  Fedu  Mjob  Fjob  …  \
390       1    1   20        1        1        0     2     2     3     3  …
391       1    1   17        1        1        1     3     1     3     3  …
392       1    1   21        0        0        1     1     1     2     2  …
393       1    1   18        0        1        1     3     2     3     2  …
```

```
394       1    1   19        1        1        1      1      1      2      0  …

     famrel  freetime  goout  Dalc  Walc  health  absences  G1  G2  G3
390       5         5      4     4     5       4        11   9   9   9
391       2         4      5     3     4       2         3  14  16  16
392       5         5      3     3     3       3         3  10   8   7
393       4         4      1     3     4       5         0  11  12  10
394       3         2      3     3     3       5         5   8   9   9

[5 rows x 33 columns]
```

```
stud.corr()['G3'].sort_values()    # Correlation wrt G3
```

```
failures     -0.360415
age          -0.161579
goout        -0.132791
romantic     -0.129970
traveltime   -0.117142
schoolsup    -0.082788
guardian     -0.070109
health       -0.061335
Pstatus      -0.058009
Dalc         -0.054660
Walc         -0.051939
school       -0.045017
famsup       -0.039157
freetime      0.011307
activities    0.016100
absences      0.034247
Fjob          0.042286
famrel        0.051363
nursery       0.051568
famsize       0.081407
studytime     0.097820
internet      0.098483
paid          0.101996
Mjob          0.102082
sex           0.103456
address       0.105756
reason        0.121994
Fedu          0.152457
higher        0.182465
Medu          0.217147
G1            0.801468
G2            0.904868
G3            1.000000
Name: G3, dtype: float64
```

```
# drop the school and grade columns
stud = stud.drop(['school', 'G1', 'G2'], axis='columns')
```

- Although G1 and G2 which are period grades of a student and are highly correlated to the final grade G3, we drop them. It is more difficult to predict G3 without G2 and G1, but such prediction is much more useful because we want to find other factors affect the grade.

```
# Find correlations with the Grade
most_correlated = stud.corr().abs()['G3'].sort_values(ascending=False)

# Maintain the top 8 most correlation features with Grade
most_correlated = most_correlated[:9]
most_correlated
```

```
G3          1.000000
failures    0.360415
Medu        0.217147
higher      0.182465
age         0.161579
Fedu        0.152457
goout       0.132791
romantic    0.129970
reason      0.121994
Name: G3, dtype: float64
```

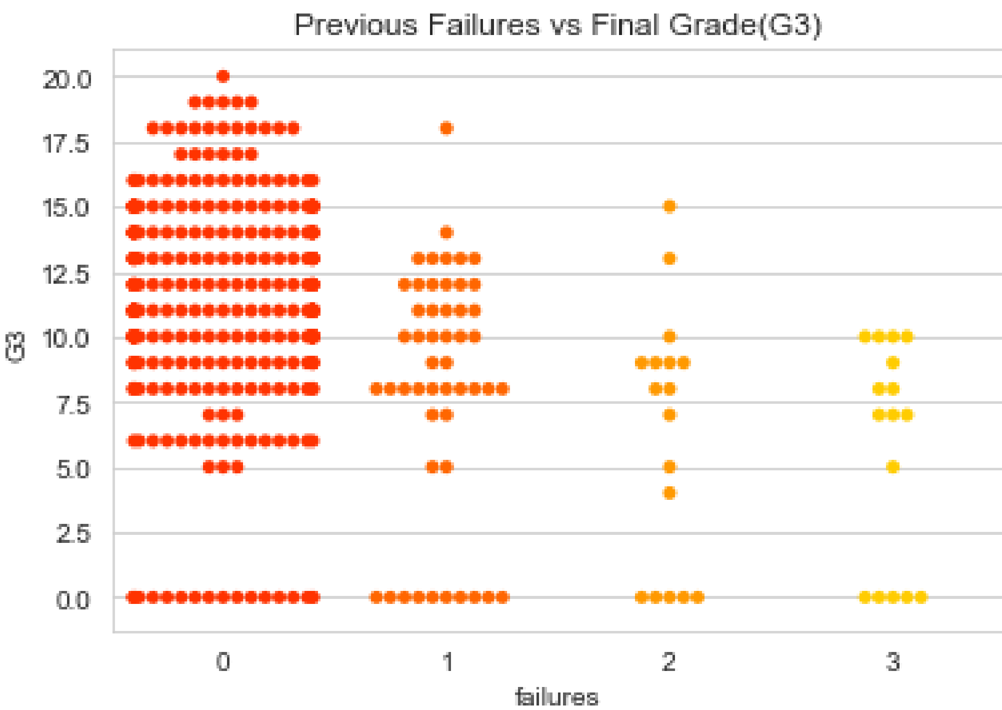
```
stud = stud.loc[:, most_correlated.index]
stud.head()
```

	G3	failures	Medu	higher	age	Fedu	goout	romantic	reason
0	6	0	4	1	18	4	4	0	0
1	6	0	1	1	17	1	3	0	0
2	10	3	1	1	15	1	2	0	2
3	15	0	4	1	15	2	2	1	1
4	10	0	3	1	16	3	2	0	1

5.2.1 Failure Attribute

```
b = sns.swarmplot(x=stud['failures'],y=stud['G3'],palette='autumn')
b.axes.set_title('Previous Failures vs Final Grade(G3)')
```

```
Text(0.5, 1.0, 'Previous Failures vs Final Grade(G3)')
```

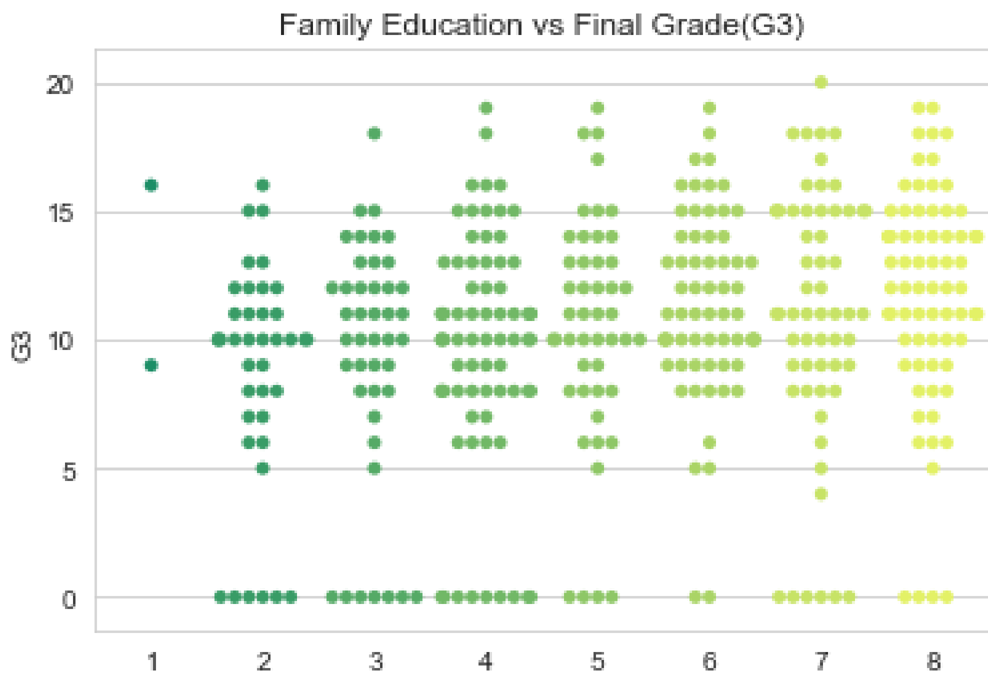


Observation : Student with less previous failures usually score higher

5.2.2 Family Education Attribute (Fedu + Medu)

```
fa_edu = stud['Fedu'] + stud['Medu']
b = sns.swarmplot(x=fa_edu,y=stud['G3'],palette='summer')
b.axes.set_title('Family Education vs Final Grade(G3)')
```

```
Text(0.5, 1.0, 'Family Education vs Final Grade(G3)')
```

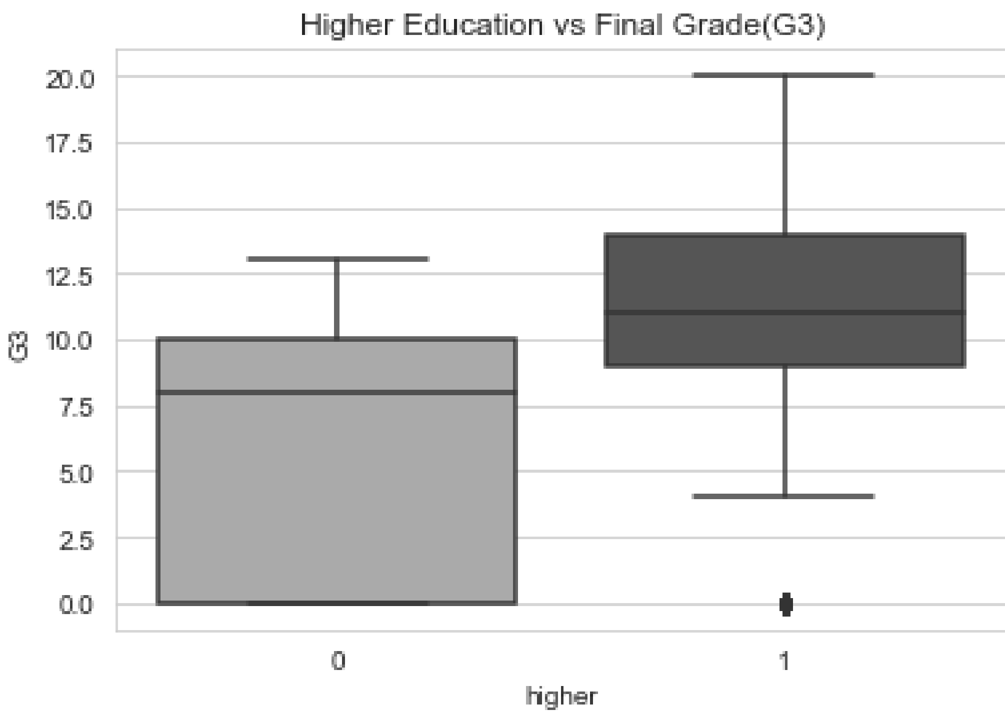


Observation : Educated families result in higher grades

5.2.3 Wish to go for Higher Education Attribute

```
b = sns.boxplot(x=stud['higher'],y=stud['G3'],palette='binary')
b.axes.set_title('Higher Education vs Final Grade(G3)')
```

```
Text(0.5, 1.0, 'Higher Education vs Final Grade(G3)')
```

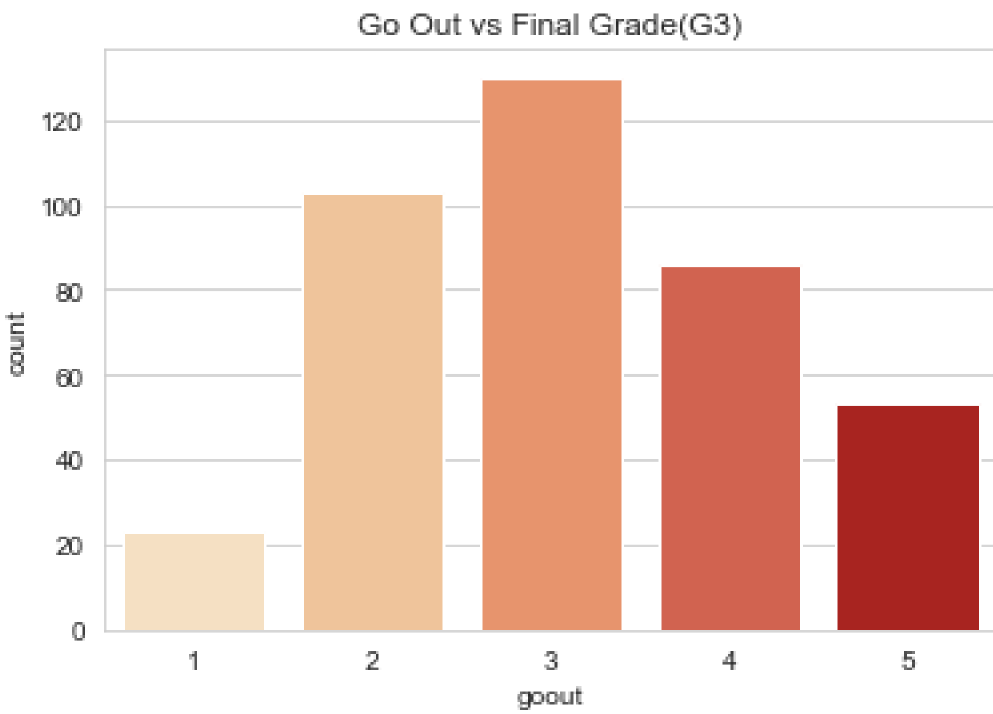


Observation : Students who wish to go for higher studies score more

5.3 Going Out with Friends Attribute

```
b = sns.countplot(x=stud['goout'],palette='OrRd')
b.axes.set_title('Go Out vs Final Grade(G3)')
```

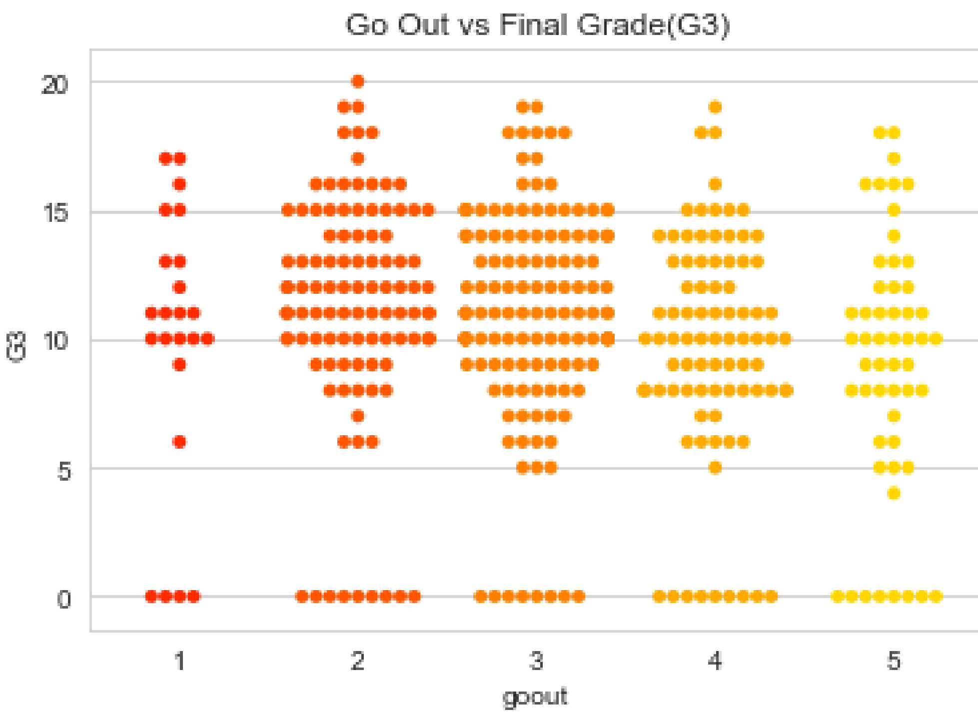
```
Text(0.5, 1.0, 'Go Out vs Final Grade(G3)')
```



Observation : The students have an average score when it comes to going out with friends.

```
b = sns.swarmplot(x=stud['goout'],y=stud['G3'],palette='autumn')
b.axes.set_title('Go Out vs Final Grade(G3)')
```

```
Text(0.5, 1.0, 'Go Out vs Final Grade(G3)')
```

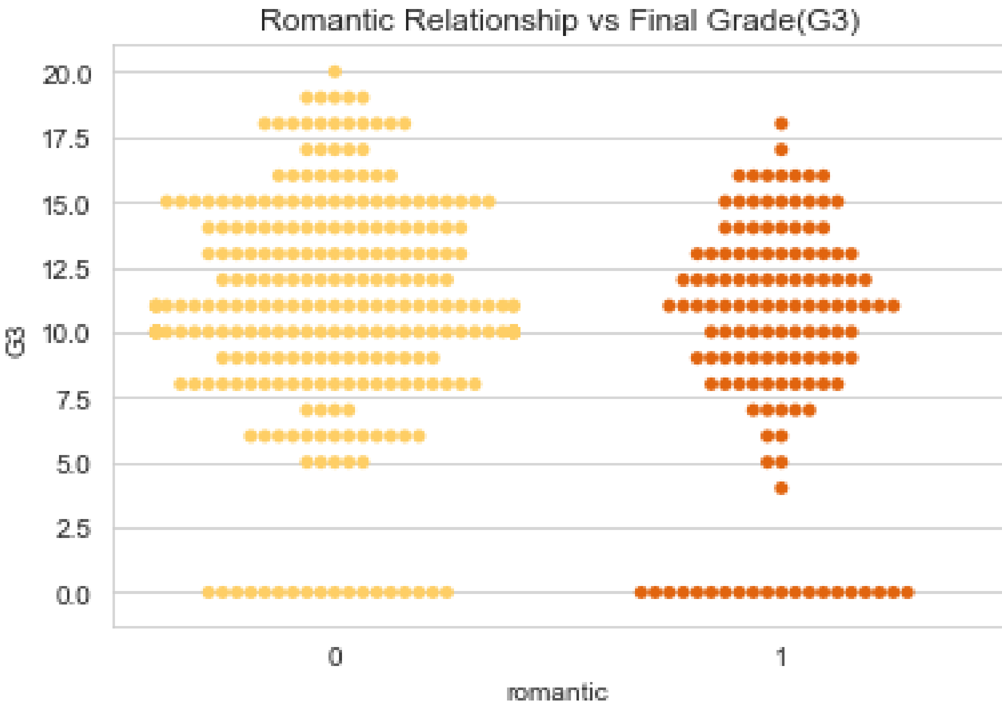


Observation : Students who go out a lot score less

5.3.1 Romantic relationship Attribute

```
b = sns.swarmplot(x=stud['romantic'],y=stud['G3'],palette='YlOrBr')
b.axes.set_title('Romantic Relationship vs Final Grade(G3)')
```

```
Text(0.5, 1.0, 'Romantic Relationship vs Final Grade(G3)')
```



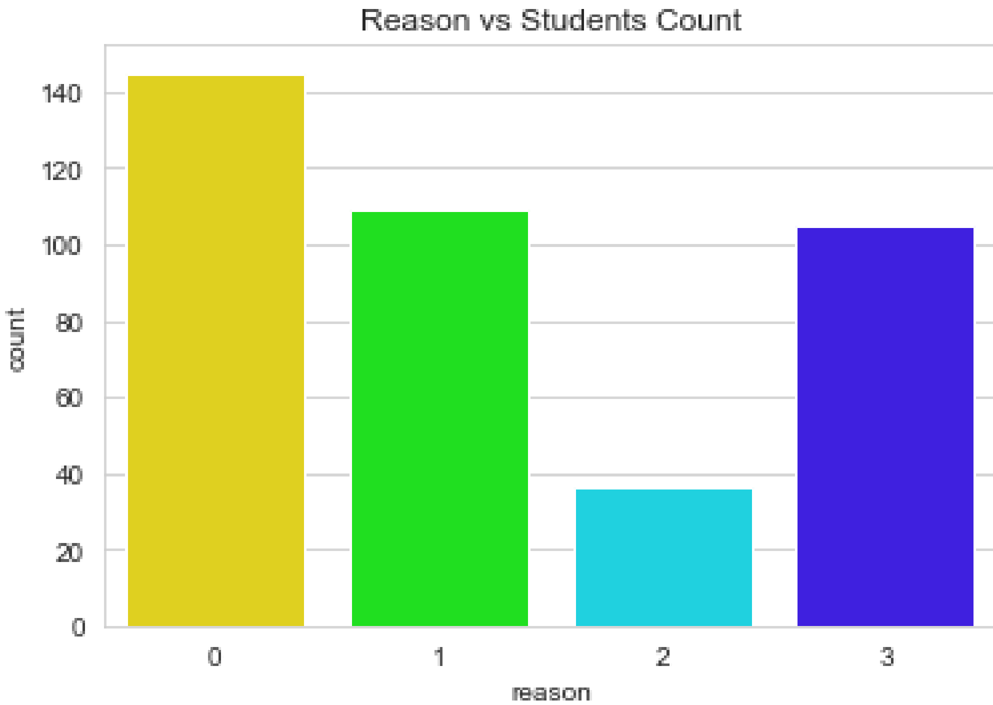
- Here romantic attribute with value 0 means no relationship and value with 1 means in relationship

Observation : Students with no romantic relationship score higher

5.3.2 Reason Attribute

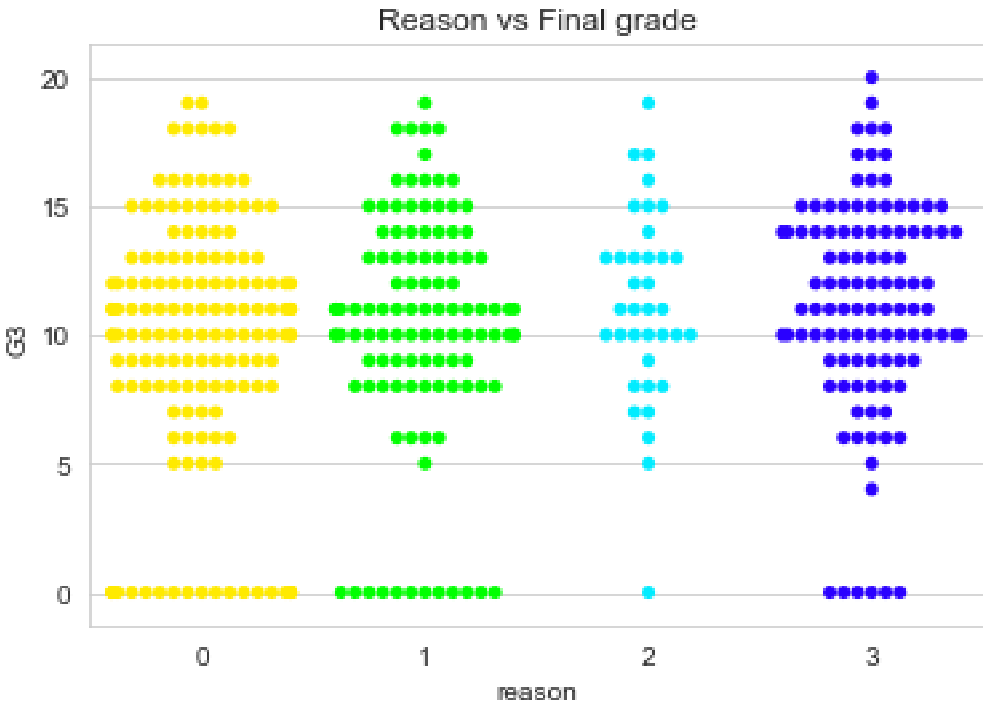
```
b = sns.countplot(x='reason',data=stud,palette='gist_rainbow')     # Reason to␣
↪choose this school
b.axes.set_title('Reason vs Students Count')
```

```
Text(0.5, 1.0, 'Reason vs Students Count')
```




```
b = sns.swarmplot(x='reason', y='G3', data=stud,palette='gist_rainbow')
b.axes.set_title('Reason vs Final grade')
```

```
Text(0.5, 1.0, 'Reason vs Final grade')
```



Observation : The students have an equally distributed average score when it comes to reason attribute.

6 Machine Learning Algorithms

```
# Standard ML Models for comparison
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR

# Splitting data into training/testing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪median_absolute_error

# Distributions
import scipy
```

```
# splitting the data into training and testing data (75% and 25%)
# we mention the random state to achieve the same split everytime we run the␣
↪code
X_train, X_test, y_train, y_test = train_test_split(stud, stud['G3'], test_size␣
↪= 0.25, random_state=42)
```

```
X_train.head()
```

```
     G3  failures  Medu  higher  age  Fedu  goout  romantic  reason
16   14         0     4       1   16     4      3         0       3
66   12         0     4       1   15     4      3         1       3
211  13         0     4       1   17     4      5         1       1
7     6         0     4       1   17     4      4         0       1
19   10         0     4       1   16     3      3         0       1
```

6.1 MAE - Mean Absolute Error & RMSE - Root Mean Square Error

```
# Calculate mae and rmse
def evaluate_predictions(predictions, true):
    mae = np.mean(abs(predictions - true))
    rmse = np.sqrt(np.mean((predictions - true) ** 2))

    return mae, rmse
```

```
# find the median
median_pred = X_train['G3'].median()

# create a list with all values as median
median_preds = [median_pred for _ in range(len(X_test))]

# store the true G3 values for passing into the function
true = X_test['G3']
```

```
# Display the naive baseline metrics
mb_mae, mb_rmse = evaluate_predictions(median_preds, true)
print('Median Baseline  MAE: {:.4f}'.format(mb_mae))
print('Median Baseline RMSE: {:.4f}'.format(mb_rmse))
```

```
Median Baseline  MAE: 3.7879
Median Baseline RMSE: 4.8252
```

```
# Evaluate several ml models by training on training set and testing on testing␣
↪set
def evaluate(X_train, X_test, y_train, y_test):
    # Names of models
    model_name_list = ['Linear Regression', 'ElasticNet Regression',
```

```
                      'Random Forest', 'Extra Trees', 'SVM',
                       'Gradient Boosted', 'Baseline']
    X_train = X_train.drop('G3', axis='columns')
    X_test = X_test.drop('G3', axis='columns')

    # Instantiate the models
    model1 = LinearRegression()
    model2 = ElasticNet(alpha=1.0, l1_ratio=0.5)
    model3 = RandomForestRegressor(n_estimators=100)
    model4 = ExtraTreesRegressor(n_estimators=100)
    model5 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')
    model6 = GradientBoostingRegressor(n_estimators=50)

    # Dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4, model5, model6]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        # Metrics
        mae = np.mean(abs(predictions - y_test))
        rmse = np.sqrt(np.mean((predictions - y_test) ** 2))

        # Insert results into the dataframe
        model_name = model_name_list[i]
        results.loc[model_name, :] = [mae, rmse]

    # Median Value Baseline Metrics
    baseline = np.median(y_train)
    baseline_mae = np.mean(abs(baseline - y_test))
    baseline_rmse = np.sqrt(np.mean((baseline - y_test) ** 2))

    results.loc['Baseline', :] = [baseline_mae, baseline_rmse]

    return results
```

```
results = evaluate(X_train, X_test, y_train, y_test)
results
```

	mae	rmse
Linear Regression	3.48512	4.4326
ElasticNet Regression	3.60805	4.57327
Random Forest	3.72601	4.61621
Extra Trees	3.7797	4.77882
SVM	3.54927	4.58147

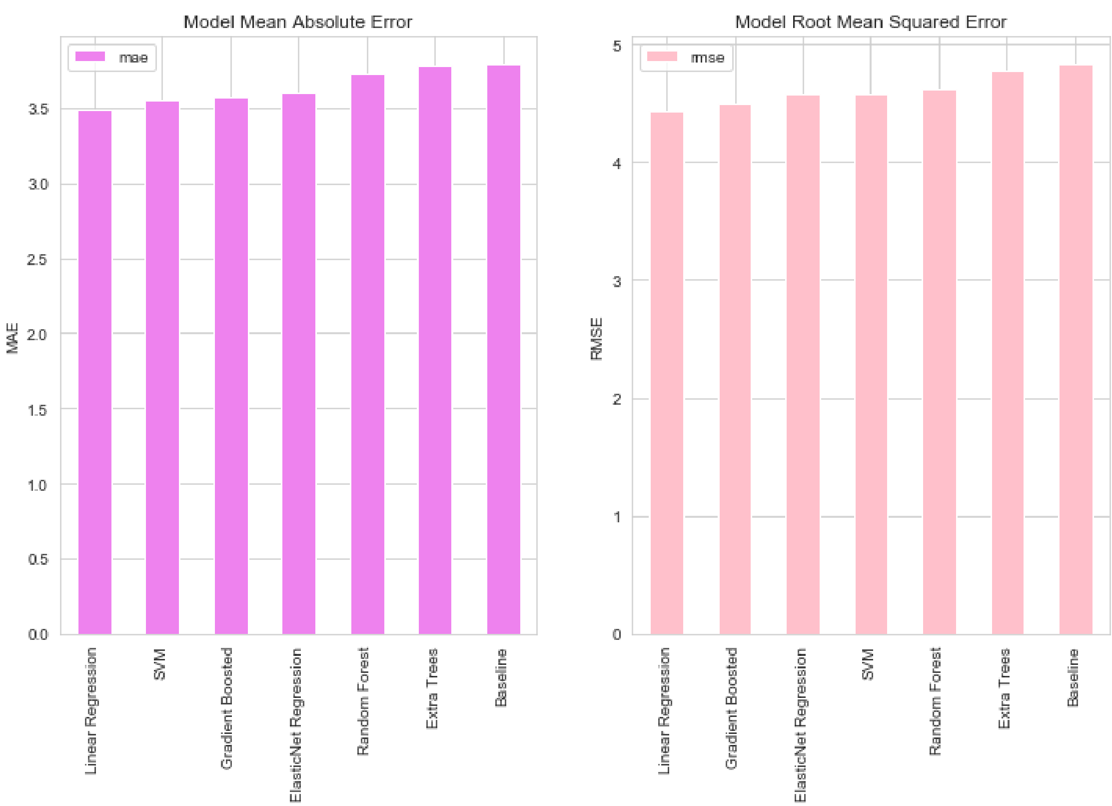
```
Gradient Boosted        3.57244  4.50059
Baseline                3.78788  4.82523
```

```
plt.figure(figsize=(12, 7))

# Root mean squared error
ax =  plt.subplot(1, 2, 1)
results.sort_values('mae', ascending = True).plot.bar(y = 'mae', color =␣
 ↪'violet', ax = ax)
plt.title('Model Mean Absolute Error')
plt.ylabel('MAE')

# Median absolute percentage error
ax = plt.subplot(1, 2, 2)
results.sort_values('rmse', ascending = True).plot.bar(y = 'rmse', color =␣
 ↪'pink', ax = ax)
plt.title('Model Root Mean Squared Error')
plt.ylabel('RMSE')

plt.show()
```



Conclusion: As we see both Model Mean Absolute Error & Model Root Mean Squared Error that the linear regression is performing the best in both cases