Lab: Student Grade Prediction

In this lab guide, we will try to predict the grades of students in mathematics class of two Portuguese schools. Specifically, we are to predict the final grades of the students. The data used here represents the student achievement in secondary education of two Portuguese schools. The data includes student grades, demographic, social, and school-related features, and it was collected by using school reports and questionnaires. A dataset is provided that describes the results of the mathematics class. The feature "G3" is the final year grade that needs to be predicted, while features "G1" and "G2" are the 1st and 2nd period grades. Since "G1" and "G2" are also grades, just at different periods, it will be much easier to predict "G3" since they are highly correlated. But it is more useful to predict "G3" without "G1" and "G2".

Feature information

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)

Grades:

1. G1 - first period grade (numeric: from 0 to 20)
2. G2 - second period grade (numeric: from 0 to 20)
3. G3 - final grade (numeric: from 0 to 20, output target)

Objective

In this lab, we will use the features provided to predict the final grades (G3) of the students for mathematics class. There are ways to make this into a classification problem, but we will be focusing on regression problem in this guide, predicting the raw scores of the students.

1. Data analysis

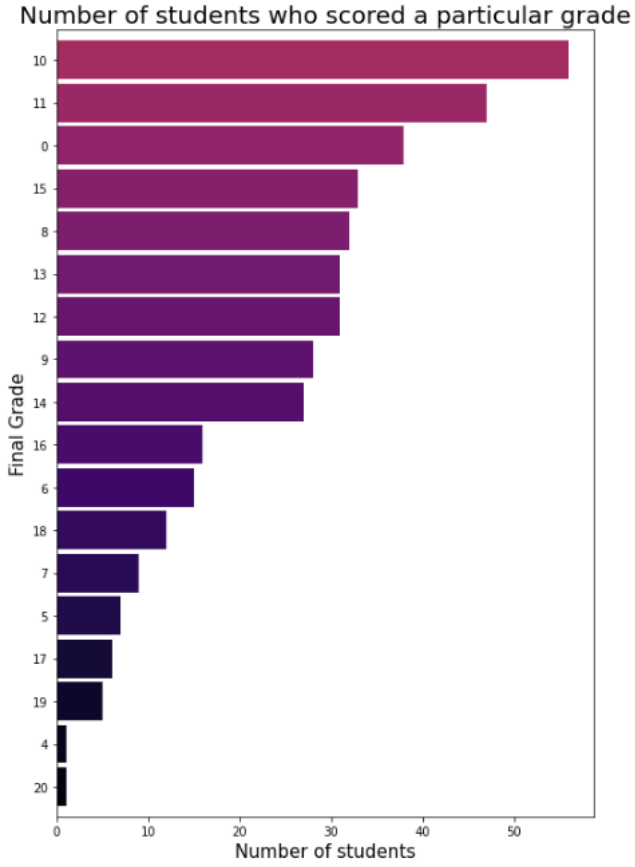
We first import the data: “student-mat.csv”. It should have the shape (395, 33): 395 rows and 33 columns.

```
student = pd.read_csv('student-mat.csv')
student.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

We will first take a look at the grades according to the number of students who scored them.

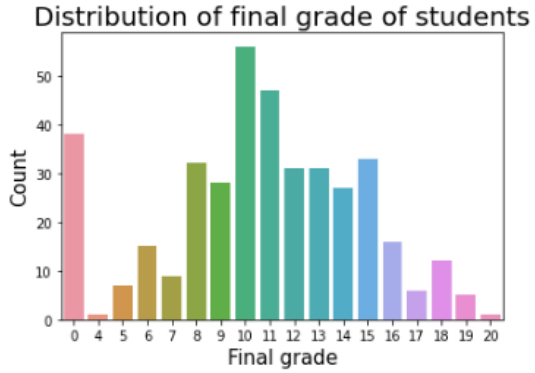
```
plt.subplots(figsize=(8, 12))
grade_counts = student['G3'].value_counts().sort_values().plot.barh(width=.9, color=sns.color_palette('inferno', 40))
grade_counts.axes.set_title('Number of students who scored a particular grade', fontsize=20)
grade_counts.set_xlabel('Number of students', fontsize=15)
grade_counts.set_ylabel('Final Grade', fontsize=15)
plt.show()
```



We can further elaborate this by graphing the final grade distribution.

```
b = sns.countplot(student['G3'])
b.axes.set_title('Distribution of final grade of students', fontsize=20)
b.set_xlabel('Final grade', fontsize=15)
b.set_ylabel('Count', fontsize=15)
plt.show()
```

```
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seaborn\_decorators.py
ble as a keyword arg: x. From version 0.12, the only valid positional argument will
out an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



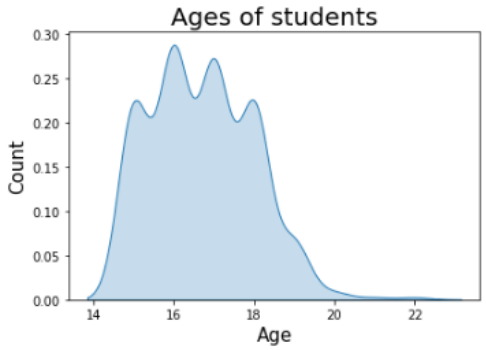
Apart from the high number of students scoring 0, the distribution is normal as expected, mainly clustering near the score of 10. Perhaps the number 0 is used in place of null. Or maybe the students who did not appear for the exam or were not allowed to sit for the exam due to some reasons are marked as 0. Upon checking whether there were any null values in the data, we can deduce that the score of 0 is not a substitute for null value.

Let's take a look at the gender feature and determine if it shows any interesting phenomena.

```
male_studs = len(student[student['sex'] == 'M'])
female_studs = len(student[student['sex'] == 'F'])

print('Number of male students: ', male_studs)
print('Number of female students: ', female_studs)
```

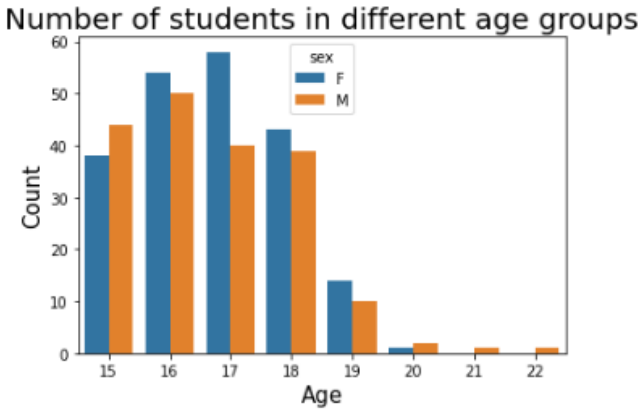
```
Number of male students:  187
Number of female students:  208
```

```
b = sns.kdeplot(student['age'], shade=True)
b.axes.set_title('Ages of students', fontsize=20)
b.set_xlabel('Age', fontsize=15)
b.set_ylabel('Count', fontsize=15)
plt.show()
```




```
b = sns.countplot('age', hue='sex', data=student)
b.axes.set_title('Number of students in different age groups', fontsize=20)
b.set_xlabel('Age', fontsize=15)
b.set_ylabel('Count', fontsize=15)
plt.show()
```

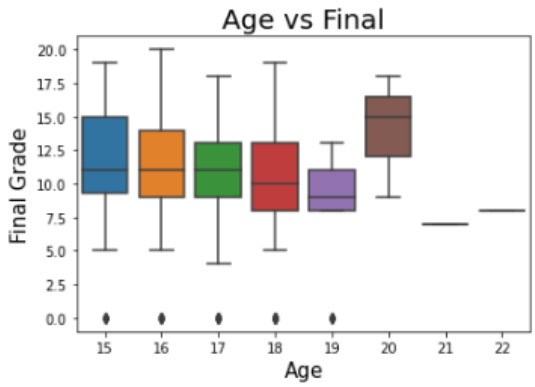
```
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seaborn\_decorato
g: Pass the following variable as a keyword arg: x. From version 0.12, the onl
ment will be `data`, and passing other arguments without an explicit keyword w
or misinterpretation.
  FutureWarning
```



The ages seem to be ranging from 15 - 19. The students above that age may not necessarily be outliers but students with year drops. Also, the gender distribution is pretty even. Therefore, there are no outliers even though there are some students with much higher ages than the rest.

Does age have anything to do with the final grade?

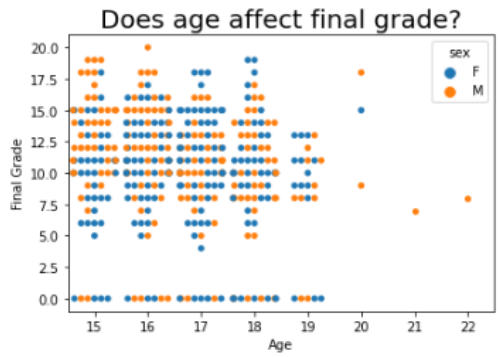
```
b = sns.boxplot(x='age', y='G3', data=student)
b.axes.set_title('Age vs Final', fontsize = 20)
b.set_xlabel('Age', fontsize = 15)
b.set_ylabel('Final Grade', fontsize = 15)
plt.show()
```



This standard boxplot showing the statistics does not really tell the entire story. So we instead plot the distribution rather than the statistics.

```
b = sns.swarmplot(x='age', y='G3',hue='sex', data=student)
b.axes.set_title('Does age affect final grade?', fontsize = 20)
b.set_xlabel('Age', fontsize = 10)
b.set_ylabel('Final Grade', fontsize = 10)
plt.show()
```

```
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seab
g: 13.4% of the points cannot be placed; you may want to decrease
pplot.
  warnings.warn(msg, UserWarning)
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seab
g: 21.2% of the points cannot be placed; you may want to decrease
pplot.
  warnings.warn(msg, UserWarning)
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seab
g: 21.4% of the points cannot be placed; you may want to decrease
pplot.
  warnings.warn(msg, UserWarning)
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seab
g: 17.1% of the points cannot be placed; you may want to decrease
pplot.
  warnings.warn(msg, UserWarning)
```

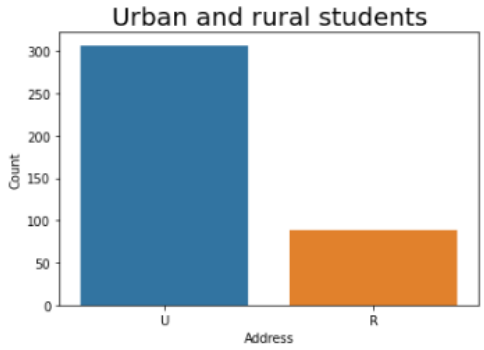


This plot shows more story. We see that age 20 has only 3 data points hence the inconsistency in statistics. Otherwise, there seems to be no clear relation of age or gender with final grade.

We can also see if there are differences in grades between urban and rural areas.

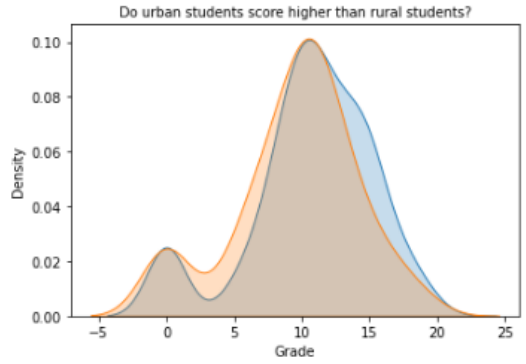
```
b = sns.countplot(student['address'])
b.axes.set_title('Urban and rural students', fontsize = 20)
b.set_xlabel('Address', fontsize = 10)
b.set_ylabel('Count', fontsize = 10)
plt.show()
```

```
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages
g: Pass the following variable as a keyword arg: x. From ver
ment will be `data`, and passing other arguments without an
or misinterpretation.
  FutureWarning
```



Most of the students are from urban areas, but do urban students perform better than rural students?

```
# Grade distribution by address
sns.kdeplot(student.loc[student['address'] == 'U', 'G3'], label='Urban', shade = True)
sns.kdeplot(student.loc[student['address'] == 'R', 'G3'], label='Rural', shade = True)
plt.title('Do urban students score higher than rural students?', fontsize = 10)
plt.xlabel('Grade', fontsize = 10);
plt.ylabel('Density', fontsize = 10)
plt.show()
```



Blue is urban and orange is rural. The graph shows that there is not much difference between the scores based on location. A slight discrepancy exists between the two distributions, but it does not warrant some certain explanations.

2. One-hot encoding

Categorical features in the dataset are in the form of strings. As we all know, machine learning cannot work well with string format data points, so we need to convert these features into numerical labels. One-hot encoding is one of the methods to convert some categorical feature into numerical features. It basically creates dummy variables representing the original features. Most of the features as object types are categorical features represented in strings. As an example, we can perform one-hot encoding on only the categorical features to determine how it will look after transformation.

```
category_df = student.select_dtypes(include=['object'])

dummy_df = pd.get_dummies(category_df)
dummy_df['G3'] = student['G3']
```

```
dummy_df.head()
```

	school_GP	school_MS	sex_F	sex_M	address_R	address_U	famsize_GT3	famsize_LE3	Pstatus_A	Pstatus_T	...	activities_yes	nursery_no	nursery_yes
0	1	0	1	0	0	1	1	0	1	0	...	0	0	1
1	1	0	1	0	0	1	1	0	0	1	...	0	1	0
2	1	0	1	0	0	1	0	1	0	1	...	0	0	1
3	1	0	1	0	0	1	1	0	0	1	...	1	0	1
4	1	0	1	0	0	1	1	0	0	1	...	0	0	1

As we can see, the categorical features are converted into 43 (44 features minus one target feature) new dummy variables, with each new feature representing one unique value of the categorical features.

We can apply one-hot coding and perform correlation analysis as a naive basis to select features to be used for regression. Although relying on correlation analysis only for feature selection is not the ideal way to do it, in this lab, we will determine if there are any noticeable differences in selecting features this way.

Furthermore, although G1 and G2 are periodic grades and are highly correlated to final grade G3 as they are past period grades, 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 affecting the grades.

```
student = student.drop(['school', 'G1', 'G2'], axis=1)

student = pd.get_dummies(student)
```

```
student.shape
```

```
(395, 55)
```

```
student.head()
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	activities_no	activities_yes	nursery_no	nursery_yes	higher_no	higher_
0	18	4	4	2	2	0	4	3	4	1	...	1	0	0	1	0	
1	17	1	1	1	2	0	5	3	3	1	...	1	0	1	0	0	
2	15	1	1	1	2	3	4	3	2	2	...	1	0	0	1	0	
3	15	4	2	1	3	0	3	2	2	1	...	0	1	0	1	0	
4	16	3	3	1	2	0	4	3	2	1	...	1	0	0	1	0	

Next, we will perform correlation analysis with "G3" as the basis, and we will select the top 8 most correlated features for regression prediction. Then, we will store the new dataframe with 8 features into “student_corr”.

```
most_correlated = student.corr().abs()['G3'].sort_values(ascending=False)

# will use the top 8 most correlated features with grade
most_correlated = most_correlated[:9]
most_correlated
```

```
G3              1.000000
failures        0.360415
Medu            0.217147
higher_yes      0.182465
higher_no       0.182465
age             0.161579
Fedu            0.152457
goout           0.132791
romantic_no     0.129970
Name: G3, dtype: float64
```

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

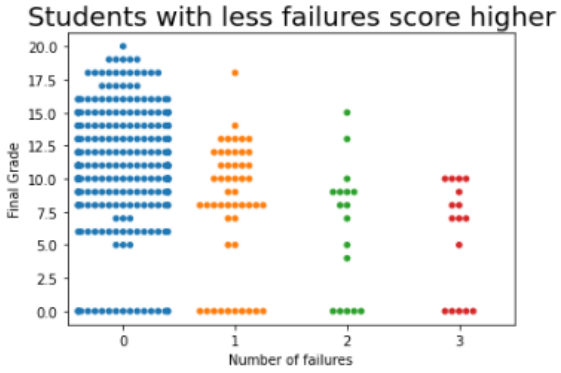
	G3	failures	Medu	higher_yes	higher_no	age	Fedu	goout	romantic_no
0	6	0	4	1	0	18	4	4	1
1	6	0	1	1	0	17	1	3	1
2	10	3	1	1	0	15	1	2	1
3	15	0	4	1	0	15	2	2	0
4	10	0	3	1	0	16	3	2	1

3. Analysis of the most correlated features

We will perform further analyses of these chosen variables. First, we would like to find out if students with less previous failures usually score higher.

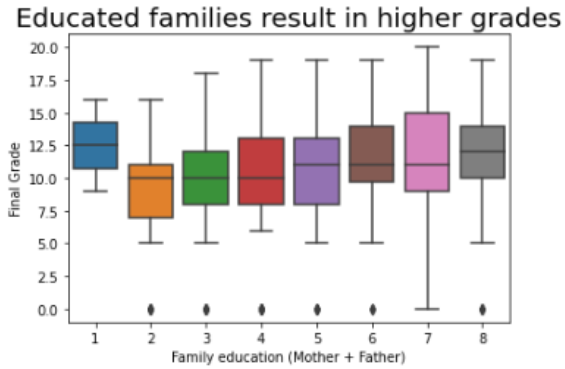
```
b = sns.swarmplot(x=student_corr['failures'],y=student_corr['G3'])
b.axes.set_title('Students with less failures score higher', fontsize = 20)
b.set_xlabel('Number of failures', fontsize = 10)
b.set_ylabel('Final Grade', fontsize = 10)
plt.show()
```

```
C:\Users\Brian Jung\anaconda3\envs\python7\lib\site-packages\seaborn\categor
g: 44.9% of the points cannot be placed; you may want to decrease the size o
pplot.
  warnings.warn(msg, UserWarning)
```

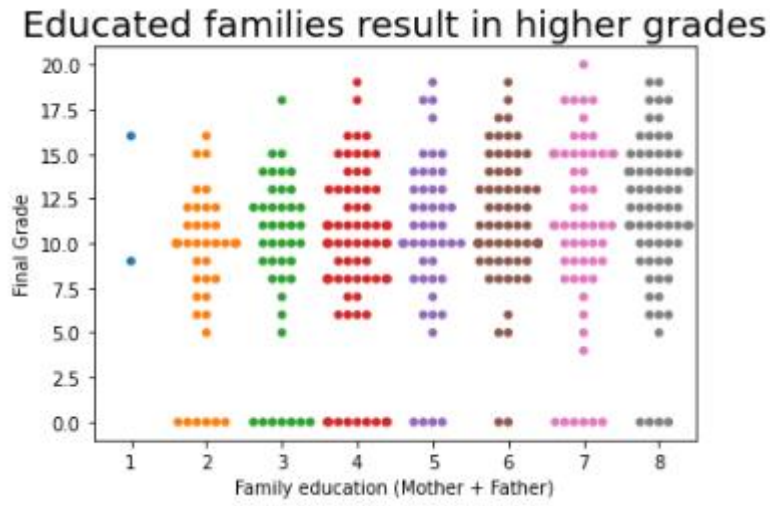


The pattern here is clear: students with less failures score higher in the final grade. This is an intuitive observation that is proven by the empirical data as we have explored above. Next, we want to find out if students with educated families result in higher grades.

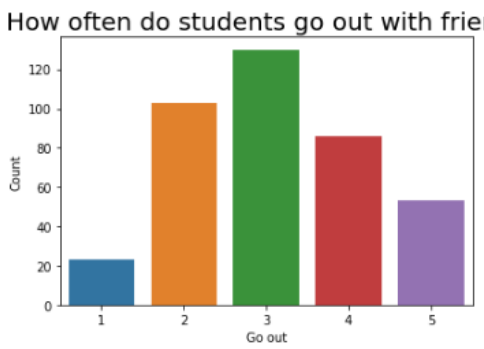
```
family_ed = student_corr['Fedu'] + student_corr['Medu']
b = sns.boxplot(x=family_ed,y=student_corr['G3'])
b.axes.set_title('Educated families result in higher grades', fontsize = 20)
b.set_xlabel('Family education (Mother + Father)', fontsize = 10)
b.set_ylabel('Final Grade', fontsize = 10)
plt.show()
```

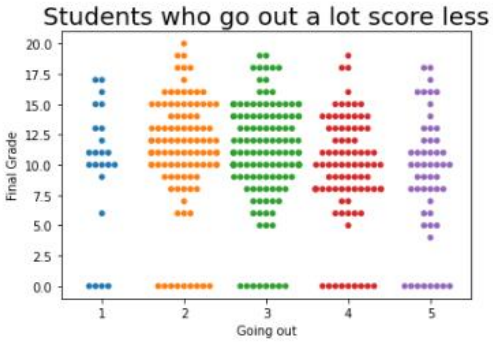


There seems to be a slight trend that with the increase in family education as the grade moves up (apart from the unusual high value at family_ed = 1). Maybe students whose parents did not get to study have more motivation.



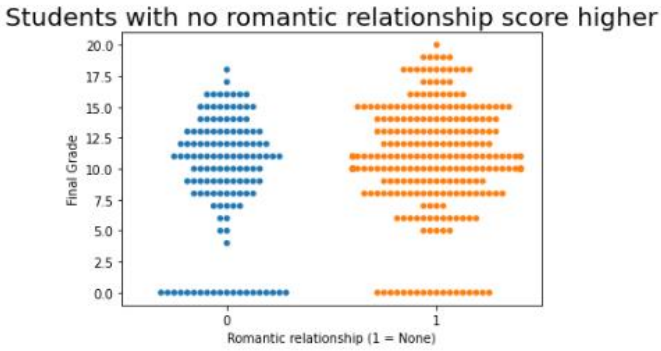
As we can see here, there are only 2 points in family_ed=1, hence our conclusion was faulty. Likewise, it is always advisable to try out different analysis methods to confirm our conclusions from the initial findings. Next, we want to find out if going out with friends often results in higher or lower grades.





This plot shows a slight downward trend, showing that there is a slight trend of students getting lower grades if they hang out with their friends more.

Next, we want to find out if having romantic relationship affect grades. Because of one hot encoding, we have a feature called "romantic_no". When the value is 1, it means there is no romantic relationship, and when it is 0, it means there is romantic relationship.



From this, there does not seem to be an obvious trend. While it may look like those with no romantic relationship gets better grades, we have to take into account that there are many students without any romantic relationships.

4. Machine learning modelling

We can frame the prediction problem in 3 ways:

1. Binary classification:
    - G3 > 10: pass
    - G3 < 10: fail
2. 5-level classification based on Erasmus grade conversion system:
    - 16-20: very good
    - 14-15: good
    - 12-13: satisfactory
    - 10-11: sufficient
    - 0-9: fail
3. Regression: predicting G3

In here, we will be using the third method, regression to predict G3. First, we will divide the data into X and y, then split them into training and test samples using "train_test_split". Remember that here, we are using only the 8 features that are most correlated with the target, "G3".

```
X = student_corr.drop(['G3'], axis=1)
y = student_corr['G3']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
X_train.head()
```

	failures	Medu	higher_yes	higher_no	age	Fedu	goout	romantic_no
28	0	3	1	0	16	4	3	1
58	0	1	1	0	15	2	2	1
52	1	4	1	0	15	2	5	1
353	1	1	1	0	19	1	4	1
218	0	2	1	0	17	3	3	1

For all models, we first initialise the models, fit the models with the training samples, then create "y_pred", which is the predicted values of y on X_test, and then calculate the metrics, mean absolute error, root mean squared error, and r-squared scores, for each model.

Linear regression:

```
lr = LinearRegression()
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.6508134588005188
RMSE:  4.6435108520125885
R-squared:  0.12382707817377103
```

Elastic net:

```
en = ElasticNet(alpha=1.0, l1_ratio=0.5)
en.fit(X_train, y_train)

y_pred = en.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.6718505117687843
RMSE:  4.822735719627613
R-squared:  0.05488680844087379
```

SVM:

```
svr = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')
svr.fit(X_train, y_train)

y_pred = svr.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.549576281856992
RMSE:  4.641224445466279
R-squared:  0.1246896988808871
```

Decision tree:

```
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  4.1044303797468356
RMSE:  5.511521942345544
R-squared:  -0.2343547112332267
```

Sometimes, there are negative r-squared scores, which means the model's predicted values are way off the mark in comparison to the real values. Negative r-squared scores arise when the model selected does not follow the trend of the data, therefore leading to a worse fit than the mean values of the target values to be predicted.

Gradient boosting:

```
gb = GradientBoostingRegressor()
gb.fit(X_train, y_train)

y_pred = gb.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.777663772054233
RMSE:  4.752242322526403
R-squared:  0.08231411268301247
```

Extra trees:

```
et = ExtraTreesRegressor()
et.fit(X_train, y_train)

y_pred = et.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.9445548523206746
RMSE:  5.047831747031905
R-squared:  -0.0353963718944692
```

Random forest:

```
rf = RandomForestRegressor()
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.6053276709226063
RMSE:  4.6613462673985575
R-squared:  0.11708350872158602
```

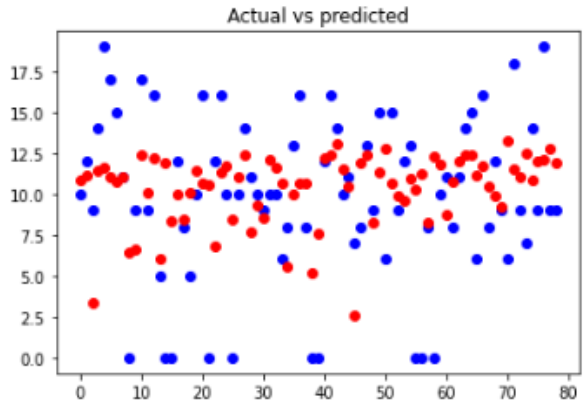
MLP (Multi-layer perceptron):

```
mlp = MLPRegressor()
mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print('MAE: ', mae)
print('RMSE: ', rmse)
print('R-squared: ', r2)
```

```
MAE:  3.689649425868175
RMSE:  4.696958115755143
R-squared:  0.10354132879933509
```

So far, we have used these models without much regularisation imposed. The highest r-squared score achieved is the SVM model. However, the r-squared scores of 0.125 is such a low score that it is akin to saying that the model was not able to predict anything well. But to make things clearer, what if we compare the predicted values with the actual values side-by-side? We will try this with the results from MLP.

```
plt.scatter(range(len(y_test)), y_test, color='blue')
plt.scatter(range(len(y_pred)), y_pred, color='red')
plt.title('Actual vs predicted')
plt.show()
```



The blue points are the actual values, and the red points are the predicted values. As we can see, the model is predicting very close to the mean, which is why the r-squared score is quite low.

	Predictions	Actual
76	10.838929	10
41	11.142589	12
127	3.385270	9
48	11.472356	14
113	11.660408	19

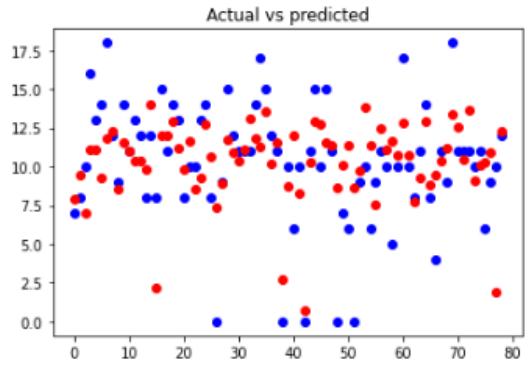
Looking at this short snippet of the predicted and actual values, we can see that the predictions for most of them are quite off. With these features, we can deduce that the model is not able to predict the grades of the students well. But what if we use all features including one-hot encoded features?

5. Regression with all features

We will now try the same prediction with all features including the dummy variables made from one-hot encoding. The results will be displayed directly, and you can refer to the notebook attached for the full codes.

Model	MAE	RMSE	R-squared
Linear regression	2.836	3.824	0.06
Elastic net	2.664	3.796	0.074
SVM	2.688	3.795	0.075
Decision tree	3.595	4.917	-0.554
Gradient boosting	3.022	3.719	0.111
Extra trees	2.879	3.876	0.035
Random forest	2.425	3.244	0.324
MLP	2.608	3.455	0.233

From the results above, we can see that the prediction actually improved by using all features. Random forest is now the best model with r-squared score of 0.324. However, we do not know if this r-squared score can be considered good for this context. Given the problem at hand, we can be tempted to claim the model to be not good in predicting. However, we need further validation to prove this point. We will use the results from the random forest model for our next analyses.



Just by looking at this, we can deduce that the prediction is better than when we were using only 8 features. There are still points where the model predicted clearly wrong, but the predicted points are closer to the actual ones than before.

	Predictions	Actual
161	7.92	7
247	9.46	8
166	7.03	10
346	11.11	16
366	11.06	13

Just like what we can observe from the plot above, it seems that the model is not predicting the higher values properly. In fact, apart from some low scores, the model is still predicting quite close to the mean values. Therefore, we can deduce that the model's prediction is still not that good enough.

6. Feature importance

One good thing about random forest model is that we can decipher the most important features in predicting the target. In random forest, feature importance is calculated by computing how much each feature contributes to decreasing the weighted impurity. The impurity in classification trees is Gini impurity/information gain (entropy), whereas in regression trees, the impurity is measured in variance.

```
importance = rf.feature_importances_

for i, v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i, v))

plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature: 0, Score: 0.04327
Feature: 1, Score: 0.02860
Feature: 2, Score: 0.02306
Feature: 3, Score: 0.03018
Feature: 4, Score: 0.03860
Feature: 5, Score: 0.15179
Feature: 6, Score: 0.02948
Feature: 7, Score: 0.03873
Feature: 8, Score: 0.04561
Feature: 9, Score: 0.01081
Feature: 10, Score: 0.02285
Feature: 11, Score: 0.03482
Feature: 12, Score: 0.18643
Feature: 13, Score: 0.01015
Feature: 14, Score: 0.01141
Feature: 15, Score: 0.00486
Feature: 16, Score: 0.00402
Feature: 17, Score: 0.00810
Feature: 18, Score: 0.00760
Feature: 19, Score: 0.00567
```

From here, we can see that the features "absences" and "goout" are most important in using random forest. However, it is quite evident so far that our predictions are not good enough. We need to improve the model by other means. What if we include "G1" and "G2" features? What implications are there in terms of the problem at hand? What if we used some standardisation tools before putting the data into the model? What if we applied hyperparameter tuning to the models?

Assignment

Using the same data, you are to try the same problem as above, regression prediction of students' final grades (G3), but with some additional steps:

1. Apply standardisation (any standardisation methods) to the features and try predicting the final grade (G3).
2. Apply hyperparameter tuning to the models and explain the differences in performances.
3. Include "G1" and "G2" features to the features and explain the differences in performances.
4. Explain and justify whether "G1" and "G2" features should be added or not.