Math Achievement Analysis Of Portuguese Students

Main Objective

The purpose of this paper is to analyze math achievements of Portuguese students. The dataset contains student grades and characteristics - our aim is to find out which characteristics have the strongest relationship with the grades. In other words, our analysis will focus on **interpretability**.

Data Description

The dataset measures secondary school student achievements in two Portuguese schools. The data attributes include student grades, demographic, social and school related features, and it was collected by using school reports and questionnaires. Study authors provided the datasets regarding the performance in two distinct subjects: Mathematics and Portuguese language. In this study, we will focus on math results.

This dataset was used in the following publication: *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

The description of data attributes is from UCI Machine Learning Repository:

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)

The following columns are grades:

31 G1 - first period grade (numeric: from 0 to 20, the target)
31 G2 - second period grade (numeric: from 0 to 20)
32 G3 - final grade (numeric: from 0 to 20)

After necessary Python module imports, we will examine the first five dataset rows:

In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats.mstats import normaltest
from scipy.stats import boxcox
pd.set_option("display.max_columns", None)
from scipy.special import inv_boxcox
import statsmodels.api as sm
```

In [2]:
```
df = pd.read_csv(r"C:\Users\Lenovo\Desktop\ibm_machine_learning\2_supervised_machine_learning_regression\week3\
df.head()
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes
1	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no
2	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3	yes
3	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no
4	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	no

The output below shows that all variables are either integer or categorical. In total, we have 33 variables and 395 entries. There are no null values:

In [3]:
```
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    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64 
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   Pstatus     395 non-null    object
 6   Medu        395 non-null    int64 
 7   Fedu        395 non-null    int64 
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64 
 13  studytime   395 non-null    int64 
 14  failures    395 non-null    int64 
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 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(16), object(17)
memory usage: 102.0+ KB
```

The following table shows main statistical features of the numeric variables. Since most of them are scores, their values are between 1 and 5 (or 0 and 4). Student age is between 15 and 22, most have had no past class failures (the 75th percentile is 0), drink little alcohol on weekdays (the 75th percentile is 2), and view their health as good (the median is 4). Their family relationships are good (25th and 50th percentiles are 4). Most students have few absences, but some of them have a lot - the maximum value is 75, and the mean is higher than the median (5.7 vs 4), with a high standard deviation of 8. *G1* score (which is out target) has values between 3 and 19, with both mean and median around 11.

In [4]:
```
df.describe()
```

Out[4]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	
count	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395
mean	16.696203	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304	3.235443	3.108861	1.481013	2.291139	3
std	1.276043	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659	0.998862	1.113278	0.890741	1.287897	1
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.000000	1.000000	1.000000	3
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.000000	1.000000	2.000000	4
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.000000	2.000000	3.000000	5
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5

The correlation matrix and heatmap below show that correlations for some variables exceed 0.6:

- Student grades (G1, G2, G3)
- Mother and father education levels (Medu and Fedu)
- Workday and weekend alcohol consumption

In [5]:
```
df[['G1', 'G2', 'G3', 'Fedu', 'Medu', 'Walc', 'Dalc']].corr()
```

Out[5]:

	G1	G2	G3	Fedu	Medu	Walc	Dalc
G1	1.000000	0.852118	0.801468	0.190270	0.205341	-0.126179	-0.094159
G2	0.852118	1.000000	0.904868	0.164893	0.215527	-0.084927	-0.064120
G3	0.801468	0.904868	1.000000	0.152457	0.217147	-0.051939	-0.054660
Fedu	0.190270	0.164893	0.152457	1.000000	0.623455	-0.012631	0.002386
Medu	0.205341	0.215527	0.217147	0.623455	1.000000	-0.047123	0.019834
Walc	-0.126179	-0.084927	-0.051939	-0.012631	-0.047123	1.000000	0.647544
Dalc	-0.094159	-0.064120	-0.054660	0.002386	0.019834	0.647544	1.000000

In [6]:
```
sns.heatmap(df.corr())
```

Out[6]: <AxesSubplot:>

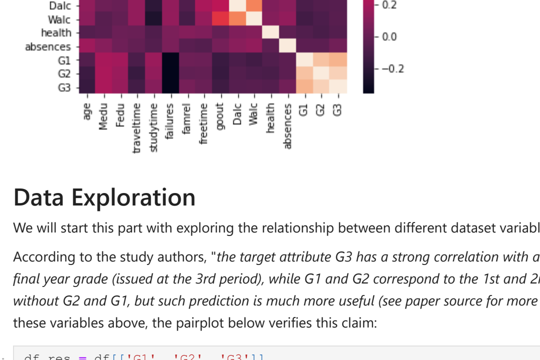
Data Exploration

We will start this part with exploring the relationship between different dataset variables.

According to the study authors, "*the target attribute G3 has a strong correlation with attributes G2 and G1. This occurs because G3 is the final year grade (issued at the 3rd period), while G1 and G2 correspond to the 1st and 2nd period grades. It is more difficult to predict G3 without G2 and G1, but such prediction is much more useful (see paper source for more details).*" We have seen high correlations between these variables above, the pairplot below verifies this claim:

In [7]:
```
df_res = df[['G1', 'G2', 'G3']]
sns.pairplot(df_res).fig.suptitle("Grades", y=1.05)
```

Out[7]: Text(0.5, 1.05, 'Grades')



Also, *G2* and *G3* variables contain many zero values, contrary to *G1*. So we will drop 2nd and 3rd term grades (*G2* and *G3*), and will stick to examining the relationship between the first grade (*G1*) and other variables. *Dalc* and *Fedu* variables will be dropped as well, as per the correlation analysis above:

In [8]:
```
df.drop(['G2', 'G3', 'Dalc', 'Fedu'], axis=1, inplace=True)
```

Now let's analyze our target variable - *G1*.

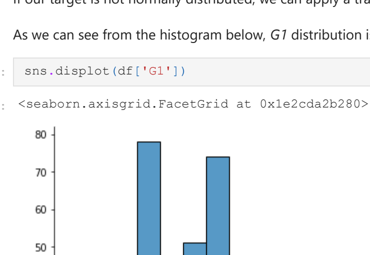
Making our target variable normally distributed often will lead to better results.

If our target is not normally distributed, we can apply a transformation to it and then fit our regression to predict the transformed values.

As we can see from the histogram below, *G1* distribution is not similar to a bell shape due to several spikes in *G1* value frequency:

In [9]:
```
sns.displot(df['G1'])
```

Out[9]: <seaborn.axisgrid.FacetGrid at 0x1e2cda2b280>



A formal statistical test below does not confirm the normal distribution of grade values - p-value is very low:

In [10]:
```
normaltest(df['G1'])
```

Out[10]: NormaltestResult(statistic=22.612048290638576, pvalue=1.2298612026597037e-05)

Applying Box-Cox transformation does not help making *G1* normal. Frequency spikes still remain, and normality test p-value is still very low (although p-value is higher). So we will apply this transformation on *G1* later, but the transformation effects will be not as great as expected.

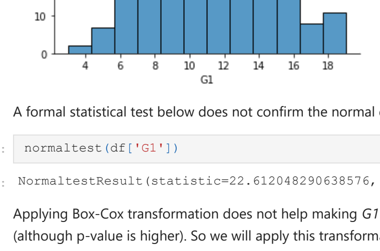
In [11]:
```
bc_result = boxcox(df['G1'])
df_g1 = bc_result[0]
lam = bc_result[1]
```

In [12]:
```
normaltest(df_g1)
```

Out[12]: NormaltestResult(statistic=17.869403653074897, pvalue=0.0001317371607983441)

In [13]:
```
sns.displot(df_g1)
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x1e2cdcdab20>

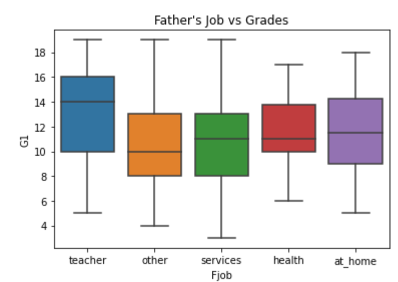


Below we will illustrate some noticeable relationships between the grades and other variables using boxplots and scatterplots.

Father's job as a teacher is related to better scores:

In [14]:
```
sns.boxplot(y='G1', x='Fjob', data=df).set_title("Father's Job vs Grades")
```

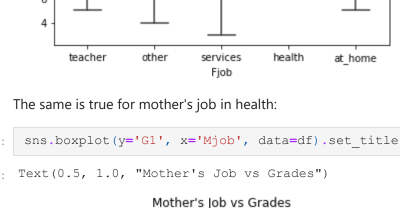
Out[14]: Text(0.5, 1.0, "Father's Job vs Grades")



The same is true for mother's job in health:

In [15]:
```
sns.boxplot(y='G1', x='Mjob', data=df).set_title("Mother's Job vs Grades")
```

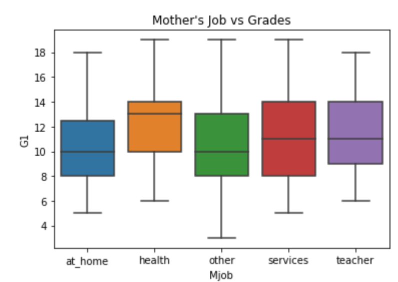
Out[15]: Text(0.5, 1.0, "Mother's Job vs Grades")



Interestingly, having a mother with both none and higher education is related to higher grades:

In [16]:
```
sns.boxplot(y='G1', x='Medu', data=df).set_title("Mother's Education vs Grades")
```

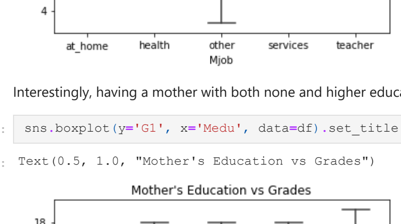
Out[16]: Text(0.5, 1.0, "Mother's Education vs Grades")



Longer study time slightly improves grades, but the longest times do not:

In [17]:
```
sns.boxplot(y='G1', x='studytime', data=df).set_title("Study Time vs Grades")
```

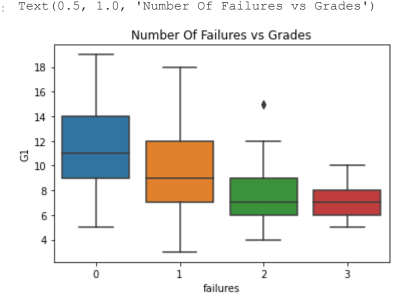
Out[17]: Text(0.5, 1.0, 'Study Time vs Grades')



The number of failures is strongly related to grades:

In [18]:
```
sns.boxplot(y='G1', x='failures', data=df).set_title("Number Of Failures vs Grades")
```

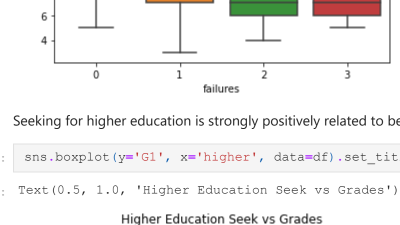
Out[18]: Text(0.5, 1.0, 'Number Of Failures vs Grades')



Seeking for higher education is strongly positively related to better results:

In [19]:
```
sns.boxplot(y='G1', x='higher', data=df).set_title("Higher Education Seek vs Grades")
```

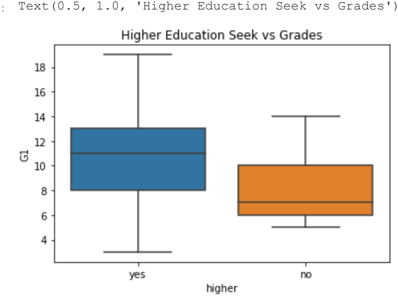
Out[19]: Text(0.5, 1.0, 'Higher Education Seek vs Grades')



Going out very frequently (score 5) is negatively related to grades:

In [20]:
```
sns.boxplot(y='G1', x='goout', data=df).set_title("Frequency Of Going Out vs Grades")
```

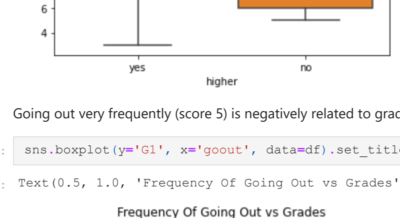
Out[20]: Text(0.5, 1.0, 'Frequency Of Going Out vs Grades')



Students who need educational support from schoold tend to receive lower grades:

In [21]:
```
sns.boxplot(y='G1', x='schoolsup', data=df).set_title("School Educational Support vs Grades")
```

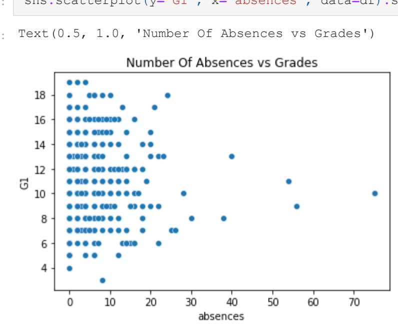
Out[21]: Text(0.5, 1.0, 'School Educational Support vs Grades')



An interesting finding is a weak relationship between the grades and absence numbers. Most students have relatively few absences, among students with both good and poor grades. The few with many absences do not perform very well, but are not in the bottom either:

In [22]:
```
sns.scatterplot(y='G1', x='absences', data=df).set_title("Number Of Absences vs Grades")
```

Out[22]: Text(0.5, 1.0, 'Number Of Absences vs Grades')



The regression analysis below will check how relevant the above findings are. Also, it will be an opportunity to examine the interaction between variables.

Feature Engineering

The next step is feature engineeering. As our dataset has some categorical variables, dummy variables will be made of them. The first values of the categorical variables will be dropped to avoid multicollinearity:

In [23]:
```
df2 = df.copy()
df2 = pd.get_dummies(df2, drop_first=True)
df2.sample(5)
```

Out[23]:

	age	Medu	traveltime	studytime	failures	famrel	freetime	goout	Walc	health	absences	G1	school_MS	sex_M	address_U	famsize
132	17	2	1	1	0	3	4	4	3	5	12	10	0	0	1	
344	18	2	1	3	0	4	3	3	2	3	4	11	0	0	1	
80	15	2	1	1	0	3	2	2	3	3	2	10	0	1	1	
154	17	4	1	1	0	4	2	1	1	4	0	11	0	0	1	
68	15	2	2	2	0	4	1	3	3	4	2	8	0	0	0	

The dataset will be split into a set of independent variables (*X*) and a dependent variable (*y*):

In [24]:
```
X = df2.drop('G1', axis=1)
y = df2['G1']
```

Regression Analysis

Before running our regression analysis, we will split both *X* and *y* into training and testing subsets. The testing subsets will be used to verify our models:

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=72018)
```

We can add polynomial features to our predictor dataset, in order to examine if some predictors have a non-linear relationship with the target variable, and if any interaction between features exists. Only second degree features will be generated:

In [26]:
```
pf = PolynomialFeatures(2)
X_poly = pd.DataFrame(pf.fit_transform(X), columns=pf.get_feature_names(X.columns))
X_poly_train, X_poly_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3,
                                                              random_state=72018)
```

Next, we will apply standard scaling to our features. We will scale the dummy variables as well, given Robert Tibshirani's recommendation in this link: "*The lasso method requires initial standardization of the regressors, so that the penalization scheme is fair to all regressors. For categorical regressors, one codes the regressor with dummy variables and then standardizes the dummy variables.*"

The scaling will be performed on both initial and polynomial training datasets. The related transformation is applied to the test datasets:

In [27]:
```
s = StandardScaler()
X_train_s = s.fit_transform(X_train)
X_test_s = s.transform(X_test)
```

In [28]:
```
s = StandardScaler()
X_poly_train_s = s.fit_transform(X_poly_train)
X_poly_test_s = s.transform(X_poly_test)
```

As discussed above, Box-Cox transformation will be applied on our predicted variable. The transformation parameter will be obtained from the train dataset only and later applied to the test dataset, to prevent data leakage:

In [29]:
```
bc_result = boxcox(y_train)
y_train_bc = bc_result[0]
lam = bc_result[1]
```

The polynomial dataset has 741 feature in total:

In [30]:
```
X_poly_train_s.shape
```

Out[30]: (276, 741)

Now we will run our regression analyses to examine the relationship between student grades and their characteristics. We will test three models, starting with a simple linear regression as a baseline. Then, we will add polynomial features to the linear regression and check if they increase predictability and/or interpretability. Also, we will try a regularized linear regression model, in particular - Lasso.

For every model, we will select the features with Sequential Feature Selector. It is a greedy procedure that iteratively finds the best new feature to add to the set of selected features. We initially start with zero features and find the one feature that maximizes a cross-validated score when an estimator is trained on this single feature. Once that first feature is selected, we repeat the procedure by adding a new feature to the set of selected features. Since we want to keep our models interpretable, we will select 10 independent variables in each case. Otherwise, we will stick to default parameters - choose forward direction and use 5-fold cross validation.

For all these regressions, R-squared score and mean squared error will be calculated, in order to assess predictive power of each model. Also, we will draw regression plots showing predicted vs actual test dataset target values.

Simple Linear Regression

So we begin with a simple linear regression:

In [31]:
```
lr = LinearRegression()
selector_lr = SequentialFeatureSelector(lr, n_features_to_select=10)
selector_lr = selector_lr.fit(X_train_s, y_train_bc)
```

Now, we will run linear regression with the selected variables only:

In [32]:
```
selected_col_list = X_train.columns[selector_lr.support_].tolist()
```

R-squared score for the train dataset is 0.29, while only 0.23 for the test dataset:

In [33]:
```
X_train_s_selected = pd.DataFrame(X_train_s, columns=X.columns).loc[:, selected_col_list]
lr_selected = LinearRegression()
lr_selected.fit(X_train_s_selected, y_train_bc)
y_train_pred_lr = lr_selected.predict(X_train_s_selected)
r2_score(y_train_bc, y_train_pred_lr)
```

Out[33]: 0.2915290567398109

In [34]:
```
X_test_s_selected = pd.DataFrame(X_test_s, columns=X.columns).loc[:, selected_col_list]
y_test_pred_lr = lr_selected.predict(X_test_s_selected)
y_test_pred_lr_bc = inv_boxcox(y_test_pred_lr, lam)
r2_score(y_test.values, y_test_pred_lr_bc)
```

Out[34]: 0.228883495665668

In [35]:
```
mean_squared_error(y_test.values, y_test_pred_lr_bc)
```

Out[35]: 8.902608381725862

R-squared for the test subset is lower, as expected. However, it is not high in both cases. Probably, spikes in *G1* distribution makes it harder to accurately predict its values.

The features and their coefficients are given below.

The most important features related to better grades are father's job as a teacher, study time and student's male sex.

However, a number of failures, need for educational support, mother's 'other' job and going out frequently are related to worse grades.

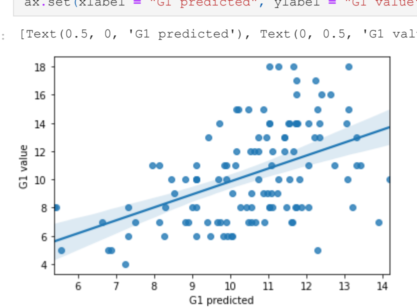
In [36]:
```
lr_selected_coef = pd.DataFrame(zip(selected_col_list, lr_selected.coef_)).sort_values(by=1)
lr_df = pd.DataFrame(lr_selected_coef[abs(lr_selected_coef.iloc[:, 1]) > 0])
lr_df.columns = ['Feature', 'Coefficient']
lr_df
```

Out[36]:

	Feature	Coefficient
1	failures	-0.416176
8	schoolsup_yes	-0.277753
5	Mjob_other	-0.238823
2	goout	-0.204732
9	famsup_yes	-0.197863
6	Mjob_teacher	-0.081814
4	Mjob_health	0.117491
3	sex_M	0.154679
0	studytime	0.195423
7	Fjob_teacher	0.308224

In [37]:
```
ax = sns.regplot(x=y_test_pred_lr_bc, y=y_test)
ax.set(xlabel = "G1 predicted", ylabel = "G1 value")
```

Out[37]: [Text(0.5, 0, 'G1 predicted'), Text(0, 0.5, 'G1 value')]



It looks like our model predicts average values more accurately than both low and high values. For example, the range for fitted values is noticeably narrower than for the actual ones. On the other hand, even for middle values, the predictions can vary - for example, students with grade equal to 11 can get predictions between 8 and 13.

Linear Polynomial Regression

The next step is to run simple linear regression with selected polynomial variables. We will again select 10 features:

In [38]:
```
lr_poly = LinearRegression()
selector_lr_poly = SequentialFeatureSelector(lr_poly, n_features_to_select=10)
selector_lr_poly = selector_lr_poly.fit(X_poly_train_s, y_train_bc)
```

In [39]:
```
selected_col_list_poly = X_poly_train.columns[selector_lr_poly.support_].tolist()
```

Using polynomial features improves prediction accuracy on the training dataset, but R-squared score is much lower for the testing dataset (R-squared score equals 0.39 and 0.07, respectively). Furthermore, mean squared error is higher as well (10.75). Probably, choosing a different (perhaps smaller) number of polynomial features would improve the prediction accuracy on the testing dataset:

In [40]:
```
X_train_s_selected_poly = pd.DataFrame(X_poly_train_s, columns=X_poly.columns).loc[:, selected_col_list_poly]
lr_selected_poly = LinearRegression()
lr_selected_poly.fit(X_train_s_selected_poly, y_train_bc)
y_train_pred_lr_poly = lr_selected_poly.predict(X_train_s_selected_poly)
r2_score(y_train_bc, y_train_pred_lr_poly)
```

Out[40]: 0.3881875028938202

In [41]:
```
X_test_s_selected_poly = pd.DataFrame(X_poly_test_s, columns=X_poly.columns).loc[:, selected_col_list_poly]
y_test_pred_lr_poly = lr_selected_poly.predict(X_test_s_selected_poly)
y_test_pred_lr_poly_bc = inv_boxcox(y_test_pred_lr_poly, lam)
y_test_pred_lr_poly_bc[np.isnan(y_test_pred_lr_poly_bc)] = np.nanmean(y_test_pred_lr_poly_bc)
y_test_pred_lr_poly_bc = np.nan_to_num(y_test_pred_lr_poly_bc)
r2_score(y_test, y_test_pred_lr_poly_bc)
```

Out[41]: 0.06927794952242738

In [42]:
```
mean_squared_error(y_test.values, y_test_pred_lr_poly_bc)
```

Out[42]: 10.745268556781184

Below we will examine the polynomial features for this model:

In [43]:
```
poly_df = pd.DataFrame(zip(selected_col_list_poly, lr_selected_poly.coef_)).sort_values(by=1)
poly_df = poly_df[abs(poly_df.iloc[:, 1]) > 0]
poly_df.columns = ['Feature', 'Coefficient']
poly_df
```

Out[43]:

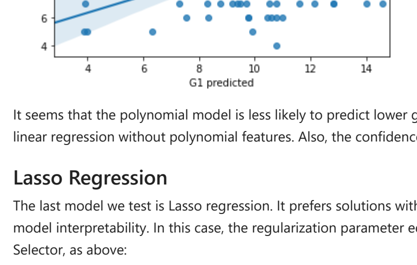
	Feature	Coefficient
2	failures nursery_yes	-0.569711
5	Pstatus_T famsup_yes	-0.250442
3	famrel schoolsup_yes	-0.242142
7	Fjob_teacher reason_reputation	-0.150505
1	traveltime famsize_LE3	0.171874
9	guardian_other nursery_yes	0.211676
0	Medu studytime	0.221469
4	address_U Mjob_health	0.244225
6	Mjob_services higher_yes	0.287323
8	Fjob_teacher internet_yes	0.449687

Unfortunately, many interaction variables are hard to interpret. For example, it is not clear why father's job as a teacher is positively related to grades in interaction with internet connection, but is negatively related to the scores in interaction with choosing the school for its reputation.

So using polynomial features makes the model less interpretable (and in our case, makes it even worse at predicting).

In [44]:
```
ax = sns.regplot(x=y_test_pred_lr_poly_bc, y=y_test)
ax.set(xlabel = "G1 predicted", ylabel = "G1 value")
```

Out[44]: [Text(0.5, 0, 'G1 predicted'), Text(0, 0.5, 'G1 value')]



It seems that the polynomial model is less likely to predict lower grades and more likely to predict average ones, compared to the simple linear regression without polynomial features. Also, the confidence interval of regression line slope is much wider in the lower grade end.

Lasso Regression

The last model we test is Lasso regression. It prefers solutions with fewer non-zero coefficients, so it might be beneficial in seeking more model interpretability. In this case, the regularization parameter equals 0.1, and we choose half of the features using Sequential Feature Selector, as above:

In [45]:
```
las01 = Lasso(alpha=0.1)
selector_las01 = SequentialFeatureSelector(las01, n_features_to_select=10)
selector_las01 = selector_las01.fit(X_train_s, y_train_bc)
```

In [46]:
```
selected_col_list_las01 = X_train.columns[selector_las01.support_].tolist()
```

R-squared score is lower for both train and test sets, so its predictability is lower than of the simple linear regression model:

```
X_train_s_selected_las01 = pd.DataFrame(X_train_s, columns=X.columns).loc[:, selected_col_list_las01]
las01_selected = Lasso(alpha=0.1)
las01_selected.fit(X_train_s_selected_las01, y_train_bc)
y_train_pred_las01 = las01_selected.predict(X_train_s_selected_las01)
r2_score(y_train_bc, y_train_pred_las01)
```

```
0.2370643713678784
```

```
X_test_s_selected_las01 = pd.DataFrame(X_test_s, columns=X.columns).loc[:, selected_col_list_las01]
y_test_pred_las01 = las01_selected.predict(X_test_s_selected_las01)
y_test_pred_las01_bc = inv_boxcox(y_test_pred_las01, lam)
r2_score(y_test.values, y_test_pred_las01_bc)
```

```
0.17046431642477045
```

```
mean_squared_error(y_test.values, y_test_pred_las01_bc)
```

```
9.57706298338495
```

Several features out of the selected subset have zero coeffcients, the magnitude of others is lower. There are 8 non-zero coefficients for this Lasso model:

```
las_selected_coef = pd.DataFrame(zip(selected_col_list_las01, las01_selected.coef_)).sort_values(by=1)
las_df = las_selected_coef[abs(las_selected_coef.iloc[:, 1]) > 0]
las_df.columns = ['Feature', 'Coefficient']
```

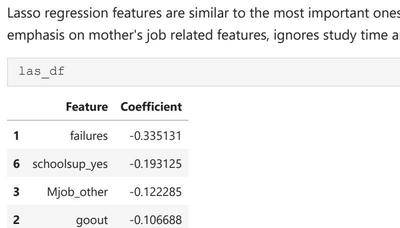
Lasso regression features are similar to the most important ones from the initial linear regression. However, Lasso regression puts less emphasis on mother's job related features, ignores study time and student's sex:

```
las_df
```

	Feature	Coefficient
1	failures	-0.335131
6	schoolsup_yes	-0.193125
3	Mjob_other	-0.122285
2	goout	-0.106688
7	famsup_yes	-0.079187
0	Medu	0.059568
4	Fjob_teacher	0.175415

```
ax = sns.regplot(x=y_test_pred_las01_bc, y=y_test)
ax.set(xlabel = "G1 predicted", ylabel = "G1 value")
```

```
[Text(0.5, 0, 'G1 predicted'), Text(0, 0.5, 'G1 value')]
```



Compared to the simple linear regression model, Lasso model is less likely to predict extreme values on both ends - the range of predicted grades is narrower.

Conclusion

Out of the examined models, the simple linear regression model seems to have the highest predictive power - it has the highest R-squared value and the lowest mean squared error on the test dataset. Also, it is not much worse in interpretabilty compared to the Lasso model. So we will examine how well its results align with regression model assumptions.

Diagnostics

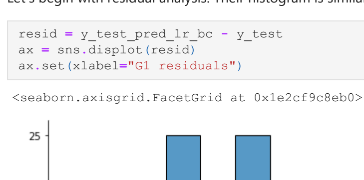
As per this source, linear regression makes several assumptions about the data, such as :

- Linearity of the data. The relationship between the predictor (x) and the outcome (y) is assumed to be linear.
- Normality of residuals. The residual errors are assumed to be normally distributed.
- Homogeneity of residuals variance. The residuals are assumed to have a constant variance (homoscedasticity)
- Independence of residuals error terms.

Let's begin with residual analysis. Their histogram is similar to a bell shape:

```
resid = y_test_pred_lr_bc - y_test
ax = sns.displot(resid)
ax.set(xlabel="G1 residuals")
```

```
<seaborn.axisgrid.FacetGrid at 0x1e2cf9c8eb0>
```



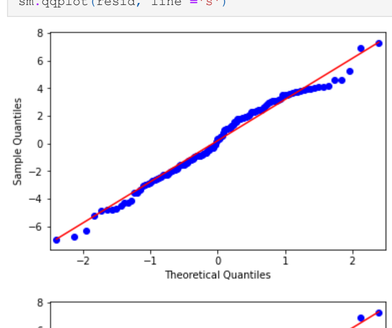
Normality test for the residuals allows us to assume that they are normally distributed (p-value is above 0.05):

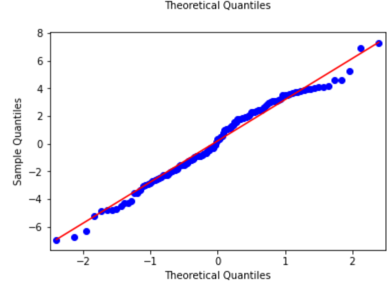
```
normaltest(resid)
```

```
NormaltestResult(statistic=2.962339572325217, pvalue=0.22737155665480807)
```

The QQ-plot below also shows that the residuals are approximately normally distributed, with some points on both ends deviating from it:

```
sm.qqplot(resid, line ='s')
```

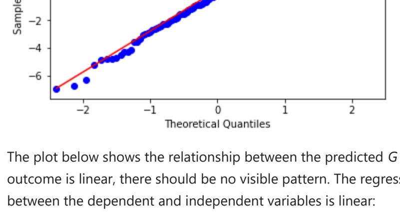




The plot below shows the relationship between the predicted *G1* values and residuals. If the relationship between the predictors and the outcome is linear, there should be no visible pattern. The regression line is almost horizontal, so we can assume that the relationship between the dependent and independent variables is linear:

```
ax = sns.regplot(x=y_test_pred_lr_bc, y=resid)
ax.set(xlabel = "G1 predicted", ylabel = "Residuals")
```

```
[Text(0.5, 0, 'G1 predicted'), Text(0, 0.5, 'Residuals')]
```

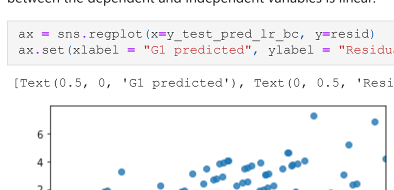


The next step is to examine homogeneity of variance of the residuals (homoscedasticity). A horizontal line with equally spread points is a good indication of homoscedasticity. This is not the case in our example, where we have a heteroscedasticity problem. Standardized residuals increase with the grade value:

```
stand_resid = resid / np.std(resid)
```

```
ax = sns.regplot(x=y_test_pred_lr_bc, y=np.sqrt(np.abs(stand_resid)))
ax.set(xlabel = "G1 predicted", ylabel = "sqrt(Standardized Residuals)")
```

```
[Text(0.5, 0, 'G1 predicted'), Text(0, 0.5, 'sqrt(Standardized Residuals)')]
```



The conclusion is that the simple linear regression model deviates from linear regression assumptions and needs improvement.

Key Findings And Insights

We tested three regression models to examine the relationships between student features and their math grades.

A simple linear regression model has the highest predictability - its R-squared score for the test dataset is about 0.23. Also, it has the lowest mean squared error (8.90). The most important features related to better grades are father's job as a teacher, study time and student's male sex. However, a number of failures, need for educational support, mother's 'other' job and going out frequently are related to worse grades.

Adding polynomial features to the above mentioned linear regression model gave no benefit in both predictability and interpretability - R-squared score for the test dataset is the lowest. Also, many interaction variables are hard to interpret. For example, it is not clear why father's job as a teacher is positively related to grades in interaction with internet connection, but is negatively related to the scores in interaction with choosing the school for its reputation.

Lasso regression comes second in terms of predictabilty - its R-squared score for the test dataset is about 0.17. Out of 10 selected variables, Lasso regression keeps 7 with non-zero coefficients. The positively related features are father's job as a teacher and mother's education. While a numer of failures, need for educational support, mother's 'other' job and going out frequently had a negative relationship with the scores.

To sum up, the simple linear regression model was the best of the three in terms of predictability - it has the highest R-squared score and the lowest mean squared error. So it was selected for the further diagnostics. However, it deviates from linear regression model assumptions - heteroscedasticity is detected. Possible suggestions to improve the situation are discussed in the last part.

One of the problems with all three examined models is their low R-squared score. Probably, the distribution of the independent *G1* variable makes it hard to predict accurately. For more predictability, alternative approaches might be examined (see the part below).

Suggestions

Our dataset contains many discrete numeric variables that are scores, such as parents' education, travel time, study time etc. They were treated as numeric variables in this analysis, under "the assumption that the numerical distance between each set of subsequent categories is equal" (see the link). Since some of them are not accurate descriptions of numeric features (e.g., the number of failures), this assumption might be too optimistic. So the model could be reassessed treating score variables as categorical, although this approach has a disadvantage as well - we would lose the ordering information. For example, travel time score values increase together with the number of minutes.

We saw that *G1* values are not normally distributed, and Box-Cox transformation was not enough to fix it. One possible way to improve the model could be applying other transformations to this variable, especially to make *G1* score spikes less influential. Moreover, particular influential cases (extreme values that might influence the regression results when included or excluded from the analysis) could be identified and removed from the analysis.

In selecting features, we relied on automatic selection of most useful features, selecting 10 for each model. Maybe changing the feature number, or choosing another selection procedure could improve the model efficiency.

Furthermore, alternative regression modelling approaches could be tried to improve the model quality, especially to deal with the aforementioned nonsmoothness of the predicted variable.

Also, a similar analysis might be conducted on Portuguese language results. This way, we would examine if our insights regarding math achievements are relevant to other school subjects as well.