In [ ]:
```
### Final Project DTSC 670
# Abigail Rhea
# 2/17/2024
```

In [1]:
```
# import necessary pandas and numpy imports
import pandas as pd
import numpy as np

# Run necessary code to guarantee file runs in CodeGrade
pd.set_option('display.max_columns', 20)
np.set_printoptions(suppress=True)
```

In [2]:
```
## FRAME THE PROBLEM AND LOOK AT THE BIG PICTURE
```

In [3]:
```
### 1) I will create a robust machine learning regression model to determine the final
# of the student population. This will assist the school system in determining which s
# are doing well in their studies and which students need more assistance. This soluti
# will look at many different features within the dataset to see if there is a correla
# between said features and the final grade of the students.

### 2 & 3) This model uses a form of regression that requires supervised machine learn
# Supervised machine learning just means that we use labeled datasets that can train a
# to classify the data and make predictions and measure prediction accuracy over time.
# By using a regression model, we can look at the relationship between independent and
# variables. Our end goal is to train the model to learn the impact of these relations
# variables (i.e. the features listed below and the final grade of the last term, G3)
# based on those impacts.

### 4) I will use a few select regression tasks to predict the squared error of the mo
# I've chosen: Linear Regression, Decision Tree Regression, Random Forest Regression
```

In [4]:
```
## GET THE DATA
```

In [5]:
```
### 1) import student-mat.csv file and name the DataFrame student_info
student_info = pd.read_csv('student-mat.csv')
```

In [6]:
```
student_info.head()
```

Out[6]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	goout	Dalc	Wa
0	GP	F	18.0	U	GT3	A	4	4	at_home	teacher	...	4	1	
1	GP	F	17.0	U	GT3	T	1	1	at_home	other	...	3	1	
2	GP	F	15.0	U	LE3	T	1	1	at_home	other	...	2	2	
3	GP	F	15.0	U	GT3	T	4	2	health	services	...	2	1	
4	GP	F	NaN	U	GT3	T	3	3	other	other	...	2	1	

5 rows × 35 columns

In [7]:
```
# It is noteworthy that the median value for final grade is 11.
# The average final grade is 10.4.
```

```
student_info.describe()
```

Out[7]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime
count	383.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.00000C
mean	16.699739	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304	3.235443
std	1.280615	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659	0.998862
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.00000C
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000	3.00000C
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.00000C
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.00000C
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.00000C

In [89]:

```
### 2) Check size and type of data

print(student_info.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 35 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   school       395 non-null    object 
 1   sex          395 non-null    object 
 2   age          383 non-null    float64
 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_G1  381 non-null    float64
 30  absences_G2  381 non-null    float64
 31  absences_G3  381 non-null    float64
 32  G1           395 non-null    int64  
 33  G2           395 non-null    int64  
 34  G3           395 non-null    int64  
dtypes: float64(4), int64(14), object(17)
memory usage: 108.1+ KB
None
```

In [9]:
```
### 3) The features in this dataset include:
 # school - the student's school: GP or MS
 # sex - the student's sex: F or M
 # age - the student's age: between 15 and 22
 # address - the student's home adress type: U for urban & R for rural
 # famsize - the student's family size: LE3 for 3 or less & GT3 for 4 or more people
 # Pstatus - parent's status: living together or apart (T or A)
 # Medu - mother's education: 0 - none, 1 - up to 4th grade, 2-up to 9th grade, 3-secc
    # 4-higher education
 # Fedu - father's education: same as mother's
 # Mjob - mother's job
 # Fjob - father's job
 # reason - reason for choosing a school: close to "home", "reputation", "course" opti
 # guardian - student's guardian (mother, father, other)
 # traveltime - from home to school ( 1: 15 min or less, 2: 15-30 mins, 3:30mins to 1h
 # studytime - weekly (1: <2hr, 2: 2-5 hr, 3:5-10hr, 4: >10hr)
```

```
 # failures - number of failures (n if 1<=n< 3, else 4)
 # schoolsup - extra edu support (yes or no)
 # famsup - family support ( yes or no)
 # paid - extra paid classes (yes or no)
 # activities - extra activities (yes or no)
 # nursey - attended nursey school (yes or no)
 # higher - wants to reach higher edu (yes or no)
 # internet - home access (yes or no)
 # romantic - in a relationship (yes or no)
 # famrel - quality of family relationships ( from 1 to 5; very bad to excellent)
 # freetime - free time after school (from 1 to 5; low to very high)
 # goout - going out with friends (from 1 to 5; very low to very high)
 # Dalc - workday alcohol consumption(1 to 5; low to high)
 # Walk - weekend alcohol consumption(1 to 5; low to high)
 # health - current health status (1 to 5; very bad to very good)
 # absences_G1 - number of school absences for G1 term
 # ansences_G2 - absences in G2 term
 # absences_G3 - absences in G3 term
 # G1 - term 1 grade (numerical value between 0-20)
 # G2 - term 2 grade (numerical value between 0-20)

### 4) The target is the grade for term G3: numeric value between 0-20
```

In [10]:

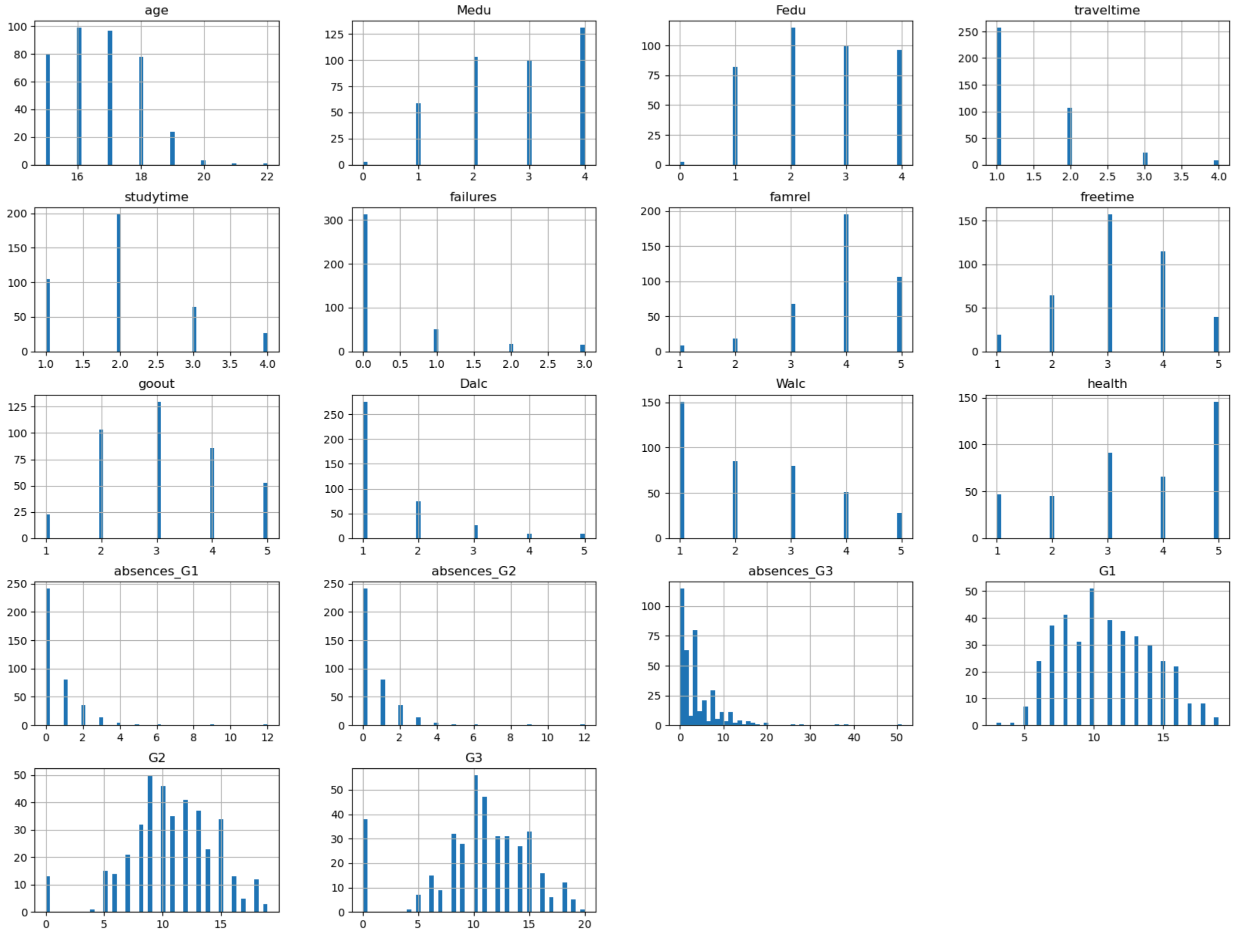
```
### 5) Split the data into training and test sets
# First, I will separate the G3 column and put it in it's own dataframe.
# We only want certain transformations to be used
# on the attributes vs the target, which is the final grade.

student_features = student_info.drop("G3", axis=1)
student_features_new = student_features[['absences_G1', 'absences_G2', 'absences_G3','
                                         'Walc','health','Medu','famrel', 'G1','G2','s
                                        'internet','studytime', 'age', 'higher']]
student_label = student_info["G3"].squeeze()

# Now we can create a training and test set. The test set will be some random values f
# dataset, precisely %20 of it. So, we will pull that aside and use the remaining 80%
# transformations later. The remaining 80% is known as the training set.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(student_features_new, student_labe
```

In [11]:

```
## EXPLORE THE DATA TO GAIN INSIGHTS
```

In [12]:

```
### Explore the training set attributes and there characteristics
# Let's plot a histogram of all the numerical attributes in the dataframe.

import matplotlib.pyplot as plt

student_info.hist(bins=50, figsize=(20,15))
plt.show()
```

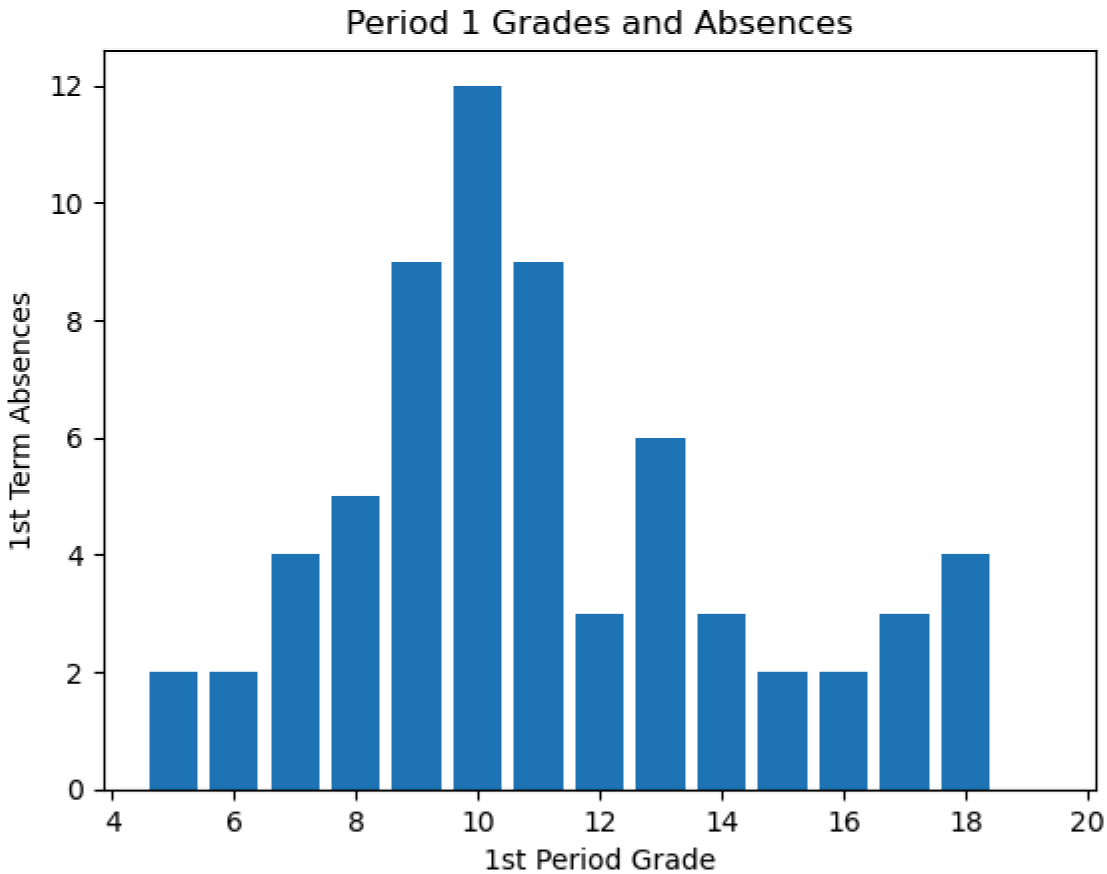


In [13]:
```
### Thoroughly study training set attributes and characteristics.

# A few things I notice from the above plots are that absences increase in the last pe
# The grade scores in term one do not include as many zeros, but the amount of zeros i
# Most of the students live close by to their school of choice.
# Family relationships are generally high.
# The grades for all three periods are relatively evenly skewed.
# Most student's study an average of 2 hours and there are very few failures.
# Students state their mothers comsume alchohol more than fathers.
```

In [14]:
```
### Produce at least four visualizations
```

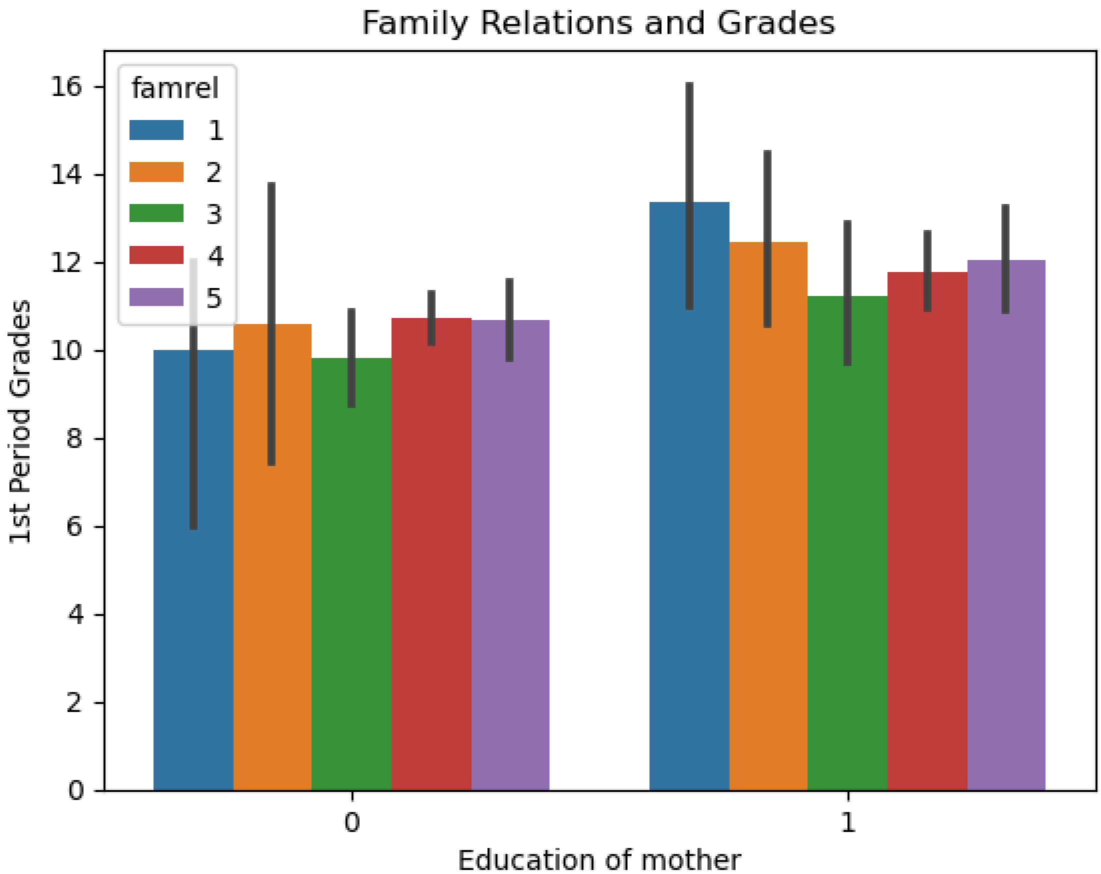
In [15]:
```
plt.bar(X_train['G1'], (X_train['absences_G1']))

plt.title("Period 1 Grades and Absences")
plt.xlabel("1st Period Grade")
plt.ylabel("1st Term Absences")

# The bar chart below shows the students grade in period 1 and the count of absences
# in period 1. The plot shows that the highest number of absences gives
# a grade somewhere between 9 and 11.
```

Out[15]:
```
Text(0, 0.5, '1st Term Absences')
```

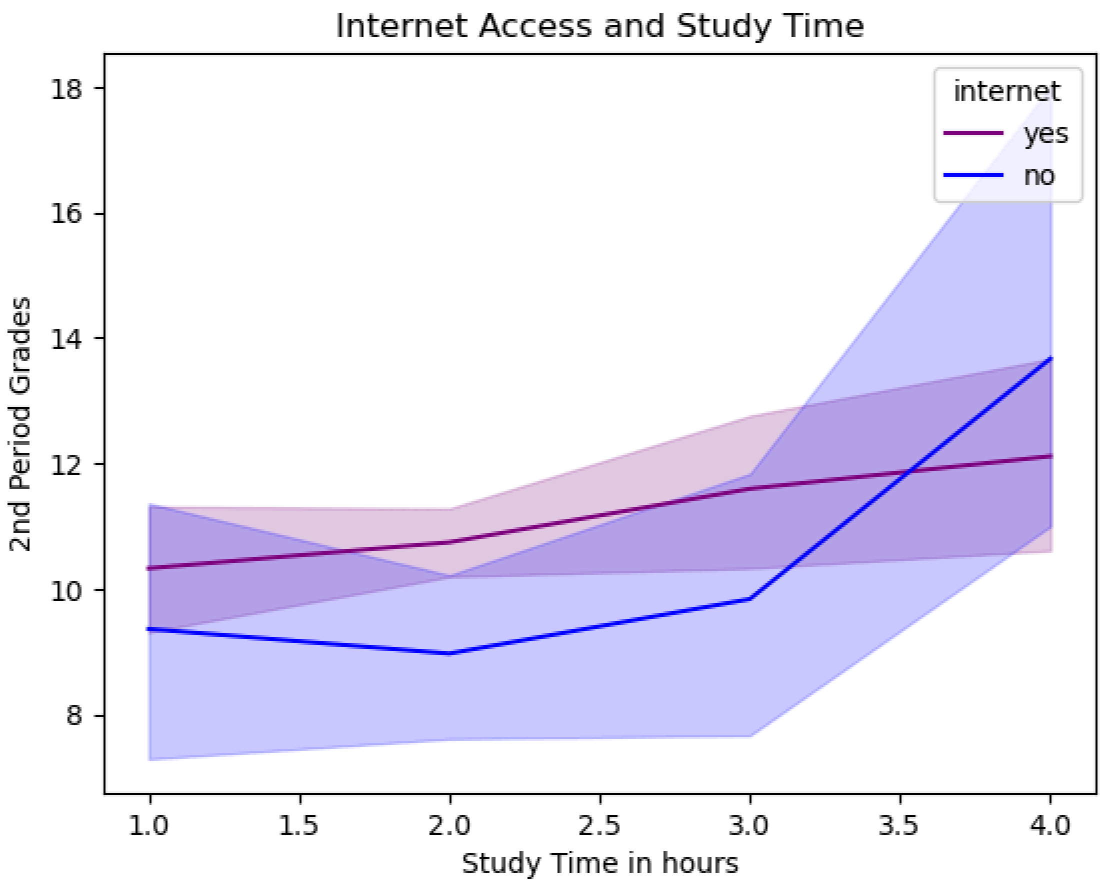


In [16]:
```
def higher_education(mom):
    if mom <= 3 :
        return 0
    if mom == 4 :
        return 1


X_train['Mhigher_edu'] = X_train.apply(lambda x: higher_education(x["Medu"]), axis=1)
```

In [17]:
```
# Now let's use a different library to view some data. I will import the seaborn
# library. It provides some more options for plotting.

import seaborn as sns

sns.barplot(x='Mhigher_edu', y='G1', data=X_train, hue='famrel')
plt.title("Family Relations and Grades")
plt.xlabel("Education of mother")
plt.ylabel("1st Period Grades")

plt.show()


# This bar plot represents the education of the mother. I've applied a function to
# a new column in the student_train dataset that scores 1 if
# the mother received any higher education. All other forms of education fall under th
# I wanted to see if this is an important factor on the relationship the
# student has with the family and how that affects the student's final grade. I was su
# The higher grades came from student's whose mother did receive a higher edu, but who
# on the quality of the family relationships. Otherwise, overall, student's whose moth
# scored slightly higher in the 1st term.
```

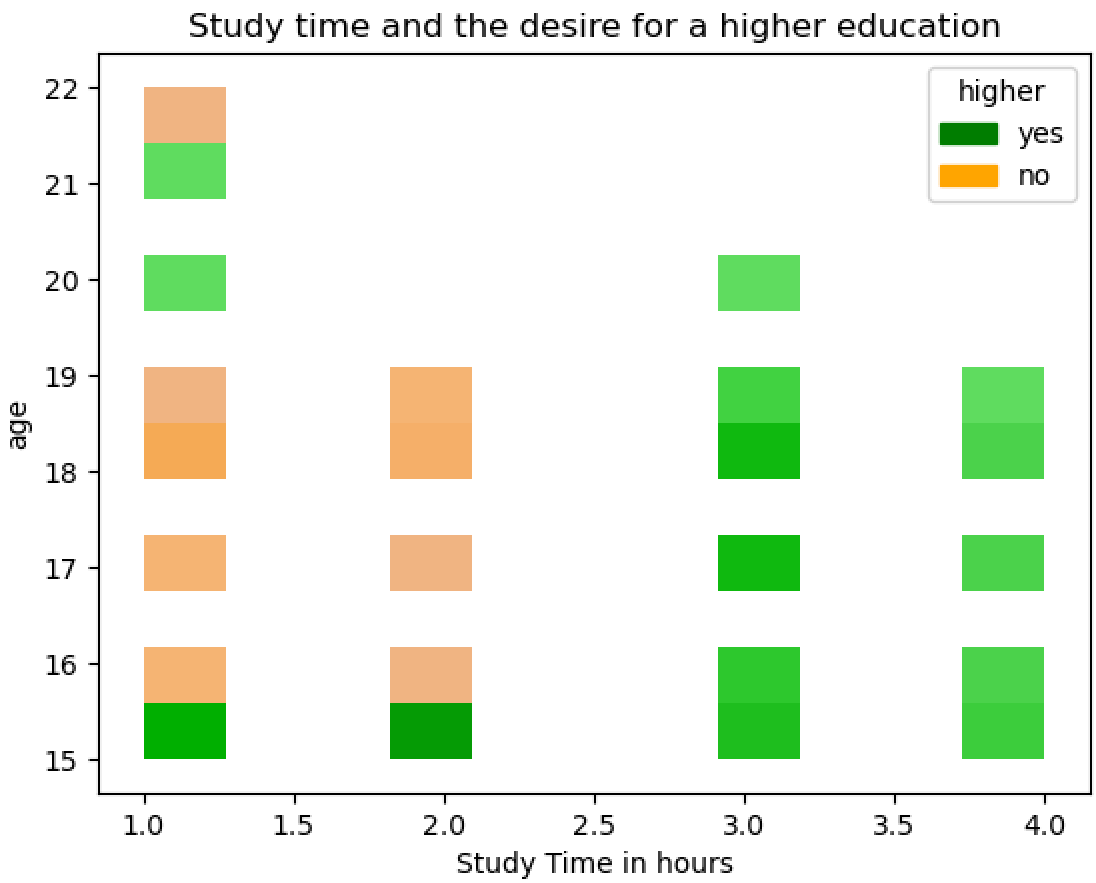


In [18]:
```
sns.lineplot(x="studytime", y="G2", data=X_train, hue='internet', palette=['purple', '
plt.title("Internet Access and Study Time")
plt.xlabel("Study Time in hours")
plt.ylabel("2nd Period Grades")

plt.show()

# In this line plot, we can see the affects of internet access at home to the amount o
# We also see how that amount of study time affects the grades in the 2nd period. It l
# have steady study habits, with only a slight increase in time. We also see that the 
# also relatively steady. Whereas, the student's who don't have access to internet stu
# to student's who don't have internet access and don't study otherwise.
```

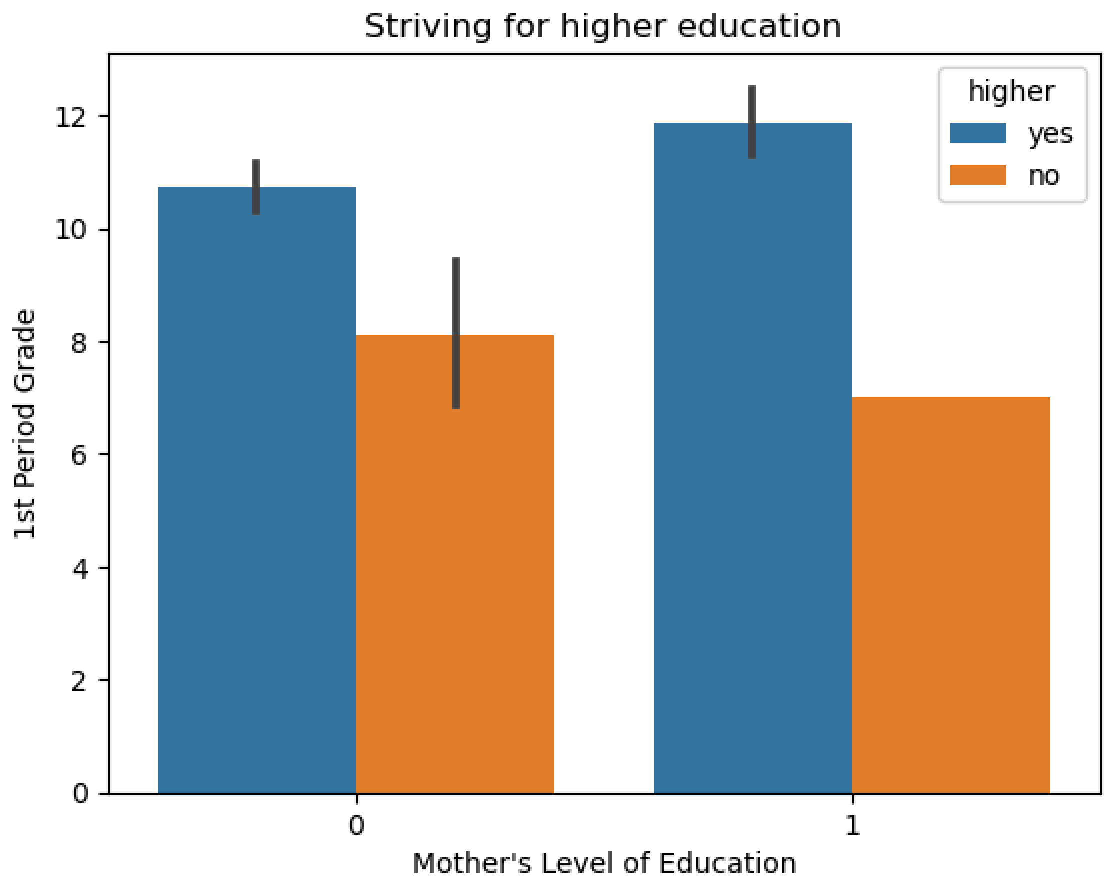


In [19]:

```
sns.histplot(x='studytime', y='age', data=X_train, hue='higher', palette=['green', 'or
plt.title("Study time and the desire for a higher education")

plt.xlabel("Study Time in hours")

plt.show()

# As we can see on this histogram, the desire to reach a higher education affects
# study time immensely. We can see that student's of all ages who don't want to
# receive any higher education, don't spend as much time studying. I'll draw another
# barplot below to show the correlation between student's who don't wany education a
# and their parents level of education received. I have a feeling this will affect
# the final grade of the student.
```



In [20]:
```
sns.barplot(x='Mhigher_edu', y="G1", data=X_train, hue="higher")

plt.title("Striving for higher education")
plt.xlabel("Mother's Level of Education")
plt.ylabel("1st Period Grade")

plt.show()

# Here we can see that mother's having a higher education does not increase
# the student's desire for a higher education. In fact, either way, the student's
# seem to deny wanting a higher education in both scenarios.
```



In [21]:
```
# I'm going to drop the Mhigher_edu columns that we added earlier so I don't mess
# up my calculations.
X_train = X_train.drop('Mhigher_edu', axis=1)
```

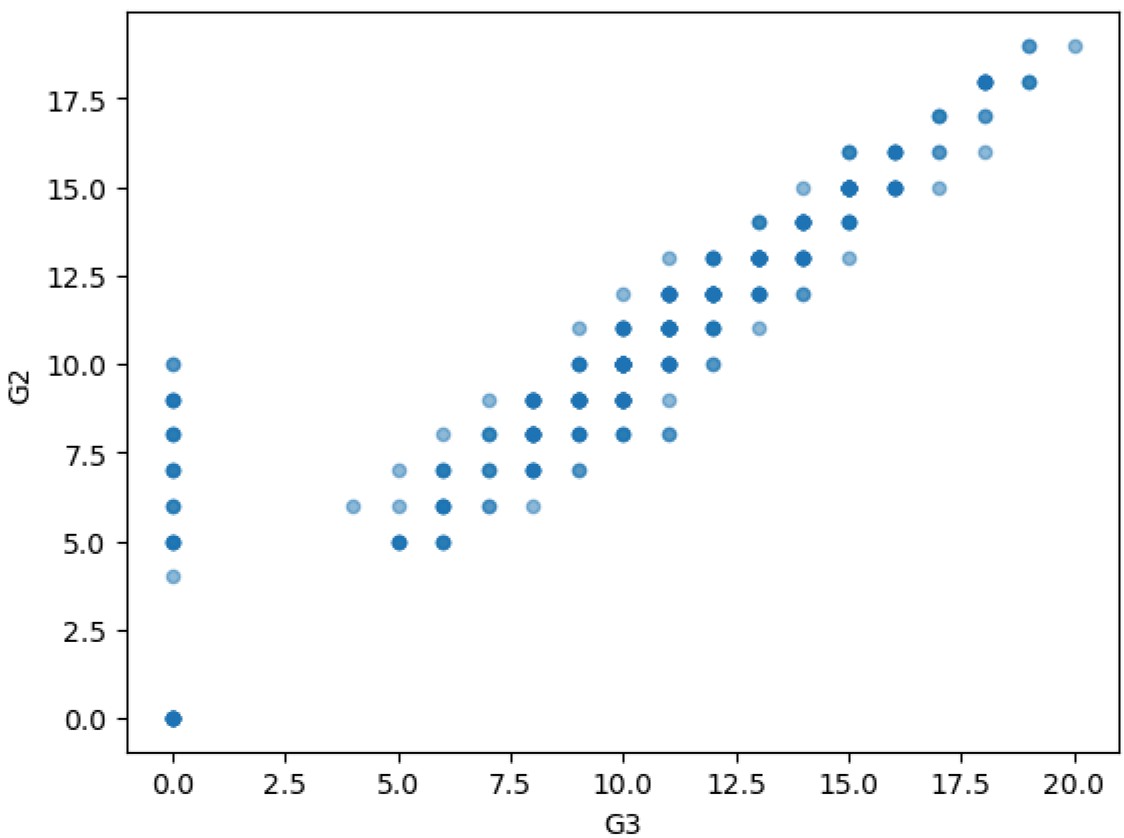
In [22]:
```
### Study correlations between attributes
# Correlation: we can pick a features and see how other features correlate with it.
# A score of -1 represents a negative correlation, while +1 represents a high correlat
# between two features. A value closer to 0 shows that there is little to no correlati
corr_matrix = student_info.corr(numeric_only=True)
```

In [23]:
```
# Let's look and see how the final grade correlates with each feature.
# The negative values just mean that the relationship is inversely related.
# If one goes up, the other goes down and vice versa.

corr_matrix['G3'].sort_values(ascending=False)
```

Out[23]:
```
G3              1.000000
G2              0.904868
G1              0.801468
Medu            0.217147
Fedu            0.152457
studytime       0.097820
absences_G3     0.067294
famrel          0.051363
absences_G1     0.012485
absences_G2     0.012485
freetime        0.011307
Walc           -0.051939
Dalc           -0.054660
health         -0.061335
traveltime     -0.117142
goout          -0.132791
age            -0.152762
failures       -0.360415
Name: G3, dtype: float64
```

In [24]:
```
# Let's also plot the corr matrix. A visual is always a good idea. We'll just
# choose the attribute that is most closely related to the final grade, term2 grade (G

from pandas.plotting import scatter_matrix

student_info.plot(kind='scatter', x="G3", y="G2", alpha=0.5)
```

Out[24]:
```
<Axes: xlabel='G3', ylabel='G2'>
```



In [25]:
```
# Interesting! We have a great visual of the positive relationship between the
# student's G2 grade and their G3 or final grade.
```

In [26]:
```
## PREPARE THE DATA
```

In [30]:
```
### Fill in missing values
# I want to remove null values from my features set and replace them with the mean val
# I will use an imputer from the SciKit library that will replace the null values from
# numerical objects and replaces it with the mean of those values.
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='median')
student_num = X_train.select_dtypes(include=[np.number])

imputer.fit(student_num)
X = imputer.transform(student_num)
```

In [31]:
```
### Ordinal Encoder
# Now, lets gather the categorical or object attributes and convert them to numeric.
# I will use another class from Scikit termed OrdinalEncoder.
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
student_cat = X_train.select_dtypes(exclude=[np.number])
student_cat_encoded = ordinal_encoder.fit_transform(student_cat)
```

In [32]:
```
### OneHotEncoder
# The one hot encoder works on categorical data by creating binary attributes
# per category. For instance, when looking at categorical variables, the encoder
# uses a 1 or 0 to show whether the category is present or not.
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
student_cat_1hot = cat_encoder.fit_transform(student_cat_encoded)
student_cat_1hot.toarray()
```

Out[32]:
```
array([[0., 1., 1., ..., 1., 0., 1.],
       [0., 1., 1., ..., 1., 0., 1.],
       [1., 0., 1., ..., 1., 0., 1.],
       ...,
       [1., 0., 0., ..., 1., 0., 1.],
       [1., 0., 0., ..., 1., 0., 1.],
       [0., 1., 0., ..., 1., 0., 1.]])
```

In [33]:
```
### Custom Transformer

# This customer transformer will perform by combining certain attributes.
# I want to create one that, if True, will combine the sum ofthe absence columns
# and create a new colum with that sum.
# Then those individual columns will be dropped.
absences_G1, absences_G2, absences_G3 = 0,1,2
G2, G3 = 8,9
from sklearn.base import BaseEstimator, TransformerMixin

class StudentTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, drop_G1_G2=True):
        self.drop_G1_G2 = drop_G1_G2

    def fit(self, X, y=None):
        return self

    def transform(self, X):
         if self.drop_G1_G2:
```

```
            X=np.delete(X, [8,9], 1)

        total_absences = X[:,absences_G1] + X[:, absences_G2] + X[:, absences_G3]
        X= np.delete(X, [0,1,2], 1)


        return np.c_[X]
```

In [34]:
```
attr_adder = StudentTransformer(drop_G1_G2 = False)
student_extra_attributes = attr_adder.transform(X_train.values)
```

In [35]:
```
### Feature Scaling
# Now, I will create pipelines for the numerical columns in the student
# training set. Pipelines help in the transformation process by keeping
# the processes occuring in the right order. We do it for both numerical and
# categorical attributes in the training data sets.
# The standard scaler in this pipeline will ensure that values stay within a range
# by subtracting the mean by each value and then dividing it by the standard
# deviation. Then there is unit variance.


from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline_drop = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder', StudentTransformer()),
    ('std_scaler', StandardScaler()),
])

student_num_tr_drop= num_pipeline_drop.fit_transform(student_num)
```

In [36]:
```
### Create Transformation Pipelines

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attr_adder',(StudentTransformer(drop_G1_G2 = False))),
    ('std_scaler', StandardScaler()),
])

student_num_tr = num_pipeline.fit_transform(student_num)
```

In [37]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
```

In [39]:
```
### Column Transformer
# The column transformer below can now take both numerical and categorical
# attributes and transform them in one transformation. Our above pipelines
# made this process easier for the column transformer to work in this way.
# I will separate numerical and categorical attributes from the X_training
# set once more then apply them alongside their respective pipelines from above.
from sklearn.compose import ColumnTransformer

numerical_attributes = list(student_num)
categorical_attributes = X_train.select_dtypes(exclude=[np.number]).columns

preprocessing_drop = ColumnTransformer([
       ("num", num_pipeline_drop, numerical_attributes),
```

```
        ("cat", OneHotEncoder(), categorical_attributes),
    ],
 )

### Prepared data excluding G1/G2
X_train_prepared_drop = preprocessing_drop.fit_transform(X_train)
```

In [40]:
```
preprocessing = ColumnTransformer([
        ("num", num_pipeline, numerical_attributes),
        ("cat", OneHotEncoder(), categorical_attributes),
    ],
 )


### Prepared data including G1/G2
X_train_prepared = preprocessing.fit_transform(X_train)
```

In [41]:
```
### Shape of transformed training set with G1 and G2 present
print(X_train_prepared.shape)
```

```
(316, 17)
```

In [42]:
```
### Shape of transformed training set with G1 and G2 absent
print(X_train_prepared_drop.shape)
```

```
(316, 15)
```

In [43]:
```
## EXPLORE MANY DIFFERENT MODELS AND SHORTLIST PROMISING MODELS
```

In [44]:
```
### 1: We will start with a basic Linear Regression Model

# This model will run the X_train_prepared data that still contains G1 & G2

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train_prepared, y_train)
```

Out[44]:
```
▾ LinearRegression
LinearRegression()
```

In [45]:
```
### Regression with G1 and G2
# Now, let's test some data from our training set to see how well it predicts.

some_data = X_train.iloc[:5]
some_labels = y_train.iloc[:5]
some_data_prepared = preprocessing.transform(some_data)
print("Predictions:", lin_reg.predict(some_data_prepared))

# Not an awful set of predictions. The final grade is averaged at 10.4 and the median
```

```
Predictions: [12.79406988 14.22648735  5.02319364  8.68213029  8.26442581]
```

In [46]:
```
# Now, let's run a calculation of the RMSE (root mean squared error).

from sklearn.metrics import mean_squared_error
student_predictions = lin_reg.predict(X_train_prepared)
lin_mse = mean_squared_error(y_train, student_predictions)
lin_rmse = np.sqrt(lin_mse)
```

```
lin_rmse

# This root_mean_squared error is stating that the error +- 1.828
# Because the actual value is quite small, this may be too much of a an error window.
# We will do some more calcs before deciding.
```

Out[46]: 1.8286639984513686

In [47]:
```
### Cross_Validation with G1/G2
# Cross-validation works by separating the training data yet again into numerous
# subsets. Then, it performs the decision tree regressor on all of those subsets
# returns the scores of each. Since the dataset is already small, I will program
# the cross-val to separate the training data into 4 subsets.

from sklearn.model_selection import cross_val_score

scores = cross_val_score(lin_reg, X_train_prepared, y_train,
                         scoring="neg_mean_squared_error", cv=4)
lin_rmse_scores = np.sqrt(-scores)


def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
display_scores(lin_rmse_scores)
```

```
Scores: [1.94408844 1.64665972 2.04978646 1.98163019]
Mean: 1.90554120369544
Standard deviation: 0.15419285109971254
```

In [48]:
```
# These are a bit higher than the original decision tree. It says that the mean error
# is 1.905 +- .1541
```

In [49]:
```
### Linear Regression without G1/G2
# Now we will run the same linear regression task on the G1 and G2 dropped data.
lin_reg_drop = LinearRegression()
lin_reg_drop.fit(X_train_prepared_drop, y_train)
```

Out[49]:
```
▾ LinearRegression
LinearRegression()
```

In [50]:
```
some_data_drop = X_train.iloc[:5]
some_labels_drop = y_train.iloc[:5]
some_data_prepared_drop = preprocessing_drop.transform(some_data_drop)
print("Predictions:", lin_reg_drop.predict(some_data_prepared_drop))

# These predictions are somewhate similar to the regression predictions from above.
```

```
Predictions: [12.78592761 11.87854446  5.1404602  11.23618363 10.58872911]
```

In [51]:
```
student_predictions_drop = lin_reg_drop.predict(X_train_prepared_drop)
lin_mse_drop = mean_squared_error(y_train, student_predictions_drop)
lin_rmse_drop = np.sqrt(lin_mse_drop)
lin_rmse_drop

# Wow, that mean squared error goes up significantly when G1 and G2 are dropped from t
# data set.
```

Out[51]:
```
4.082512889940057
```

In [52]:
```
### Cross_Validation without G1/G2
scores_drop = cross_val_score(lin_reg_drop, X_train_prepared_drop, y_train,
                        scoring="neg_mean_squared_error", cv=4)
lin_rmse_scores_drop = np.sqrt(-scores_drop)


def display_scores(scores_drop):
    print("Scores:", scores_drop)
    print("Mean:", scores_drop.mean())
    print("Standard deviation:", scores_drop.std())
display_scores(lin_rmse_scores_drop)

# It seems that keeping the G1 and G2 columns lessen the error window.
```

```
Scores: [4.17897699 4.3569983  4.63869687 4.33120778]
Mean: 4.376469985909966
Standard deviation: 0.16597758691556178
```

In [53]:
```
### Decision Tree Regressor with G1/G2
# Now, we will train a decision tree regressor.
# This regressor is able to find
# more complex, non-linear relationships.
# Let's start with the training data that still has G1 and G2 present.
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(X_train_prepared, y_train)
```

Out[53]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [54]:
```
# Now let's run our training set through the model and see what comes up.

student_predictions = tree_reg.predict(X_train_prepared)
tree_mse = mean_squared_error(y_train, student_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[54]:
```
0.0
```

In [55]:
```
# We received 0.0 error window! This seems great. However, it has been stated that
# a zero error may actually be a case of data overfitting. So, we will do a cross-vali
# before making any calls.
```

In [56]:
```
### Cross_Validation with G1/G2
# I'd like to perform a cross_val_score on this regression to get a standard deviation
# This can help me decide if such a low root_mean_squared error is accurate or
# if the std dev will nulify it by being a hefty value.

scores = cross_val_score(tree_reg, X_train_prepared, y_train,
                        scoring="neg_mean_squared_error", cv=4)
tree_rmse_scores = np.sqrt(-scores)
```

In [57]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
display_scores(tree_rmse_scores)

# Hmmm, the scores are worse than the linear regression model. These values below
# are saying that the mean value is 2.4701 +- 0.3933.
# The std dev is quite high on top of a high RMSE.
```

```
Scores: [2.38932224 1.90601978 2.58525517 3.        ]
Mean: 2.4701492969906025
Standard deviation: 0.39330848516257927
```

In [58]:
```
### Decision Tree Regressor without G1/G2
# Let's run it again, but without G1 and G2 in the data.

tree_reg_drop = DecisionTreeRegressor()
tree_reg_drop.fit(X_train_prepared_drop, y_train)
```

Out[58]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [59]:
```
student_predictions_drop = tree_reg_drop.predict(X_train_prepared_drop)
tree_mse_drop = mean_squared_error(y_train, student_predictions_drop)
tree_rmse_drop = np.sqrt(tree_mse_drop)
tree_rmse_drop

# This produces a decent error, not even 1, but the score is worse when the
# G1 and G2 columns are dropped.
```

Out[59]:
```
0.5979924219493594
```

In [60]:
```
### Cross_validation without G1/G2
scores_drop = cross_val_score(tree_reg_drop, X_train_prepared_drop, y_train,
                        scoring="neg_mean_squared_error", cv=4)
tree_rmse_scores_drop = np.sqrt(-scores_drop)
```

In [61]:
```
def display_scores(scores_drop):
    print("Scores:", scores_drop)
    print("Mean:", scores_drop.mean())
    print("Standard deviation:", scores_drop.std())
display_scores(tree_rmse_scores_drop)

# Yikes, the root mean squared error definitely increases when the G1 and G2 columns
# are dropped from the training data. The upside is that the standard deviation is 0.1
```

```
Scores: [6.38951871 6.36147423 6.55647327 6.27633691]
Mean: 6.395950783147308
Standard deviation: 0.1016180977246936
```

In [62]:
```
### Random Forest Regression with G1/G2
# Now, we will try another regression model called RandomForestRegressor
# This is a good model because it trains many decisiontrees and averages the predictio
# This model builds models on top of other models and pushes machine learning
# algorithms further than a simple model.
# We will again run the dataset that still has G1 and G2 present.
```

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(X_train_prepared, y_train)
forest_reg_preds = forest_reg.predict(X_train_prepared)
forest_mse = mean_squared_error(y_train, forest_reg_preds)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[62]: 0.762575582475168

In [63]:
```
display_scores(forest_rmse)

# This has, so far, been the best group in regards to the RMSE and standard deviation.
```

```
Scores: 0.762575582475168
Mean: 0.762575582475168
Standard deviation: 0.0
```

In [64]:
```
### Cross_Validation with G1/G2
scores_forest = cross_val_score(forest_reg, X_train_prepared, y_train,
                        scoring="neg_mean_squared_error", cv=4)
forest_rmse_scores = np.sqrt(-scores)
```

In [65]:
```
def display_scores(scores_forest):
    print("Scores:", scores_forest)
    print("Mean:", scores_forest.mean())
    print("Standard deviation:", scores_forest.std())
display_scores(forest_rmse_scores)
```

```
Scores: [2.38932224 1.90601978 2.58525517 3.        ]
Mean: 2.4701492969906025
Standard deviation: 0.39330848516257927
```

In [66]:
```
# Oh! This model returns a decent RMSE score. There is also a low std deviation which
# This model has great potential to be the best one so far.
```

In [67]:
```
### Random Forest Regressor without G1/G2
# Let's check it out using the data set that dropped G1 and G2.

forest_reg_drop = RandomForestRegressor()
forest_reg_drop.fit(X_train_prepared_drop, y_train)
forest_reg_preds_drop = forest_reg_drop.predict(X_train_prepared_drop)
forest_mse_drop = mean_squared_error(y_train, forest_reg_preds_drop)
forest_rmse_drop = np.sqrt(forest_mse_drop)
forest_rmse_drop
```

Out[67]: 1.8089841665928332

In [68]:
```
display_scores(forest_rmse_drop)
```

```
Scores: 1.8089841665928332
Mean: 1.8089841665928332
Standard deviation: 0.0
```

In [ ]:
```
# The RMSE for randomforestregressor without the G1/G2 columns is lower than the alter
# This is very interesting. It was presumed at the beginning of this project
```

```
# that dropping those two columns, due to their similarity, would make for a more
# useful set of predictions.
```

In [71]:
```
### Cross_Validation without G1/G2

scores_forest_drop = cross_val_score(forest_reg_drop, X_train_prepared_drop, y_train,
                        scoring="neg_mean_squared_error", cv=4)
forest_rmse_scores_drop = np.sqrt(-scores)
```

In [72]:
```
def display_scores(scores_forest_drop):
    print("Scores:", scores_forest_drop)
    print("Mean:", scores_forest_drop.mean())
    print("Standard deviation:", scores_forest_drop.std())
display_scores(forest_rmse_scores_drop)

# The cross validation for random forest resgressor with and without G1/G2 are the sam
# I believe this regression task is th most efficient for this dataset.
```

```
Scores: [2.38932224 1.90601978 2.58525517 3.        ]
Mean: 2.4701492969906025
Standard deviation: 0.39330848516257927
```

In [73]:
```
### Transform testing data using pipelines
# I need to prepare my X_test data by running it through the column transformer
# like I did for my X_train data.

X_test_prepared = preprocessing.transform(X_test)
X_test_prepared_drop = preprocessing_drop.transform(X_test)
```

In [74]:
```
### Support Vector Regression with G1/G2
# Lastly, I will run the training data through a support vector regression model.
# Again, we will start with the full set of training data.

from sklearn.svm import SVR

svr= SVR().fit(X_train_prepared, y_train)
```

In [75]:
```
# I will run the mean absolute error on this data
# to see what the error margin is looking like
from sklearn.metrics import mean_absolute_error

y_pred = svr.predict(X_test_prepared)
mean_absolute_error(y_test, y_pred)

# This produces a good mean absolute error value, which represent the differences betw
# predicted values and actual values.
```

Out[75]:
```
1.496240186375362
```

In [76]:
```
# The default kernel for SVR is rbf. I want to try and run a linear kernel
# on the SVR and see what changes in our MAE.

svr_linear = SVR(kernel="linear").fit(X_train_prepared, y_train)
```

In [77]:
```
y_pred_linear = svr_linear.predict(X_test_prepared)
mean_absolute_error(y_test, y_pred_linear)
```

Out[77]:
```
1.0318923711991566
```

In [ ]:
```
# The mean absolute error in the linear SVR model above produced good results.
# An error like this means that my predictions will be equal to the actual values, +-
# this value.
```

In [78]:
```
### Support Vector Regression without G1/G2
# Let's try it with the training data that drops G1 and G2.
svr_drop = SVR().fit(X_train_prepared_drop, y_train)
```

In [79]:
```
y_pred_drop = svr_drop.predict(X_test_prepared_drop)
mean_absolute_error(y_test, y_pred_drop)
```

Out[79]:
```
3.2879136337675288
```

In [80]:
```
svr_linear_drop = SVR(kernel="linear").fit(X_train_prepared_drop, y_train)
```

In [81]:
```
y_pred_linear_drop = svr_linear_drop.predict(X_test_prepared_drop)
mean_absolute_error(y_test, y_pred_linear_drop)
```

Out[81]:
```
3.5444144757007052
```

In [ ]:
```
# The mean absolute error of the dropped data in the linear SVR model
# above is not as appealing as the MAE from the dataset containing G1/G2.
```

In [ ]:
```
## FINE TUNE YOUR MODELS AND COMBINE INTO A GREAT SOLUTION
```

In [ ]:
```
# Looking back at the regression models I chose, the RandomForestRegressor
# displayed decent error windows for both the data including G1 and G2 and the
# data not including those columns. So, I've decided to fine tune this model.
```

In [82]:
```
### Perform grid search on selected model - Random Forest Regressor
# with G1/G2
from sklearn.model_selection import GridSearchCV

param_grid = [
{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
{'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3,4]}
]

# instantiate grid search
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
scoring='neg_mean_squared_error',
return_train_score=True)

# run grid search
grid_search.fit(X_train_prepared, y_train)
```

Out[82]:
```
▸ GridSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

In [83]:
```
grid_search.best_params_
```

Out[83]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [84]:
```
grid_search.best_estimator_
```

Out[84]:
```
RandomForestRegressor
RandomForestRegressor(max_features=8, n_estimators=30)
```

In [85]:
```
# If we look at the combination for the RFR from directly above (8, 30), we can
# see that this combination below returns a RMSE score of around 2.

cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
3.3331754261539785 {'max_features': 2, 'n_estimators': 3}
2.9215269878055534 {'max_features': 2, 'n_estimators': 10}
2.6610622208710284 {'max_features': 2, 'n_estimators': 30}
2.557688410775198 {'max_features': 4, 'n_estimators': 3}
2.2995053168846886 {'max_features': 4, 'n_estimators': 10}
2.30727521208914 {'max_features': 4, 'n_estimators': 30}
2.5669410799503187 {'max_features': 6, 'n_estimators': 3}
2.1421609814939027 {'max_features': 6, 'n_estimators': 10}
2.1867912810101338 {'max_features': 6, 'n_estimators': 30}
2.405937687306723 {'max_features': 8, 'n_estimators': 3}
2.018990691582904 {'max_features': 8, 'n_estimators': 10}
2.0174725640605904 {'max_features': 8, 'n_estimators': 30}
3.0680821871127097 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
2.8440679876286303 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
2.8880140235116585 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
2.593714590927212 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
2.6868511070121106 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
2.3253824996850025 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [86]:
```
# Let's analyze the model and the error score.
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[86]:
```
array([0.03781001, 0.02285769, 0.02303032, 0.02304433, 0.01751941,
       0.22687734, 0.56963396, 0.01306828, 0.03043286, 0.00335987,
       0.00658886, 0.00450704, 0.00351335, 0.00588826, 0.00355814,
       0.00210024, 0.00621005])
```

In [87]:
```
extra_attributes = ['total_absences']
cat_encoder = preprocessing.named_transformers_["cat"]
cat_one_hot_attributes = list(cat_encoder.categories_[0])
attributes = numerical_attributes + extra_attributes + cat_one_hot_attributes
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[87]:
```
[(0.569633956227654, 'Medu'),
 (0.22687733614560654, 'health'),
 (0.03781001041888744, 'absences_G1'),
 (0.03043286209734174, 'G1'),
 (0.02304432687957861, 'failures'),
 (0.02303031765028139, 'absences_G3'),
 (0.022857690211420978, 'absences_G2'),
 (0.017519413251157, 'Walc'),
 (0.013068278003411597, 'famrel'),
 (0.006588855934377171, 'studytime'),
 (0.005888256895917232, 'F'),
 (0.004507043421856271, 'age'),
 (0.0035581388906217866, 'M'),
 (0.0035133471506143934, 'total_absences'),
 (0.003359868029521683, 'G2')]
```

In [ ]:
```
# Because the scores always got worse and the models less effective when G1 and G2 wer
# I decided to use the dataset that included these two columns for my final model.
# Here we can see the impact on final grades that the mother's education has.
```

In [88]:
```
### Evaluate Test Set

final_model = grid_search.best_estimator_

final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
final_rmse
```

Out[88]:
```
2.2028111434927693
```

In [ ]:
```
## PRESENT SOLUTION
```

In [ ]:
```
# After running my model in full and seeing the ouput of all features and
# predictions, I was then able to select the features that showed to have more
# of an impact on the final grades of students.
### Feature Selection
# This is where my feature selection occure, as opposed to at the beginning of the
# project.

# The RandomForestRegressor model provided small error windows for both the data set
# including the G1 and G2 columns and the data set without those columns.
# This provides some flexibility for the user if they'd like to use one or the other.
# It was discovered that the mother's education status has a valueable impact on the
# student's final grades. This is a surprising outcome, but also makes some sense.
# Since mother's are typically the primary supporter of children (other than financial
# it seems accurate that her education status would affect her ability to help
# her children prepare for exams and work on projects or homework.

# The regression tasks that did not perform as well were Linear Regression, Decision T
# Regression, and Support Vector Regression.
# I believe the Linear model did not work as well because of it's sensitivity to outli
# The decision tree regressor had a very low set of error windows, but because decisio
# is known to overfit data, I wanted to choose a model that I felt would be more relia
# The support vector regression model perform very well on the data that still contain
# G1 and G2 columns. If this were our only set, I would've chosen SVR as my final mode
# However, when the two columns were removed, the absolute error increased substantial
# SVR does not perform as well on noisy data sets where targets are overlapping.
```

```
# Next steps based on these findings would be to magnify the effects of mother's educa
# on other important features. I would likely perform more models on these features.
```