Finding the Secret to Academic Success

- Student name: Gamze Turan
- Student pace: self paced
- Scheduled project review date/time: September, 28 2022 / 11:30 am
- Instructor name: Claude Fried

Fig.1 Student taking part in an exam (Image Credits - Photo by [Jeswin Thomas (https://unsplash.com/@jeswinthomas?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)](https://unsplash.com/@jeswinthomas?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText))

Overview

My goal is to examine the **effects of several socio-economic factors** on the **grades of secondary school students.** These analyses will allow us to predict student performance based upon a variety of features like study hours, alcohol comsumption, parent's education, etc.

I will run some interesting analysis like -

1. Effect of Alcohol consumption on Student performance.
2. Does being in a relationship affect high school student performance?
3. Impact of a parent's education on student grades. This can be very useful as a lot of schools use parent's education as a criteria for admission. This analysis can present an argument either in favor or against this approach.

Finally, I will create various machine learning models to predict student's final performance and then compare these models using a set of accuracy metrics.

I will be using **CRISP-DM (https://thinkinsights.net/data-literacy/crisp-dm/)** data scheme process for this project.

Business Understanding

My initial focus will be on understanding whether **alcohol comsumption** can have an **impact on student performance** or not. In United States, one in four individuals between the ages of 12 and 20 reported ninge drinking alcohol on a monthly basis (source:niaaa.nih.gov (https://www.niaaa.nih.gov/publications/brochures-and-fact-sheets/underage-drinking)).Underage drinking has become a very serious health concern and binge drinking by young people poses enormous health and safety risks.

There has been a steady rise in the number of teenagers found driving under the influence and this behaviour often leads to serious accidents resulting in property damage, injuries, and sometimes deadth as well (source:drugfree.org (https://drugfree.org/drug-and-alcohol-news/23-of-teenssurveyed-admit-to-driving-under-influence-of-alcohol-or-drugs/)). Thus, it is critical that we implement prevention strategies during early adolescence to prevent escalation in alcohol consumption.

I will run similar analyses for multiple factors. Based upon my observations, I will try to build a **persona of a model student** who has the highest chance of academic success.

This information can then be used by school districts, academic counsellors, and parents to help guide their students towards better school performance.

Data Understanding

I will use the UCI Machine Learning Repository: Student Performance Data Set (http://archive.ics.uci.edu/ml/datasets/Student+Performance) to present interesting insights about the factors that predict student performance.

> Source: 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. Available at: Web Link (www3.dsi.uminho.pt/pcortez/student.pdf).

This data contains student achievement information in secondary education of 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. Two datasets are provided regarding the performance in two distinct subjects: Mathematics(`student-mat.csv`) and Portuguese language(`student-por.csv`).

In Cortez and Silva, 2008 (www3.dsi.uminho.pt/pcortez/student.pdf), the two datasets were modeled under binary/five-level classification and regression tasks.

Import Libraries

In [1]:

```
# Data handling
import numpy as np
import pandas as pd

# Libraries for plotting
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
from plotly import tools

# Statistical tests
import statsmodels.api as sm
import scipy

# Standard ML Models for comparison
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

# Sklearn utilities
from sklearn.model_selection import (
    cross_val_predict,
    KFold,
    cross_val_score,
    GridSearchCV,
    train_test_split,
    RandomizedSearchCV,
)
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    r2_score,
    accuracy_score)

from sklearn.preprocessing import MinMaxScaler
import xgboost as xgb
# ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

Data Preparation

In [2]:
```
# Load dataset for two subjects, Math and Portuguese
mat_df = pd.read_csv("data/student-mat.csv", sep=';')
por_df = pd.read_csv("data/student-por.csv", sep=';')
```

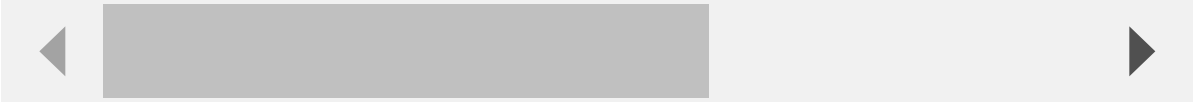
Let's take a look at both the datasets

In [3]:
```
#students studying portugese
por_df.head()
```

Out[3]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	f
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	

5 rows × 33 columns



In [4]:
```
1 por_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

Observations:

- There are no missing values in the dataset containning info about students studying portugese
- All the data types seem okay and don't need any explicit cleaning.We will dive further into the columns during EDA.

In [5]:
```
1 mat_df.head()
```

Out[5]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	f
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	

5 rows × 33 columns

In [6]:
```
1 mat_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
```

Observations:

- While portugese has a total of 649 students, mathematics has 395 students. Both dataset have the same columns.
- No null values in this datasets as well.

Now, we will merge both the dataset together.

In [7]:
```
# Merging both the dataframes using concatenation
student_df = pd.concat([mat_df, por_df], ignore_index=True)
student_df.head()
```

Out[7]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	f
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	

5 rows × 33 columns

In [8]:
```
print('Total number of students:' ,len(student_df))
```

```
Total number of students: 1044
```

Let's rename some of the columns to make the dataset unambiguous.

In [9]:
```
1 list(student_df)
```

Out[9]:
```
['school',
 'sex',
 'age',
 'address',
 'famsize',
 'Pstatus',
 'Medu',
 'Fedu',
 'Mjob',
 'Fjob',
 'reason',
 'guardian',
 'traveltime',
 'studytime',
 'failures',
 'schoolsup',
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic',
 'famrel',
 'freetime',
 'goout',
 'Dalc',
 'Walc',
 'health',
 'absences',
 'G1',
 'G2',
 'G3']
```

In [10]:
```
1 student_df.rename(columns = {'famsize':'family_size', 'Pstatus':'parent_
2                              'Mjob':'mother_job', 'Fjob':'father_job','t
3                              'schoolsup':'school_support', 'famsup':'fam
4                              'famrel':'family_quality', 'freetime':'free_
5                              'Walc':'weekend_alcohol_usage', 'G1':'perio
```

In [11]:
```
1 student_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1044 entries, 0 to 1043
Data columns (total 33 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   school                 1044 non-null   object
 1   sex                    1044 non-null   object
 2   age                    1044 non-null   int64 
 3   address                1044 non-null   object
 4   family_size            1044 non-null   object
 5   parent_status          1044 non-null   object
 6   mother_education       1044 non-null   int64 
 7   father_education       1044 non-null   int64 
 8   mother_job             1044 non-null   object
 9   father_job             1044 non-null   object
 10  reason                 1044 non-null   object
 11  guardian               1044 non-null   object
 12  commute_time           1044 non-null   int64 
 13  study_time             1044 non-null   int64 
 14  failures               1044 non-null   int64 
 15  school_support         1044 non-null   object
 16  family_support         1044 non-null   object
 17  paid_classes           1044 non-null   object
 18  activities             1044 non-null   object
 19  nursery                1044 non-null   object
 20  desire_higher_edu      1044 non-null   object
 21  internet               1044 non-null   object
 22  romantic               1044 non-null   object
 23  family_quality         1044 non-null   int64 
 24  free_time              1044 non-null   int64 
 25  go_out                 1044 non-null   int64 
 26  weekday_alcohol_usage  1044 non-null   int64 
 27  weekend_alcohol_usage  1044 non-null   int64 
 28  health                 1044 non-null   int64 
 29  absences               1044 non-null   int64 
 30  period1_score          1044 non-null   int64 
 31  period2_score          1044 non-null   int64 
 32  final_score            1044 non-null   int64 
dtypes: int64(16), object(17)
memory usage: 269.3+ KB
```

In [12]:
```
# Look for missing values
student_df.isnull().sum()
```

Out[12]:
```
school                  0
sex                     0
age                     0
address                 0
family_size             0
parent_status           0
mother_education        0
father_education        0
mother_job              0
father_job              0
reason                  0
guardian                0
commute_time            0
study_time              0
failures                0
school_support          0
family_support          0
paid_classes            0
activities              0
nursery                 0
desire_higher_edu       0
internet                0
romantic                0
family_quality          0
free_time               0
go_out                  0
weekday_alcohol_usage   0
weekend_alcohol_usage   0
health                  0
absences                0
period1_score           0
period2_score           0
final_score             0
dtype: int64
```

Observations:

- No null values in the dataset.
- Target variable is the `final_score` columns.

The dataset is now ready for analysis. Our target variable is `final_score` and we have a total of 32 features.

As the data comes from a study, it is relatively clean and does not require extensive cleaning procedures. I will however need to perform formating options to get the data ready for machine learning.

I will look at them later on.

EDA - Exploratory Data Analysis

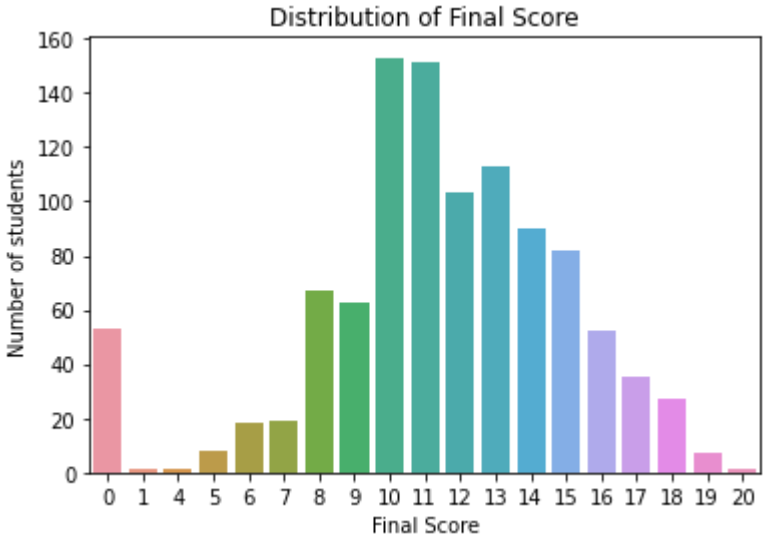
Target Variable (final_score)

In [13]:
```
# general statistics
student_df['final_score'].describe()
```

Out[13]:
```
count    1044.000000
mean       11.341954
std         3.864796
min         0.000000
25%        10.000000
50%        11.000000
75%        14.000000
max        20.000000
Name: final_score, dtype: float64
```

So, the grade lies between 0 and 20.The average grade is roughly 56% at 11.34. Let's look at the distribution of grades to find more information.

In [14]:
```
# final score distribution
tmp_plt = sns.countplot(x = 'final_score', data = student_df)
tmp_plt.axes.set_title('Distribution of Final Score')
tmp_plt.set_xlabel('Final Score')
tmp_plt.set_ylabel('Number of students')
#plt.savefig('images/final_score_distribution ')
plt.show()
```



Overall, the distribution looks like a normal distribution which is ideal.

But, there are a lot of students that score 0 in the final grade. This is intriguing. We can expect some zero values but definitely not so many.

In [15]:
```
1 print('Number of students that scored 0:',len(student_df[student_df['fin
```

```
Number of students that scored 0: 53
```

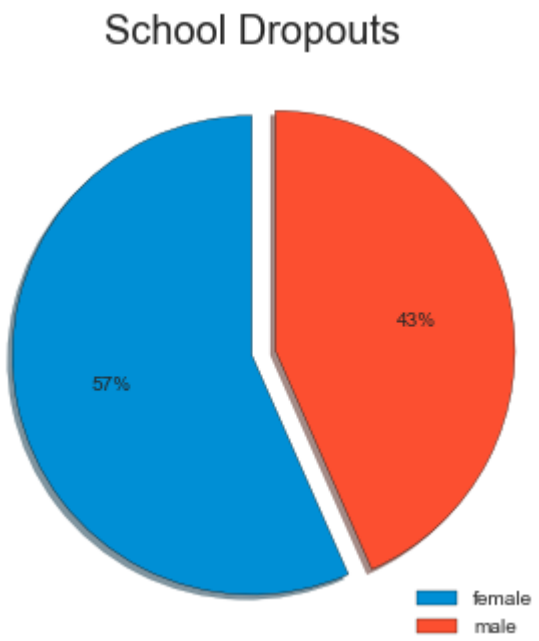
Reading the source paper (www3.dsi.uminho.pt/pcortez/student.pdf), we realized that this can be due to multiple reasons:

1. Student actually scored a zero on the exam
2. Student dropped out of school before the exam
3. School barred the student from taking the exam

Let's try and analyze this further. Who has a higher chance of scoring a zero - male or a female.

In [16]:

```
# Draw Pie-Chart of frequency distribution for internet access
plt.style.use('seaborn')
total_dropout = student_df[student_df['final_score']==0]['sex'].value_co
labels = 'female', 'male'
colors = ['#008fd5', '#fc4f30']
explode = (0, 0.1)
explode2 = (0.2, 0)
plt.tight_layout()
plt.pie(total_dropout.iloc[0], startangle=90, colors=colors, wedgeprops=
plt.legend(loc='best', labels=labels, fontsize='medium')
plt.title('School Dropouts',fontsize = 20)
#plt.savefig('images/school_dropouts')
plt.show()
```

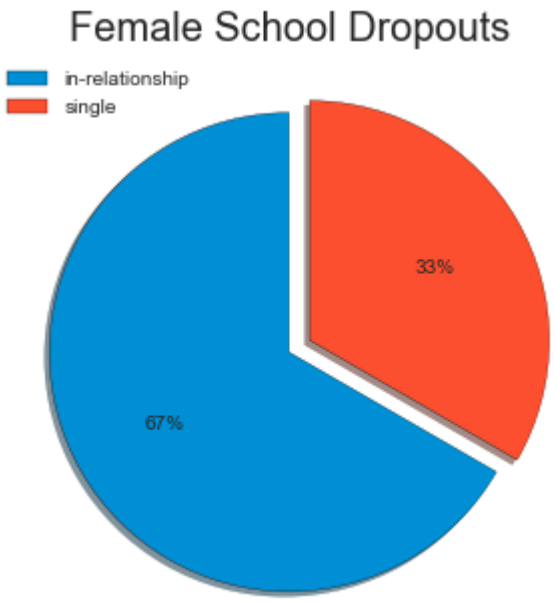


This is weird. More females are scoring 0 vs males. Usually females have better performance, then why are more females scoring 0 in their final exams. These must be dropouts. But what exactly is causing these females to drop out of high school?

Let's look at the relationship status of the females that are dropping out of school.

In [17]:

```
# Draw Pie-Chart of frequency distribution of relationship status of fem
plt.style.use('seaborn')
total_dropout = student_df[(student_df['final_score']==0)&(student_df['s
labels = 'in-relationship', 'single'
colors = ['#008fd5', '#fc4f30']
explode = (0, 0.1)
explode2 = (0.2, 0)
plt.tight_layout()
plt.pie(total_dropout.iloc[0], startangle=90, colors=colors, wedgeprops=
plt.legend(loc='best', labels=labels, fontsize='medium')
plt.title('Female School Dropouts',fontsize = 20)
#plt.savefig('images/Female_school_dropouts')
plt.show()
```



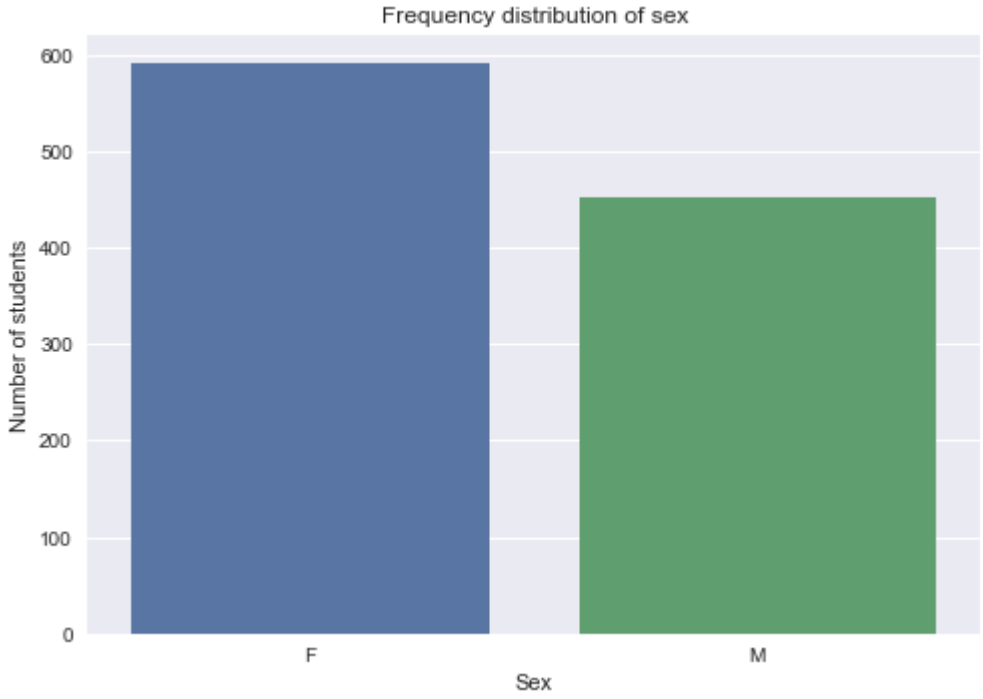
There are many reports (https://www.caf.com/en/knowledge/views/2021/01/teen-pregnancy-as-a-cause-of-school-dropout/) that suggest that as many as 36% of high-school dropouts can be attributed to teen pregnancy!

Having multiple reasons for a 0 final score might complicate things for us and we will consider

removing these values before creating our machine learning models.

Gender

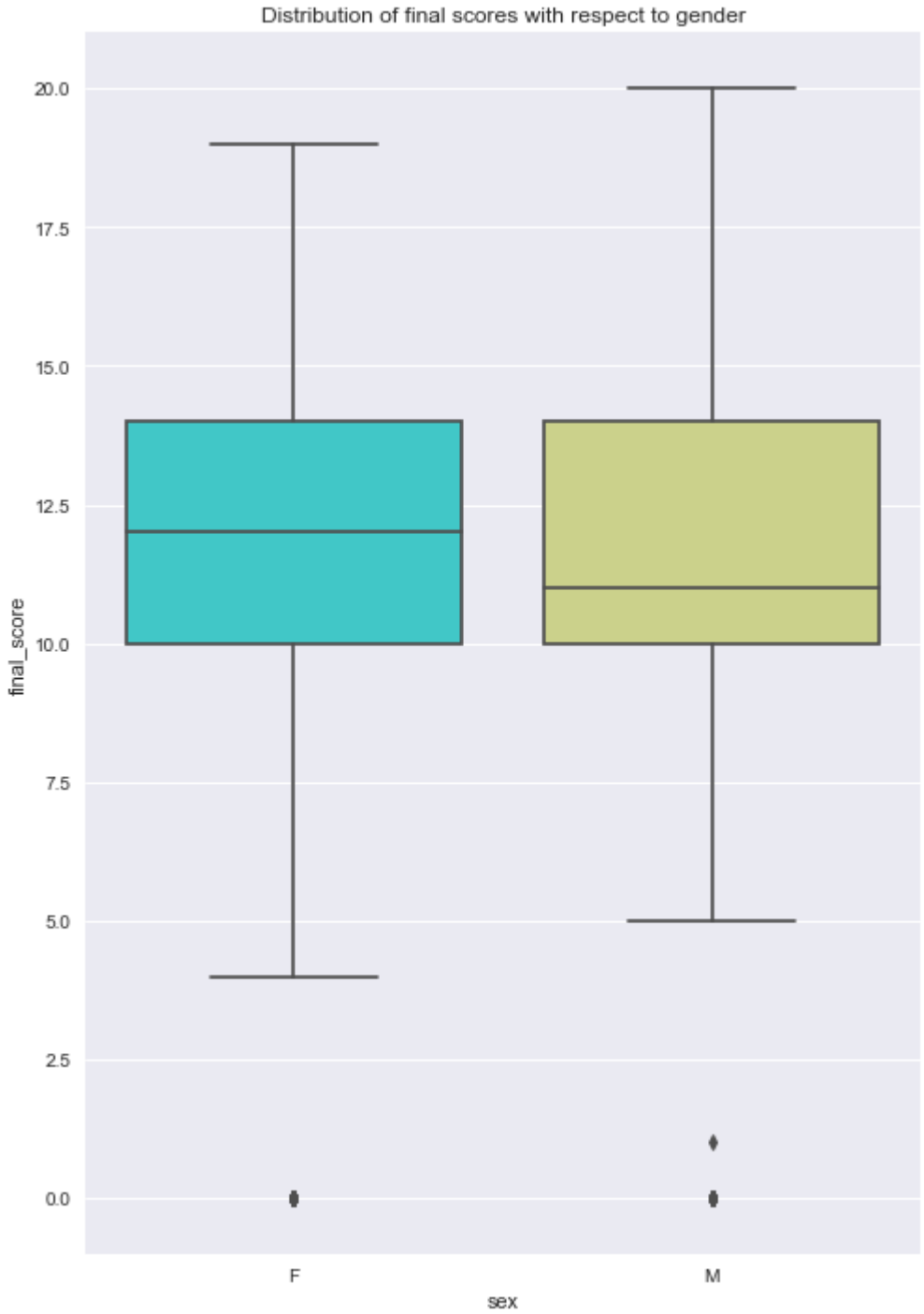
In [18]:
```
# creating frequency distribution of gender countplot
tmp_plt = sns.countplot(x = "sex", data = student_df)
tmp_plt.axes.set_title('Frequency distribution of sex')
tmp_plt.set_xlabel('Sex')
tmp_plt.set_ylabel('Number of students')
#plt.savefig('images/frequency_dist_sex')
plt.show()
```



There are a higher number of females in the dataset.Let's see if gender has any relationship with academic success.

In [19]:

```
plt.figure(figsize=(8, 12))
plt.title("Distribution of final scores with respect to gender")
sns.boxplot(y="final_score", x="sex", data = student_df, palette ="rainb
#plt.savefig('images/dist_final_gender')
```



From the graph, I can see that overall distribution of scores is very similar for both the genders. The median final score is slightly lower for males than females.

In [20]:
```
print('Median final score for males:', student_df[student_df['sex']=='M'
print('Median final score for females:', student_df[student_df['sex']=='
```

```
Median final score for males: 11.0
Median final score for females: 12.0
```

Let's see if this difference is significant by running a [t-test (https://www.investopedia.com/terms/t/t-test.asp)](https://www.investopedia.com/terms/t/t-test.asp):

In [21]:
```
male_student_scores = student_df[student_df['sex']=='M']['final_score'].
female_student_scores = student_df[student_df['sex']=='F']['final_score'
```

In [22]:
```
# Running a two-tailed t test https://docs.scipy.org/doc/scipy/reference/
t_value,p_value=scipy.stats.ttest_ind(male_student_scores,female_student_

print('Test statistic is %f'%float("{:.6f}".format(t_value)))
print('p-value for two tailed test is %f'%p_value)
```

```
Test statistic is -1.016422
p-value for two tailed test is 0.309664
```

In [23]:
```
alpha = 0.05
# testing for significance
if p_value<=alpha:
    print('Difference between male and female final scores is statistica
else:
    print('Difference between male and female final scores is not statis
```

```
Difference between male and female final scores is not statistically signif
icant
```

So, while the graph showed some difference between the final scores of both genders, this difference is not statistically significant.

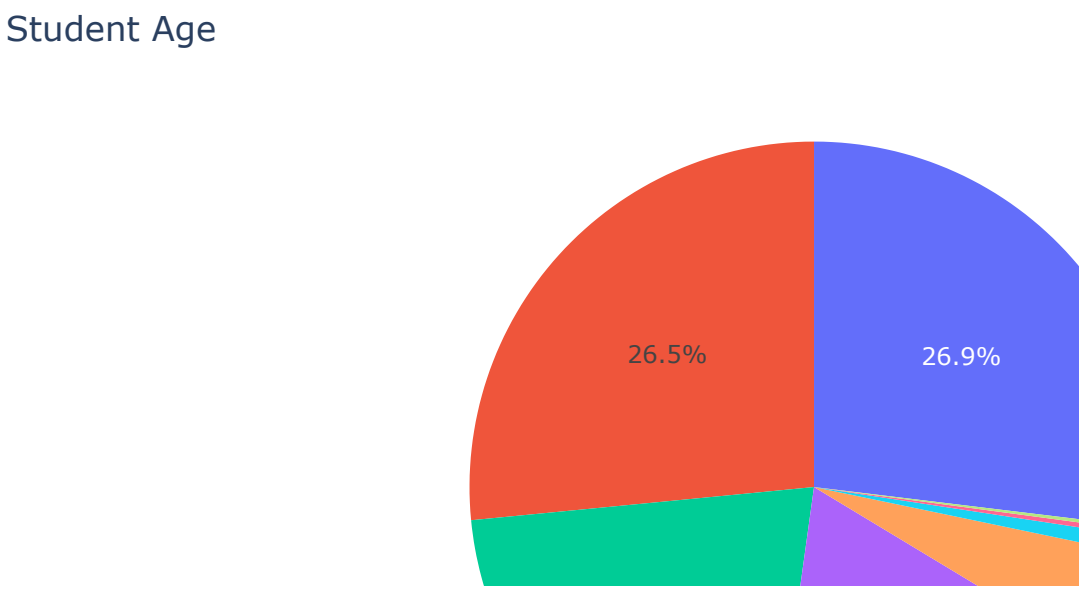
Age

In [24]:
```
# Frequency distribution of student age
student_df["age"].value_counts()
```

Out[24]:
```
16    281
17    277
18    222
15    194
19     56
20      9
21      3
22      2
Name: age, dtype: int64
```

In [25]:

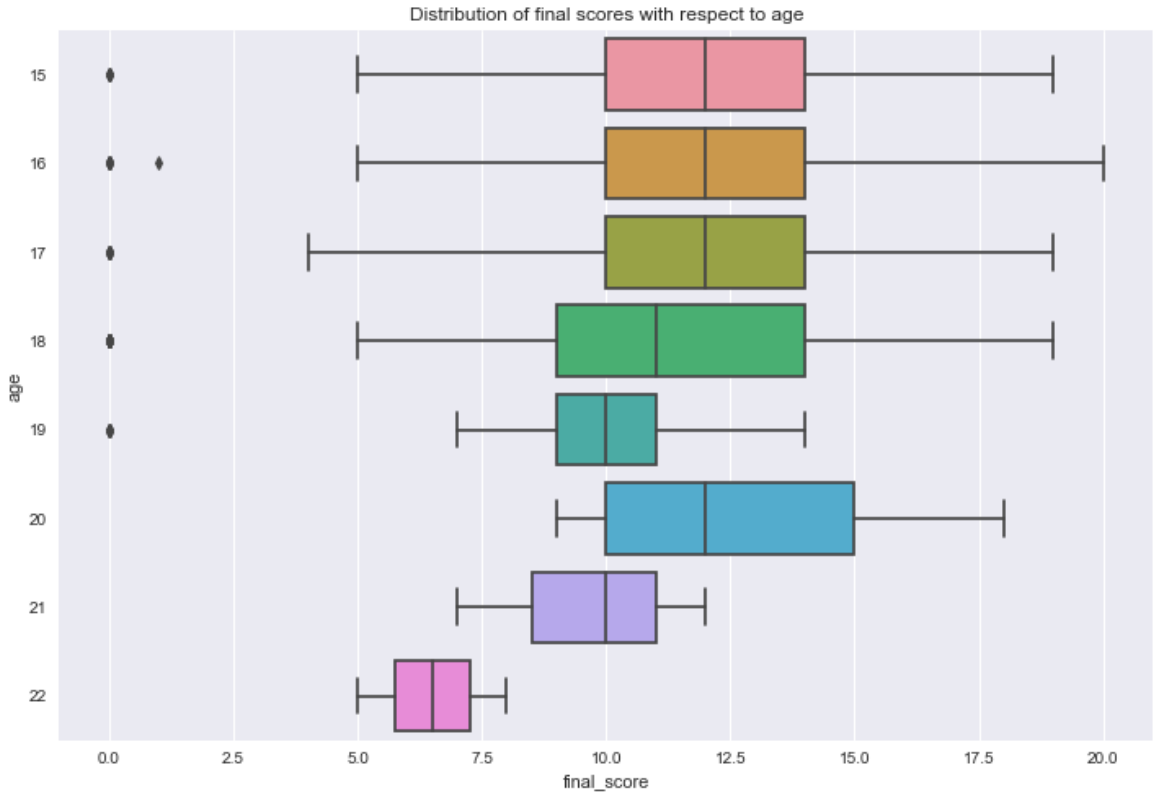
```
# Pie chart for age frequency distribution
ages = student_df["age"].value_counts().sort_index()

# all ages
labels = (np.array(ages.index))

# corresponding percentages of each age
sizes = (np.array((ages / ages.sum())*100))

# plotting pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=sizes)], layout=go.La
#plt.savefig('images/student_age')
py.iplot(fig, filename = "age")
```



Students are between 15 and 22 years old. 16 and 17 year olds are the most common. Let's see if age can determine academic success:

In [26]:

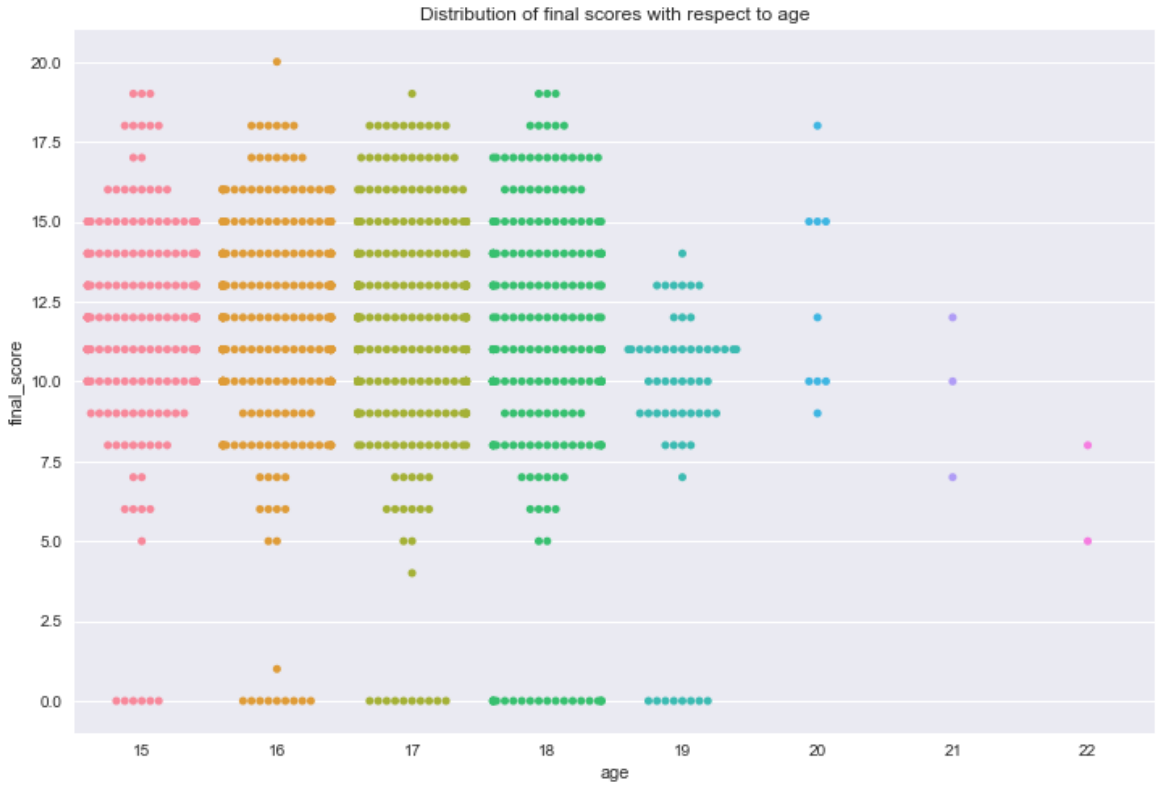
```
plt.figure(figsize=(12,8))
plt.title("Distribution of final scores with respect to age")
sns.boxplot(y="age", x="final_score", orient = 'h', data = student_df);
#plt.savefig('images/dist_final_age')
```



The boxplots seem to indicate that higher ages(20-22) result in poor grades.But we saw in value counts, that these 3 ages had only few datapoints.Let's look at the swarmplot here:

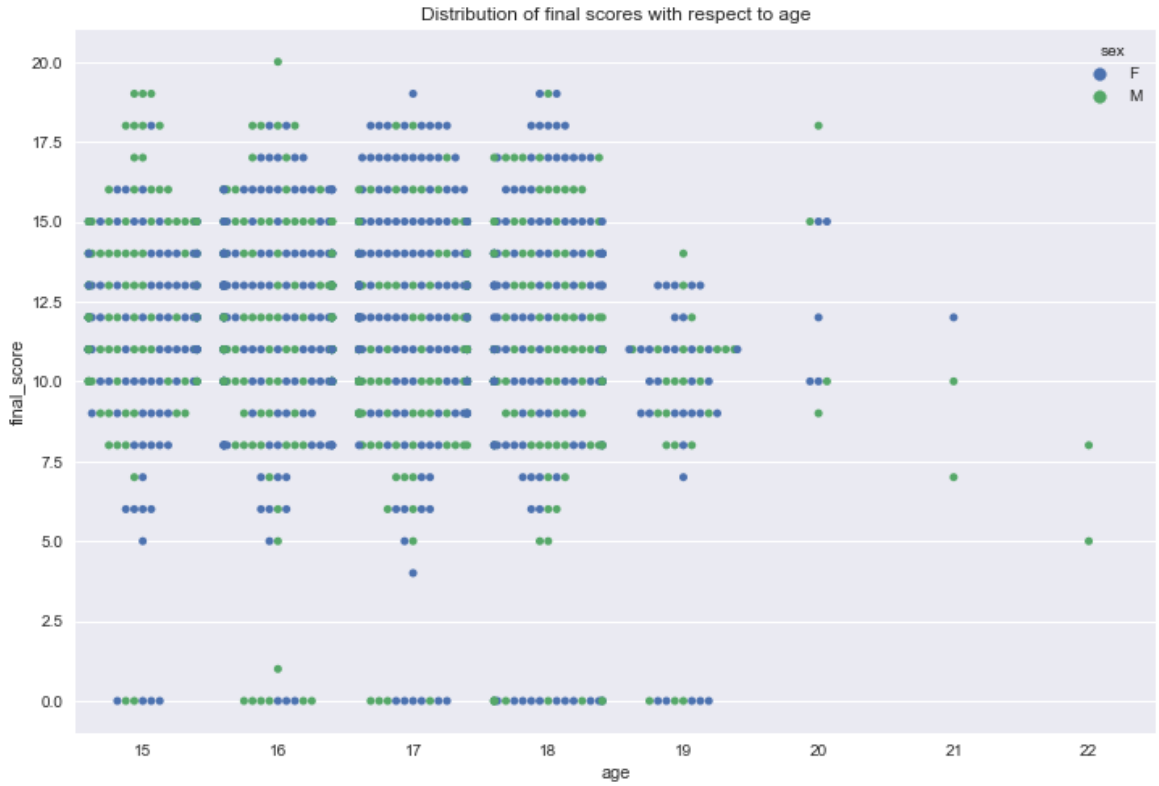
In [27]:

```
plt.figure(figsize = (12,8))
plt.title("Distribution of final scores with respect to age")
sns.swarmplot(y="final_score", x="age", data = student_df);
#plt.savefig('images/dist_final_age2')
```



Now it is clear that it's only lack of data in the last three ages (20,21 and 22). Otherwise, age does not seem to affect final score. I can look at age and sex together.

In [28]:
```
plt.figure(figsize = (12,8))
plt.title("Distribution of final scores with respect to age")
sns.swarmplot(y="final_score", x="age", hue="sex", data = student_df);
#plt.savefig('images/dist_final_age_sex')
```



The above graph indicates that there is no clear relation of age or gender with final score.

This is good news.While using machine learning for business applications, I must cafefully not to propogate any bias towards any demographic.Now, that these features are not directly related to the model, I can choose to remove them in my final model.

Study Time

According to [data documentation (https://archive.ics.uci.edu/ml/datasets/student+performance)](https://archive.ics.uci.edu/ml/datasets/student+performance), `studytime` column encodes weekly study time of student where:

- 1 -<2 hours
- 2 -2 to 5 hours
- 3 -5 to 10 hours
- 4 - >10 hours

In [29]:

```
# function to convert study time encoding to actual meaning
def convertor_function(study_time):
    if study_time == 1:
        return '<2 hours'
    elif study_time == 2:
        return '2 to 5 hours'
    elif study_time == 3:
        return '5 to 10 hours'
    elif study_time == 4:
        return '>10 hours'

student_df['new_study_time'] = student_df['study_time'].apply(convertor_

student_df[['study_time', 'new_study_time']].sample(10)
```

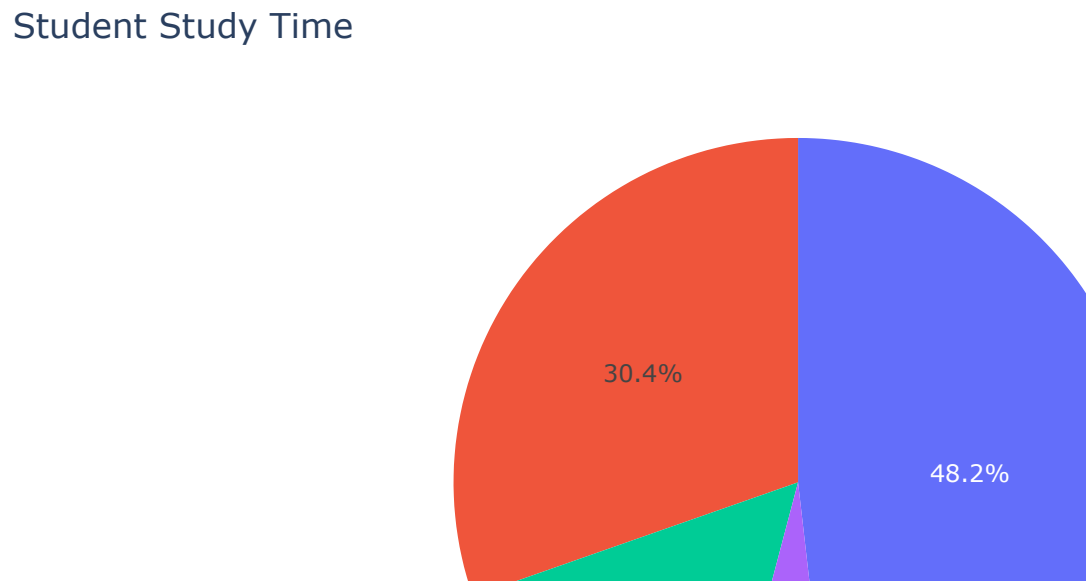
Out[29]:

	study_time	new_study_time
803	2	2 to 5 hours
523	1	<2 hours
121	4	>10 hours
648	2	2 to 5 hours
339	2	2 to 5 hours
794	3	5 to 10 hours
497	1	<2 hours
119	1	<2 hours
531	1	<2 hours
126	2	2 to 5 hours

Let's look at the distribution of study time.

In [30]:

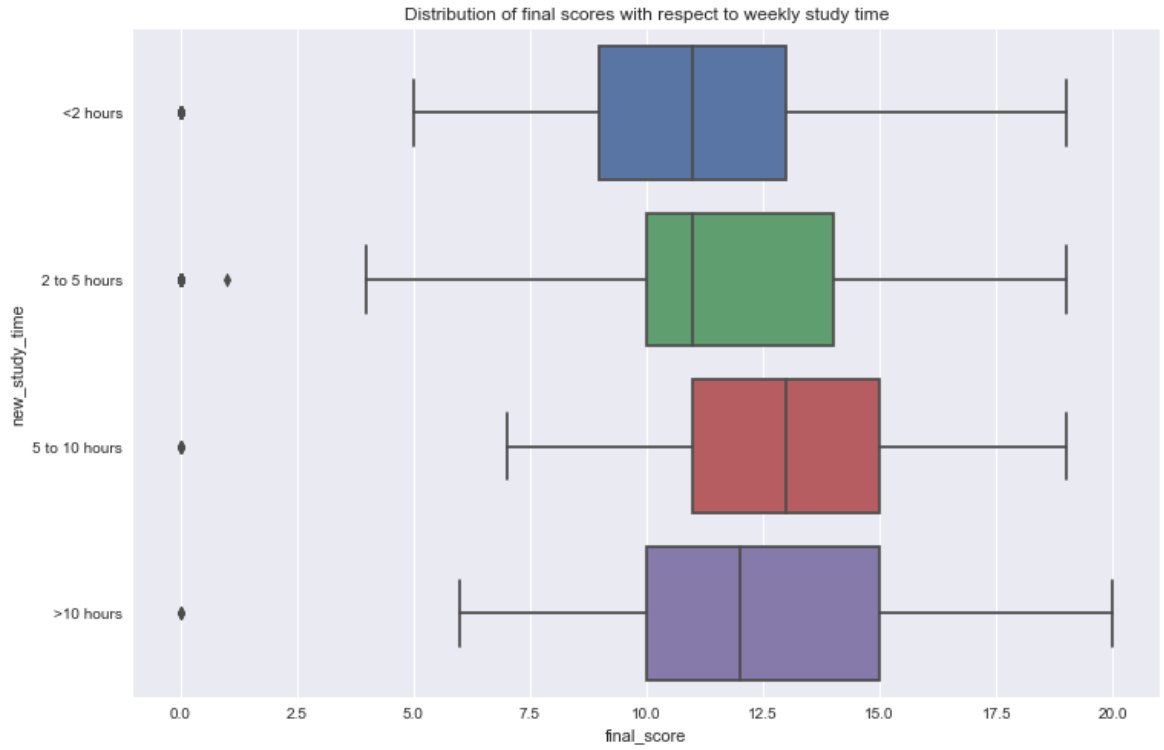
```
# Pie chart for study frequency distribution
study_times = student_df["new_study_time"].value_counts().sort_index()

# all study times
labels = (np.array(study_times.index))

# corresponding percentages of each age
sizes = (np.array((study_times / study_times.sum())*100))

# plotting pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=sizes)], layout=go.La
plt.savefig('images/study_time')
py.iplot(fig, filename="study_time")
```




```
<Figure size 576x396 with 0 Axes>
```

Almost 50% of the student in the dataset study 2-5 hours. Let's see if there is a relationship between study time and final score using boxplots:

In [31]:
```
plt.figure(figsize=(12,8))
o = ['<2 hours','2 to 5 hours','5 to 10 hours','>10 hours']
plt.title("Distribution of final scores with respect to weekly study tim
sns.boxplot(y="new_study_time", x="final_score", orient='h',order=o, dat
#plt.savefig('images/dist_final_weekly_st')
```



In [32]:
```
print('Average score of students studying <2 hours weekly', student_df[s
print('Average score of students studying 2 to 5 hours weekly', student_
print('Average score of students studying 5 to 10 hours weekly', student
print('Average score of students studying >10 hours weekly', student_df[
```

```
Average score of students studying <2 hours weekly 10.580441640378549
Average score of students studying 2 to 5 hours weekly 11.335984095427435
Average score of students studying 5 to 10 hours weekly 12.493827160493828
Average score of students studying >10 hours weekly 12.274193548387096
```

I see that as study time increases, the avarage score of the student increases. I can run a hypothesis test to check for significance of this result.

In [33]:
```
study_less_than_2_scores = student_df[student_df['new_study_time']=='<2
study_2_to_5_scores = student_df[student_df['new_study_time']=='2 to 5 h
study_5_to_10_scores = student_df[student_df['new_study_time']=='5 to 10
study_greater_than_10_scores = student_df[student_df['new_study_time']==
```

I will run a [One-Way ANOVA test (https://www.simplypsychology.org/anova.html)](https://www.simplypsychology.org/anova.html) to check if there is exists a statistically significant difference between the mean score of different study time.

In [34]:
```
# running a one-way ANOVA test https://docs.scipy.org/doc/scipy/referenc
t_value, p_value = scipy.stats.f_oneway(study_less_than_2_scores, study_

print('Test statistics is %f'%float("{:.6f}".format(t_value)))
print('p_value for ANOVA test is %f'%p_value)
```

```
Test statistics is 10.374445
p_value for ANOVA test is 0.000001
```

In [35]:
```
alpha = 0.05
#testing for significance
if p_value<=alpha:
    print('Difference between final scores for different study time is s
else:
    print('Difference between final scores for different study time is n
```

```
Difference between final scores for different study time is statistically s
ignificantly
```

Thus, I see that in the data, study time is a significant component for academic success and studying more will lead to better results.

Address (U - Urban, R- Rural)

`address` column contains information about student's home address type.

- "U" -urban
- "R" -rural

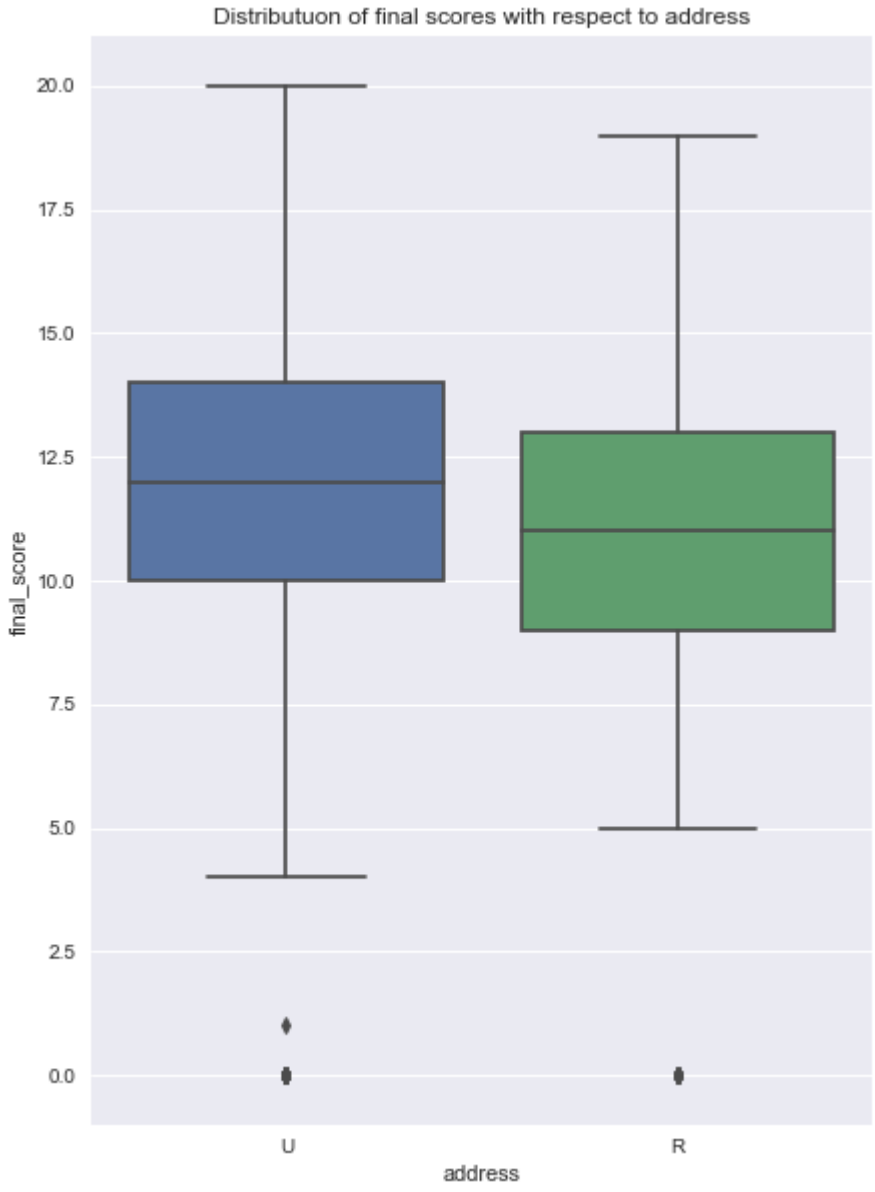
This might be an important feature predicting academic success.Urban students might have easier access to alcohol and other fun activities which can result in distraction.Rural students might have less incentive to go to school.Let's look at their frequency distribution first.

In [36]:
```
1 student_df['address'].value_counts()
```

Out[36]:
```
U    759
R    285
Name: address, dtype: int64
```

So, vast majority are urban students.Let's see if a direct relationship exists between address and final_grade.

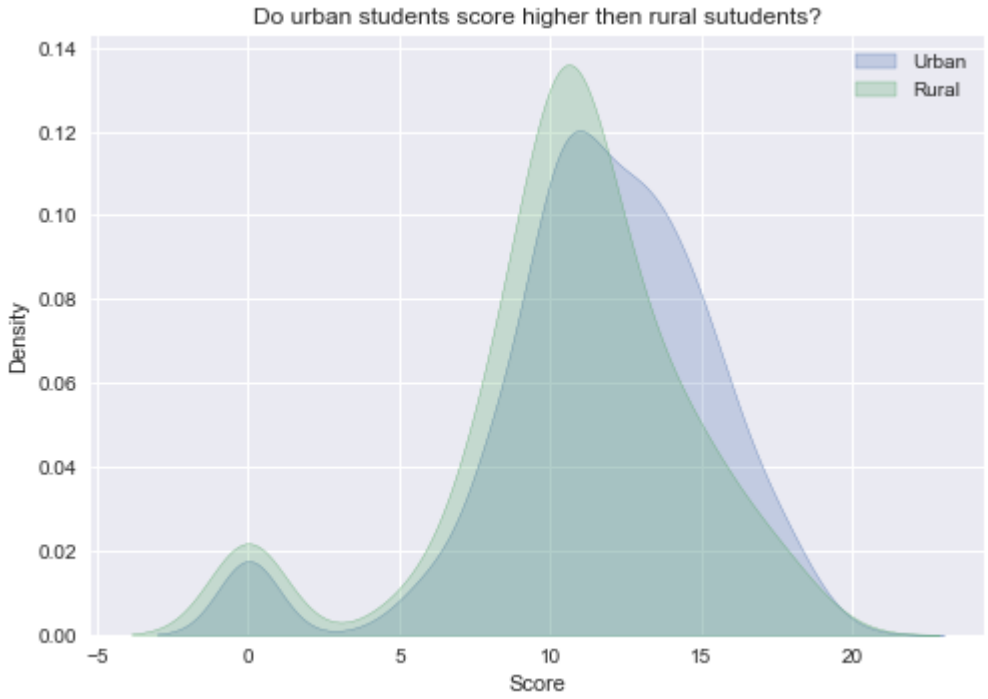
In [37]:
```
1 plt.figure(figsize=(7,10))
2 plt.title("Distributuon of final scores with respect to address")
3 sns.boxplot(y="final_score", x="address", data = student_df);
4 #plt.savefig('images/dist_final_address')
```



Boxplots indicate that students residing in urban areas tend to perform better than students

residing in rural areas.Let's look at the distributions further using KDE plots (https://seaborn.pydata.org/generated/seaborn.kdeplot.html) to better understand the differences.

In [38]:
```
sns.kdeplot(student_df[student_df['address'] == 'U']['final_score'], lab
sns.kdeplot(student_df[student_df['address'] == 'R']['final_score'], lab
plt.title( 'Do urban students score higher then rural sutudents?')
plt.xlabel('Score')
plt.ylabel('Density')
plt.legend()
#plt.savefig('images/urban_rural_student')
plt.show()
```



In this plot, both the distribution seem similar.Let's run a two-tailed t-test to check whether the difference in the mean score for urban students and the mean score for ruarl students is statiscally significant.

In [39]:
```
urban_student_scores = student_df[student_df['address'] == 'U']['final_s
rural_student_scores = student_df[student_df['address'] == 'R']['final_s
```

In [40]:
```
# Running a two-tailded t-test https://docs.scipy.org/doc/scipy/referenc
t_value, p_value = scipy.stats.ttest_ind(urban_student_scores, rural_stu

print('Test statistic is %f'%float("{:.6f}".format(t_value)))
print('p-value for two tailed test id %f'%p_value)
```

```
Test statistic is 3.825806
p-value for two tailed test id 0.000138
```

In [41]:
```
alpha = 0.05
#testing for significance
if p_value<=alpha:
    print('Difference between urban and rural final scores is statistaca
else:
    print('Difference between urban and rural final scores is not statis
```

```
Difference between urban and rural final scores is statistacally significan
t
```

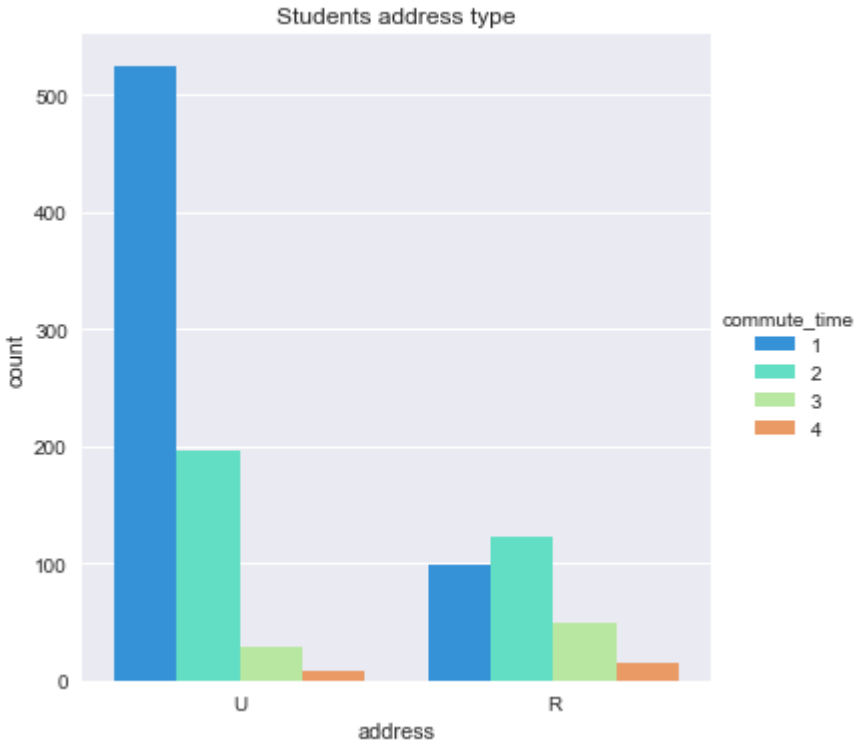
Now I know that Rural students tend to score less than Urban ones.Let's try and analyze a variety of factors that might cause this.One of them I believe is `commute_time` .

According to [data documentation (https://archive.ics.uci.edu/ml/datasets/student+performance)](), `commute_time` column encodes home to school travel time of a student where:

- 1 - <15 mins
- 2 - 15 to 30 mins
- 3 - 30 mins to 1 hour
- 4 - >1 hour

In [42]:

```
sns.catplot(x="address", kind="count", hue = "commute_time", palette = "
plt.title("Students address type")
#plt.savefig('images/student_address_type')
plt.show()
```



It seems like rural students tend to have higher commute times.

In [43]:
```
1 print('Number of Rural Students with commute time > 1 hour:', len(studen
2 print('Number of Urban Students with commute time > 1 hour:',len(student
```

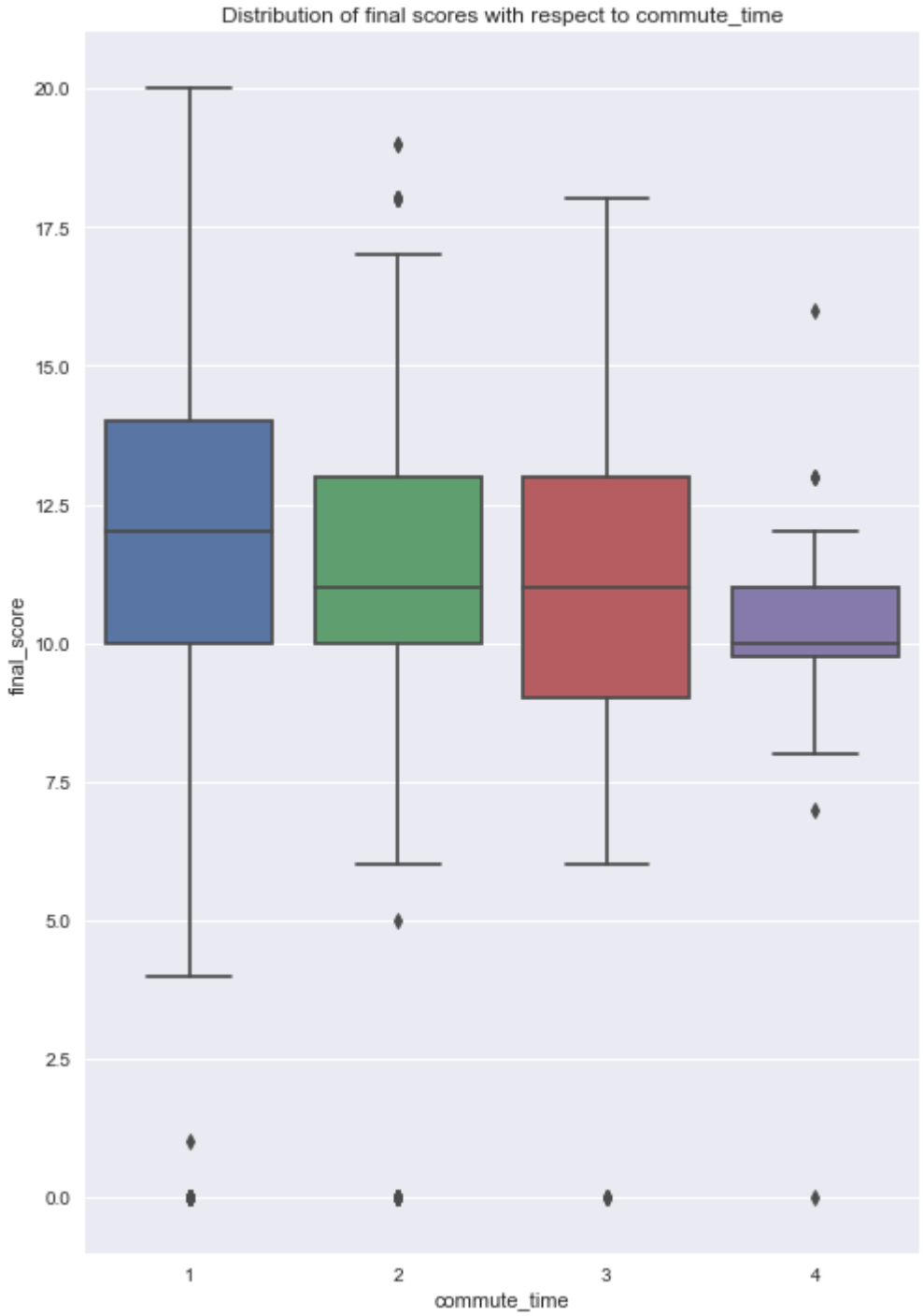
```
Number of Rural Students with commute time > 1 hour: 15
Number of Urban Students with commute time > 1 hour: 9
```

It is well known that students with higher commute thime usually get less time to study and thus there studies suffer.(Source: Kobus et al.,2018 (https://www.sciencedirect.com/science/article/abs/pii/S0166046215000216))

Let's see if that happens in this dataset as well:

In [44]:

```
plt.figure(figsize=(8,12))
plt.title("Distribution of final scores with respect to commute_time")
sns.boxplot(y="final_score", x="commute_time", data = student_df);
#plt.savefig('images/dist_final_commute_time')
```



The graph clearly depicts a downward trend with respect to travel time.

Schools in many countries consider commute time a criteria while admitting students.There are multiple studies that indicate higher commute times have adverse affect on student health as well(Source:Pradhan et al.,2017 (https://www.researchgate.net/publication/313851579_Impact_of_commuting_distance_and_school_

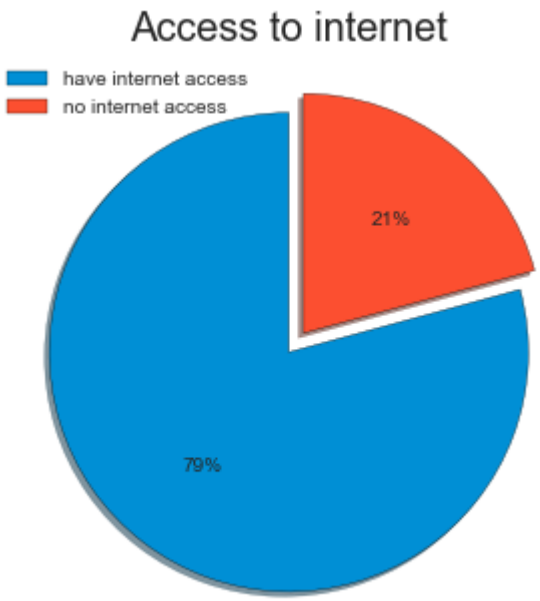
So, this graph tends as a proof that parents should prefer schools with lower commute times.

Another factor that might lead to the poor performance of ruaral students is access to `internet`.

`internet` column encodes whether a student has access to internet or not at home.Let's look at the frequency distribution for internet access.

In [45]:

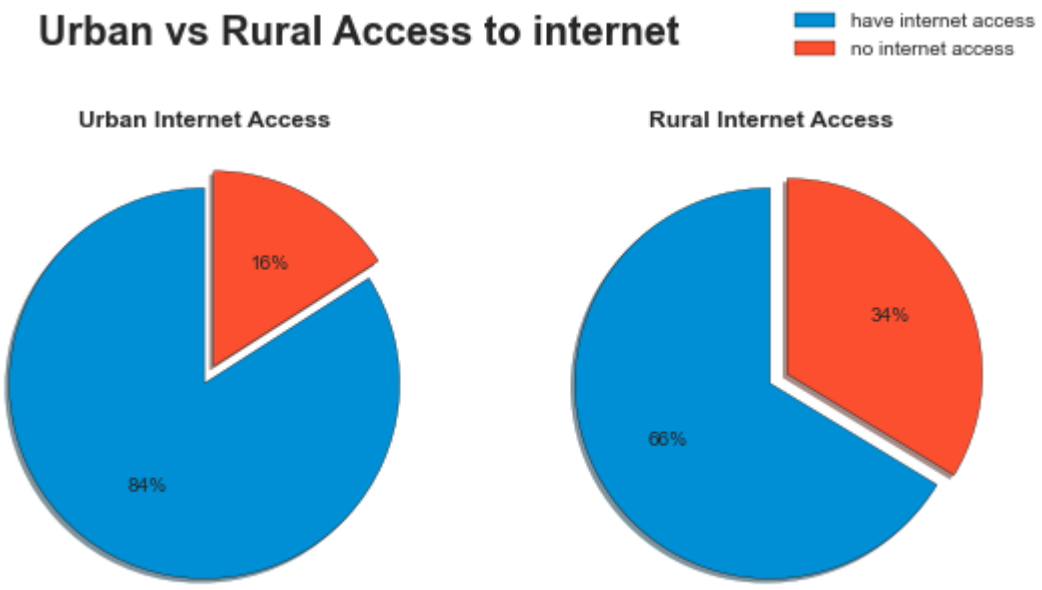
```
# Draw Pie-Chart of frequency distribution for internet access
plt.style.use('seaborn')
total_internet = student_df['internet'].value_counts().to_frame().T
labels = 'have internet access', 'no internet access'
colors = ['#008fd5', '#fc4f30']
explode = (0, 0.1)
explode2 = (0.2, 0)
plt.tight_layout()
plt.pie(total_internet.iloc[0], startangle=90, colors=colors, wedgeprops
plt.legend(loc='best', labels=labels, fontsize='medium')
plt.title('Access to internet', fontsize = 20)
#plt.savefig('images/access_to_internet')
plt.show()
```



Roughly 80% of all students in the dataset have access to internet.Let's see if this distribution is different for rural and urban areas.

In [46]:

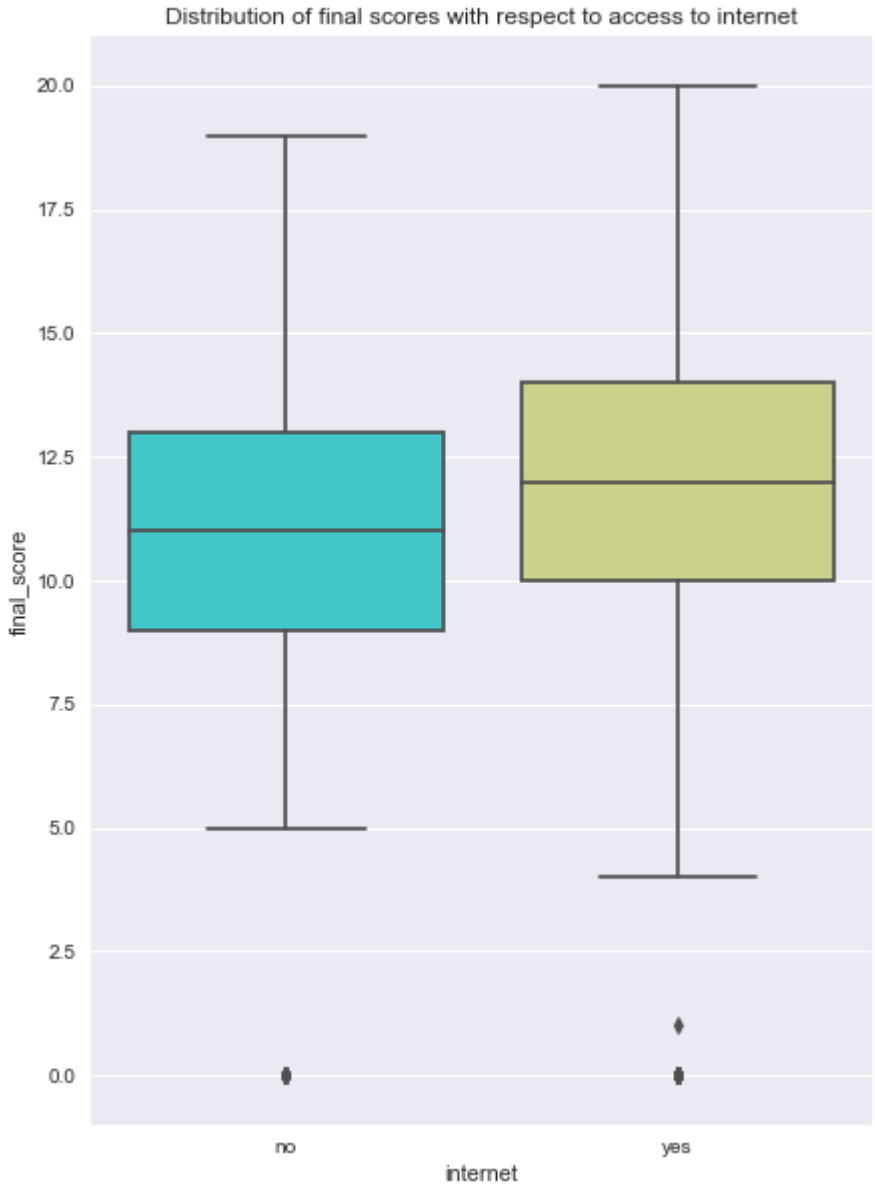
```
# Draw Pie-Chart of frequency distribution for internet access for urban
plt.style.use('seaborn')
urban_internet = student_df[student_df['address']=='U']['internet'].valu
rural_internet = student_df[student_df['address']=='R']['internet'].valu

labels = 'have internet access', 'no internet access'
colors = ['#008fd5', '#fc4f30']

fig, ax = plt.subplots(nrows=1, ncols=2)

explode = (0, 0.1)

plt.tight_layout()
ax[0].pie(urban_internet.iloc[0], startangle=90, colors=colors, wedgepro
ax[0].set_title('Urban Internet Access', fontweight='bold')

ax[1].pie(rural_internet.iloc[0], startangle=90, colors=colors, wedgepro
ax[1].set_title('Rural Internet Access', fontweight='bold')



fig.legend(labels=labels, fontsize='medium')
plt.suptitle('Urban vs Rural Access to internet', fontsize = 20, fontwei
#plt.savefig('images/U_R_Access_to_internet')
plt.show()
```



As expected, ahigher proportion of urban students have access to internet. 84% of urban students have access to internet vs 66% students with internet access for rural areas.

Studies have shown that having access to internet improves academic performance(Source : NCES (https://nces.ed.gov/pubs2017/2017098/ind_15.asp)).Let's see if that's the case in this dataset as well.

In [47]:
```
plt.figure(figsize=(7,10))
plt.title("Distribution of final scores with respect to access to interne
sns.boxplot(y="final_score", x="internet", data = student_df, palette ="
#plt.savefig('images/dist_final_access_to_internet')
```



Box plot indicates a higher score for students with internet access.

In [48]:
```
print('Mean final score for students with internet access:', student_df[
print('Mean final score for students without internet access:', student_
```

```
Mean final score for students with internet access: 12.0
Mean final score for students without internet access: 11.0
```

Thus, if students get internet access at home, they tend to perform better.It is important to monitor student internet usage though, as it might lead to an endless cycle of video games or youtube videos.

School districs should also work with students who don't have access to internet and see if some arrangements can be made to help such students in this day and age of online learning.Some tips include:

1. Encouraging students to use community resources like public libraries.
2. Make remote learning accessible through offline features.
3. Provid mobile hotspots to students from disadvantaged backgrounds.

Wish to go for Higher Education

`desire_higher_edu` column indicates whether student wants to take higher education or not.Let's look at its frequency distribution:

In [49]:
```
student_df['desire_higher_edu'].value_counts()
```

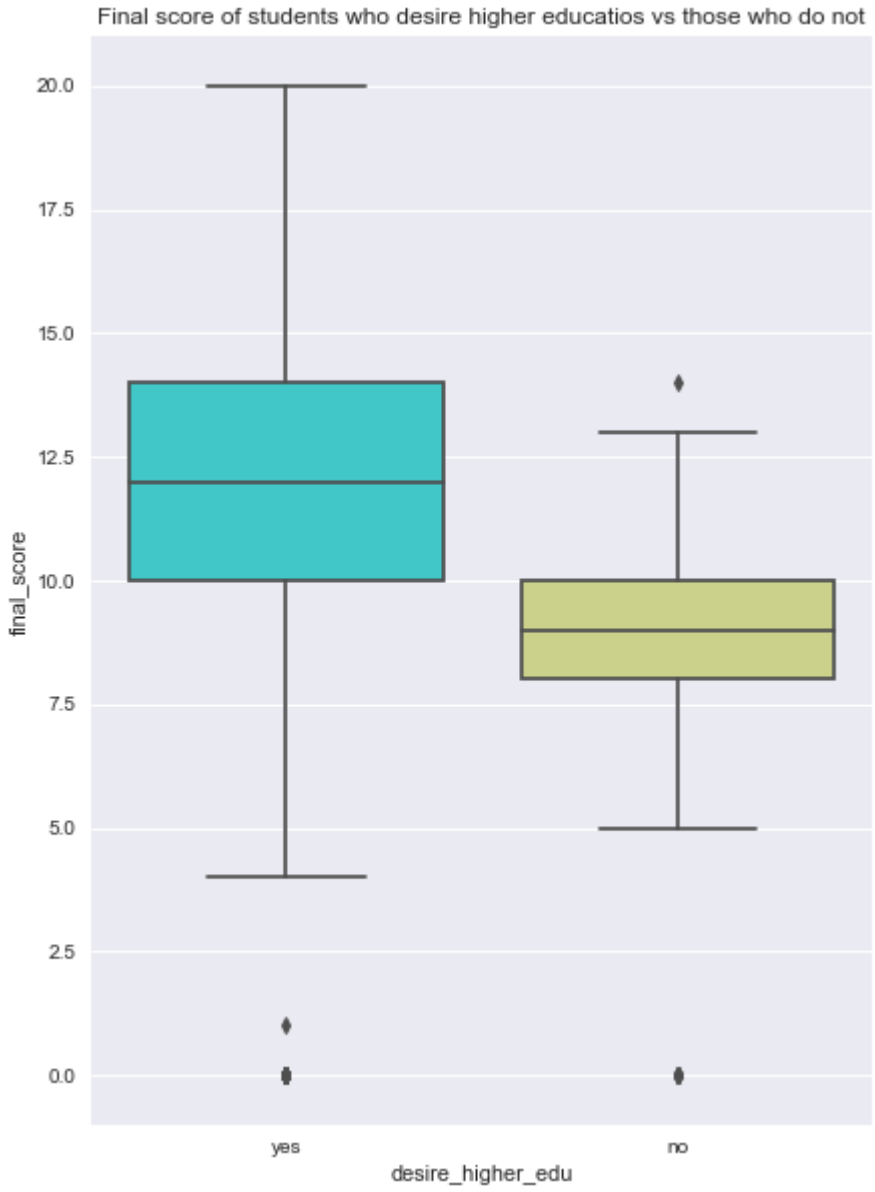
Out[49]:
```
yes    955
no      89
Name: desire_higher_edu, dtype: int64
```

It's nice to see that majority of the students wish yo opt for higher (college) education. This data is slightly old and is for portugese students.There has been decline in the number of students wanting to opt for university.

In US, about 75% high school students plan on going for higher education(Source:EAB (https://eab.com/insights/daily-briefing/enrollment/75-of-teens-plan-to-attend-higher-ed-after-high-school/)).

As a lot of higher education programs consider academic performance in the addmission criteria, it is expected that students who desire to go that path will focus more on earning a better grade.Let's see if that's the case in this dataset.

In [50]:
```
plt.figure(figsize=(7,10))
plt.title("Final score of students who desire higher educatios vs those w
sns.boxplot(y="final_score", x="desire_higher_edu", data = student_df, pa
#plt.savefig('images/final_who_do_dont')
```



There is very significant trend here. This seems to be very important feature in predicting academic success.

Students who desire to go for higher education perform better in high-school finals.

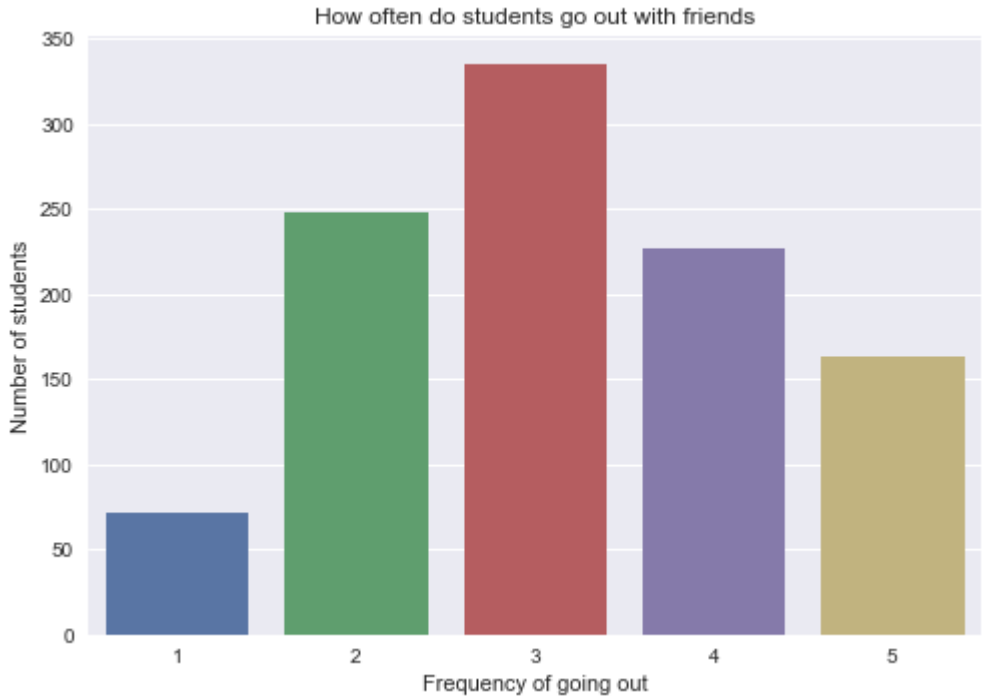
Going out with Friends

Socializing is a very important part of high school life.There are many benifits of having healthy friendly relationship that go well beyond student life.It helps overcome social anxiety, improve communication skills, and improve mental and social wellbeing.

As my target is academic performance, I will analyse the impact of going out with friends on `final_score` .

`go_out` column in the dataset indicates the frequency of going out with friends.It ranges from 1 - very low to 5 - very high.

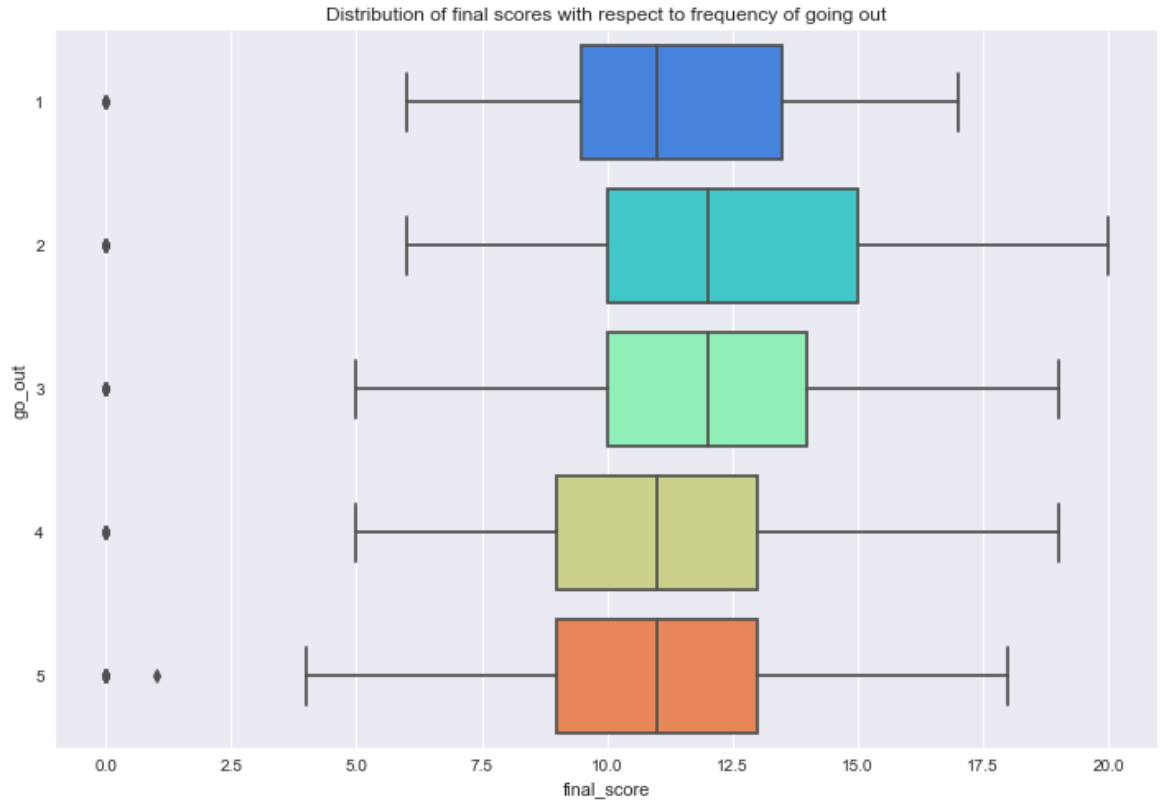
In [51]:
```
#frequency distribution of going out
tmp_plt = sns.countplot(x = 'go_out', data = student_df)
tmp_plt.axes.set_title('How often do students go out with friends')
tmp_plt.set_xlabel('Frequency of going out')
tmp_plt.set_ylabel('Number of students')
#plt.savefig('images/often_go_out')
plt.show()
```



I see that most students lie in the middle of this distribution which means that most students go out in moderation.Let's see the effect of going out on `final_score` :

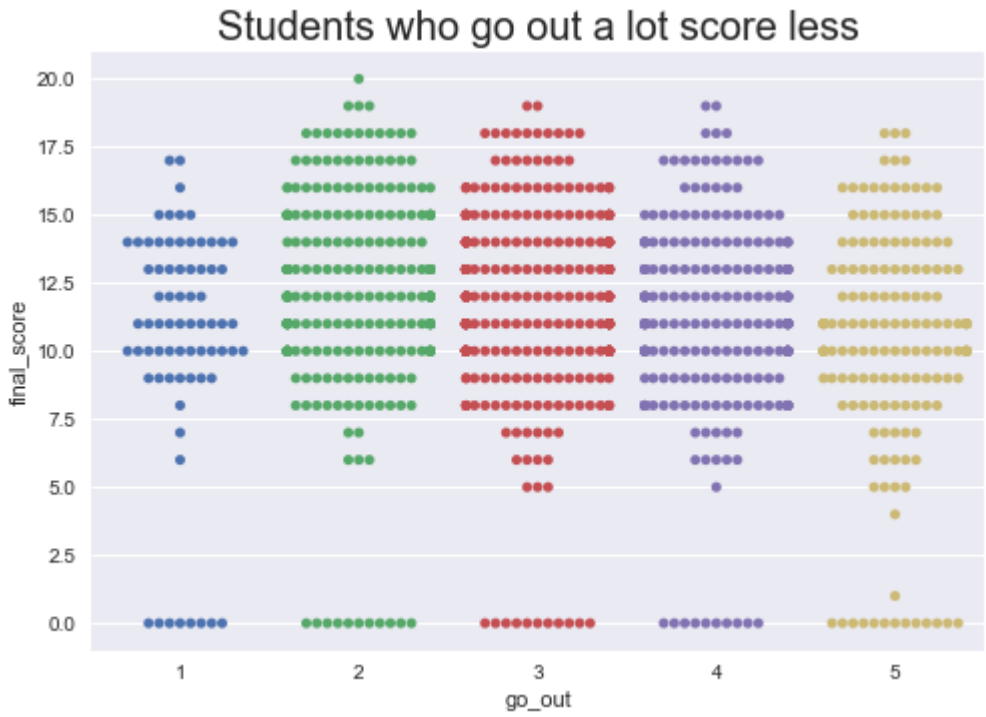
In [52]:

```
plt.figure(figsize=(12,8))
plt.title("Distribution of final scores with respect to frequency of goi
sns.boxplot(y="go_out", x="final_score", orient='h', data = student_df,
#plt.savefig('images/dist_final_frequency_go_out')
plt.show()
```



In [53]:

```
sns.swarmplot(x='go_out', y='final_score', data = student_df)
plt.title('Students who go out a lot score less', fontsize=20)
#plt.savefig('images/student_go_out_score_less')
plt.show()
```



There seems to be a slightly downward trend on the graphs especially for students who go out very frequently.

Students who tend to go out moderately seem to perform the best.There are many studies which indicate that prosocial behaviour is a critical component of academic success(Source:DeVeries et al.,2018 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5994475/)).

Absentism

I think it's fair to expect that students who are more absent in school generally perform poorly in

exams.This is why many schools have strict attendance requirements.

Let's study the impact of absentism on `final_score` in our dataset.

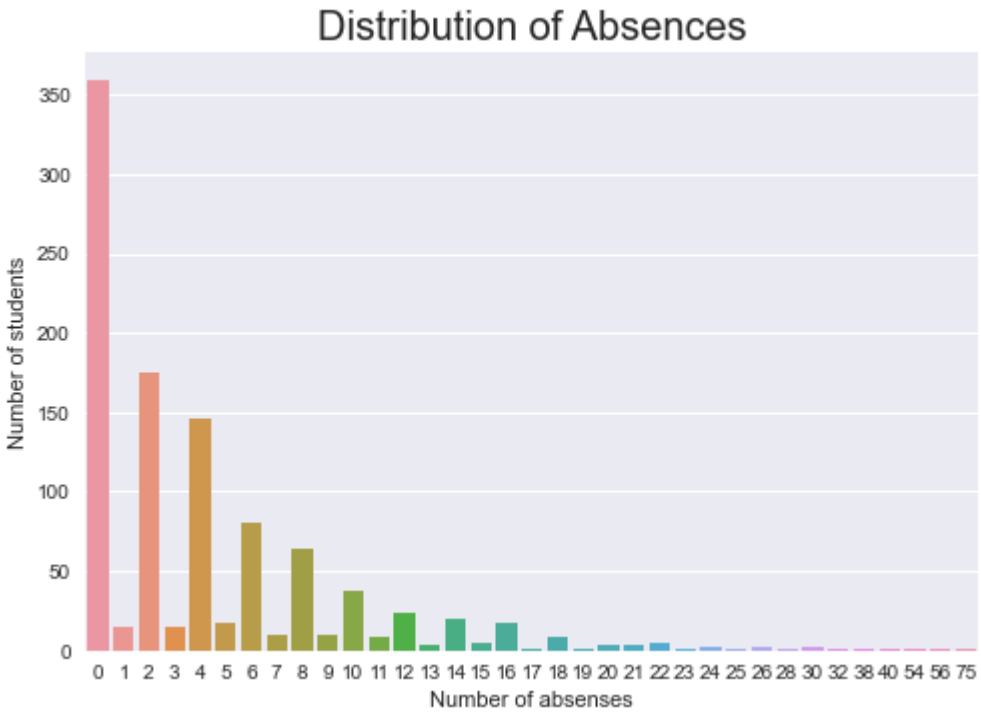
`absences` is a numarical column that indicates number of school absences.

In [54]:
```
1 student_df['absences'].describe()
```

Out[54]:
```
count    1044.000000
mean        4.434866
std         6.210017
min         0.000000
25%         0.000000
50%         2.000000
75%         6.000000
max        75.000000
Name: absences, dtype: float64
```

I see that absences are between the range of 0 and 75.Let's look at it's distribution.

In [55]:
```
1 # absence distribution
2 tmp_plt = sns.countplot(x = 'absences', data = student_df)
3 tmp_plt.axes.set_title('Distribution of Absences', fontsize=20)
4 tmp_plt.set_xlabel('Number of absenses')
5 tmp_plt.set_ylabel('Number of students')
6 #plt.savefig('images/dist_absences')
7 plt.show()
```

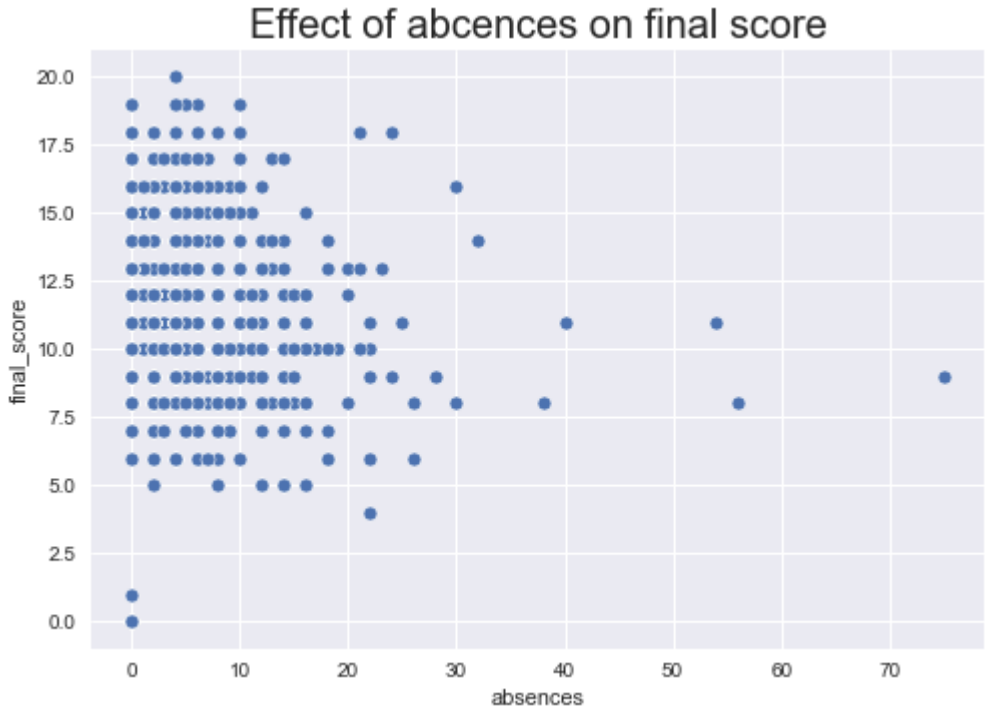


The graph shows that most of the students are very regular to the classes and there are only a few

students that are very irregular to school.Let's see the impact of number of absences to final score of the student.

In [56]:
```
print('Correlation Coefficient of absences with final_score', student_df
sns.scatterplot(x='absences', y='final_score', data = student_df)
plt.title('Effect of abcences on final score', fontsize=20)
#plt.savefig('images/effect_of_abcenses_final')
plt.show()
```

```
Correlation Coefficient of absences with final_score -0.04567057698837366
```



There seems to be a negligible negative correlation between the two columns.

The lack of a clear tend may be due to the fact that I don't have a lot of students that are irregular in school.But for now, I don't see a clear relationship between attendance regularity and final grade.

Romantic Relationship

This is very interesting sunject to study:'Effect of Romantic relationship on academic success of high-school students'.

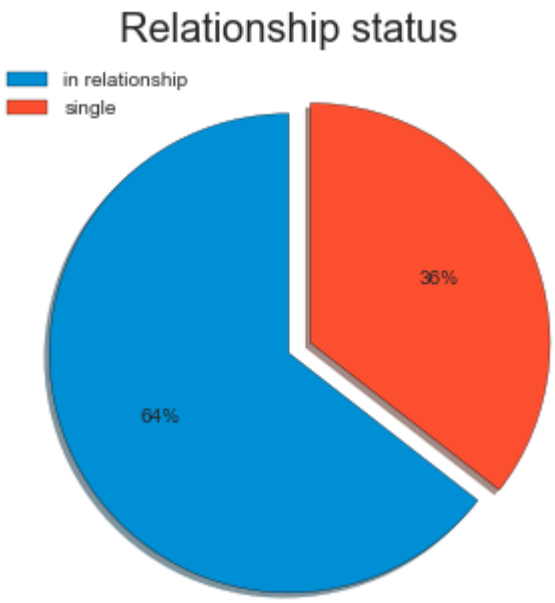
High school relationship have a tendency to be volatile and can end up causing alot of stress for a teenager as they are not fully mature to deal with emotional aspects of a relationship.

At the same time, it is seen that being in healthy relationships helps with mood regulation and overall mental illness.

`romantic` column in this dataset indicates whether a student is in relationship(`yes`) or not (`no`).

Let's start by looking at the frequency distribution:

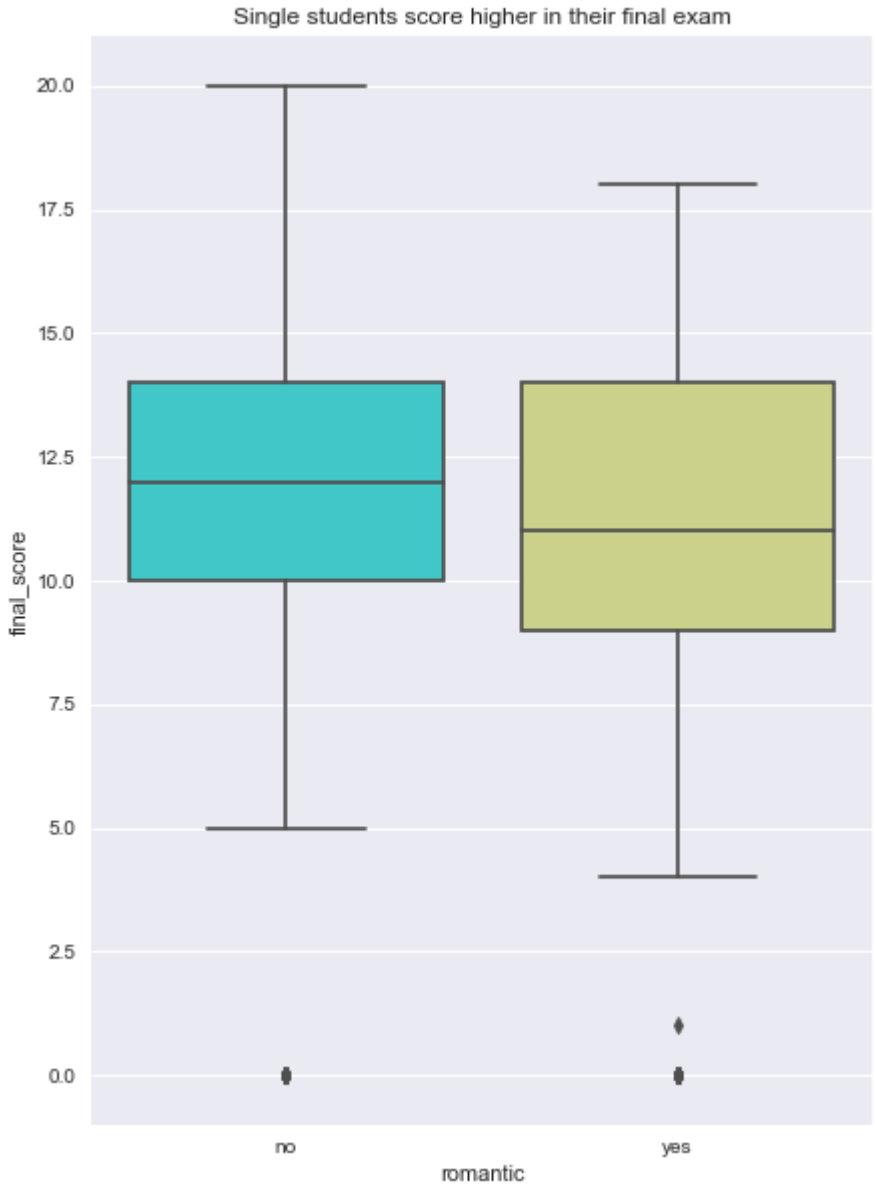
In [57]:
```
# Draw Pie-Chart of frequency distribution for internet access
plt.style.use('seaborn')
total_relationship = student_df['romantic'].value_counts().to_frame().T
labels = 'in relationship', 'single'
colors = ['#008fd5', '#fc4f30']
explode = (0, 0.1)
explode2 = (0.2, 0)
plt.tight_layout()
plt.pie(total_relationship.iloc[0], startangle=90, colors=colors, wedgep
plt.legend(loc='best', labels=labels, fontsize='medium')
plt.title('Relationship status', fontsize = 20)
#plt.savefig('images/relationship_status')
plt.show()
```



Approximately 64% of the students in this dataset are in a relationship.Let's now look at its impact on their final score:

In [58]:

```
plt.figure(figsize=(7,10))
plt.title("Single students score higher in their final exam")
sns.boxplot(y="final_score", x='romantic', data = student_df, palette =
#plt.savefig('images/single_student_higher_final')
plt.show()
```



The boxplot indicates that single students perform better in their exams. Let's check this hypothesis using two-tailed t-test.

In [59]:
```
single_student_scores = student_df[student_df['romantic'] == 'no']['fina
in_rel_student_scores = student_df[student_df['romantic'] == 'yes']['fin
```

In [60]:
```
# Running a two-tailded t-test https://docs.scipy.org/doc/scipy/referenc
t_value, p_value = scipy.stats.ttest_ind(single_student_scores, in_rel_s

print('Test statistic is %f'%float("{:.6f}".format(t_value)))
print('p-value for two tailed test id %f'%p_value)
```

```
Test statistic is 3.190633
p-value for two tailed test id 0.001462
```

In [61]:
```
alpha = 0.05
#testing for significance
if p_value<=alpha:
    print('Difference between single and in-relationship final scores is
else:
    print('Difference between single and in-relationship final scores is
```

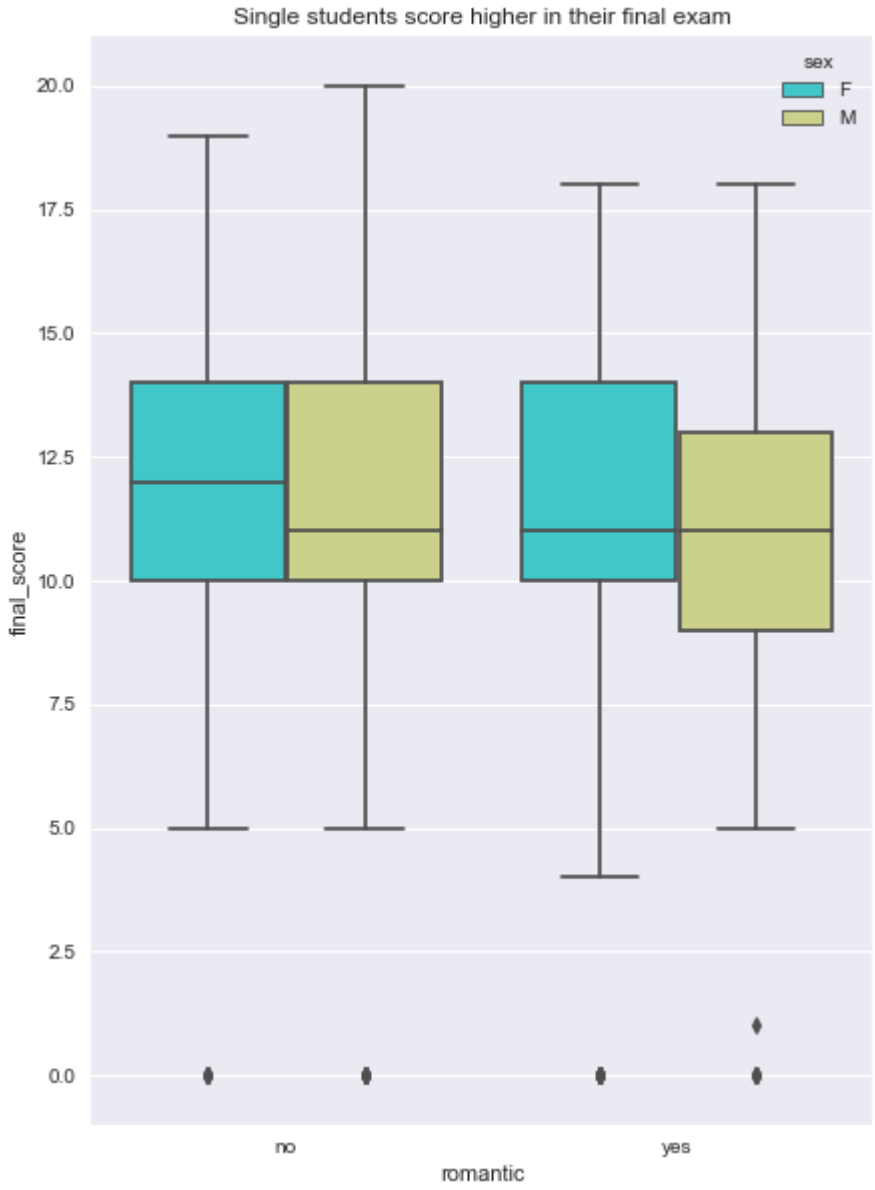
```
Difference between single and in-relationship final scores is statistacally
significant
```

I have established that romantic relationsship tend to negatively affect student performance.This could be due to the fact that tenagers in love would prefer to spend time each other over studying.

Now, let's try to analyze this trend further.Let's see if this effect is more profound for one gerder over other:

In [62]:

```
plt.figure(figsize=(7,10))
plt.title("Single students score higher in their final exam")
sns.boxplot(y="final_score", x="romantic", hue="sex", data = student_df,
#plt.savefig('images/single_f_m_higher_final')
plt.show()
```



In [63]:
```
1 student_df.groupby(["sex", "romantic"])['final_score'].mean()
```

Out[63]:
```
sex  romantic
F    no          11.836158
     yes         10.869198
M    no          11.388715
     yes         10.761194
Name: final_score, dtype: float64
```

Both the genders seem to be effected by relationships.The effect seems to be higher on females.This is also found in literature as well(Zayed,2016 (https://www.researchgate.net/publication/353193236_Romantic_relationships_effects_academically

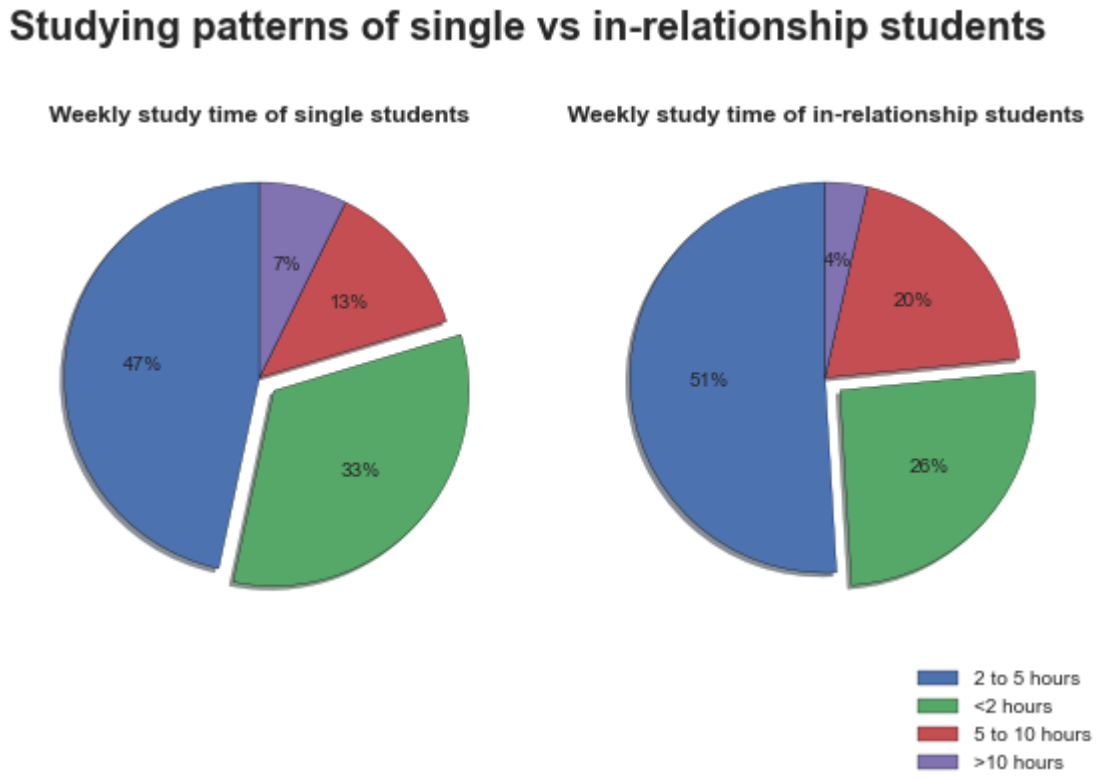
Let's see if relationship status impact studying time:

In [64]:
```
1 student_df[student_df['romantic']=='no']['new_study_time'].value_counts(
```

Out[64]:
```
2 to 5 hours     314
<2 hours         222
5 to 10 hours     88
>10 hours         49
Name: new_study_time, dtype: int64
```

In [65]:

```
# Draw pie charts to see frequency distribution of internet access for r

plt.style.use('seaborn')

single_study_time = student_df[student_df['romantic']=='no']['new_study_
in_relationship_study_time = student_df[student_df['romantic']=='yes']['

labels = ['2 to 5 hours', '<2 hours', '5 to 10 hours', '>10 hours']
#colors = ['#008fd5', '#fc4f30']


fig, ax = plt.subplots(nrows=1, ncols=2)

explode = (0, 0.1,0,0)

plt.tight_layout()
ax[0].pie(single_study_time.iloc[0], startangle=90, wedgeprops={'edgecol
ax[0].set_title('Weekly study time of single students', fontweight='bold

ax[1].pie(in_relationship_study_time.iloc[0], startangle=90, wedgeprops=
ax[1].set_title('Weekly study time of in-relationship students', fontwei

fig.legend(labels=labels, loc='lower right',fontsize='medium')
plt.suptitle('Studying patterns of single vs in-relationship students',
#plt.savefig('images/weekly_study_single_inrel')
plt.show()
```



This seems to be quite mixed.While the students who study the most are single, the students who seem to study the least are also single.

Alcohol Consumption

Now let's look at teen alcohol consumption.

Considering that it is illegal in US to drink below 21 years of age, it is surpising to know that nearly 50% of all high school students drink alcohol.Excesive consumption of alcohol in people below 21 years of age is considered to be responsible for approximately 4,300 deaths every year from 2006 - 2010(Esser et al.,2016 (https://www.cdc.gov/mmwr/volumes/66/wr/mm6618a4.htm)).

Studies also indicate there has been a rise in binge drinking (https://www.cdc.gov/alcohol/fact-sheets/binge-drinking.htm) among high-schoolers, which means that when students drink they end up drinking a lot of alcohol.This is even more dangerous than regular alcohol usage.

it is clear that teenage drinking is a major health concern in US and while we have seen some improvements over the years(Esser et al.,2016 (https://www.cdc.gov/mmwr/volumes/66/wr/mm6618a4.htm)), more needs to be done to mitigate this situation.

In this analysis, I will focus on the impact of alcohol consumption on academic performance.

Studies indicate that increase in alchohol consumption leads to poorer academic performance(Balsa et al.,2012 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3026599/),Ansari et al.,2013 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3843305/)).

Let's see if this dataset shows similar trend. Two columns in dataset will be our prime focus:

- `weekday_alcohol_usage` : workday alcohol consumption (from 1 -very low to 5 -very high)
- `weekend_alcohol_usage` : workend alcohol consumption (from 1 -very low to 5 -very high)

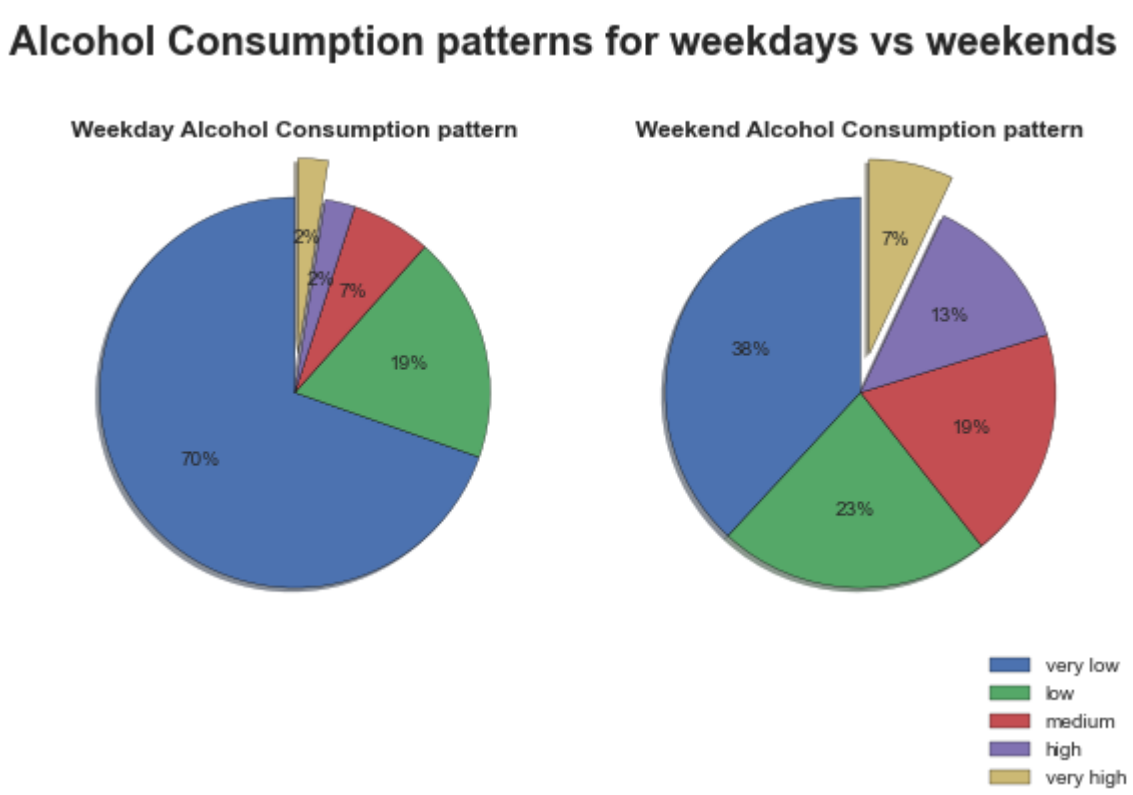
Now let's see at the frequency distribution of alcohol consumption between weekdays and weekends.

In [66]:
```
1 student_df['weekend_alcohol_usage'].value_counts()
```

Out[66]:
```
1    398
2    235
3    200
4    138
5     73
Name: weekend_alcohol_usage, dtype: int64
```

In [67]:

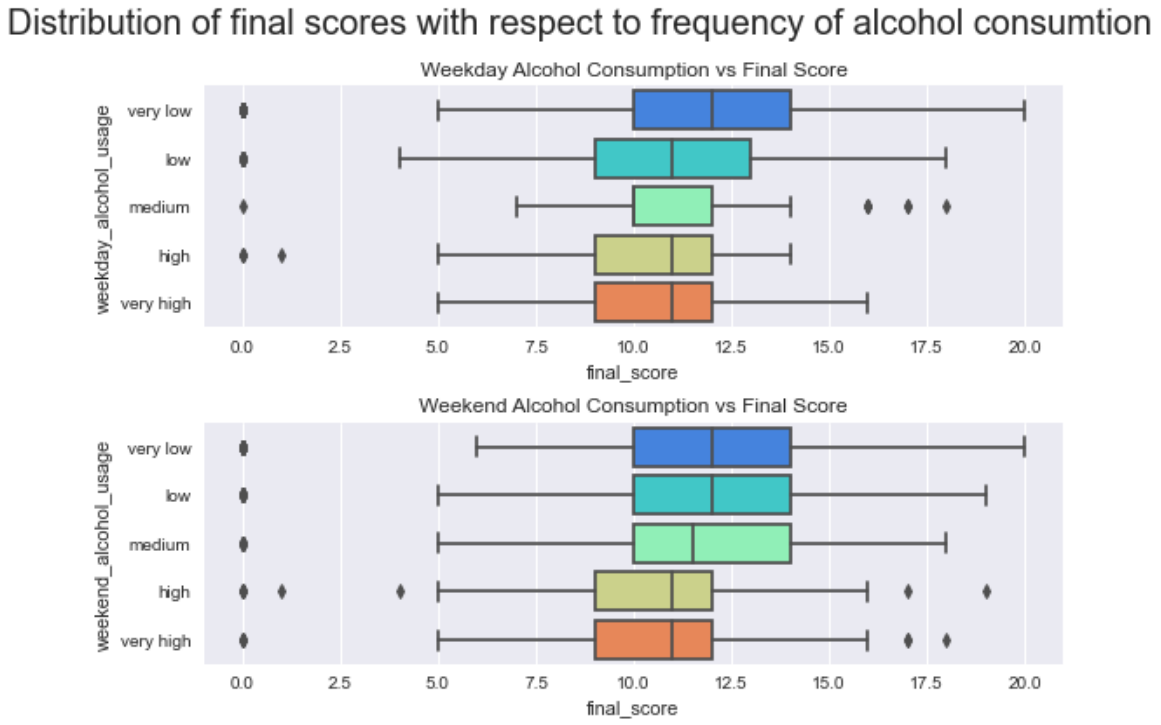
```
# Draw pie charts to see frequency distribution of alcohol consumption f

plt.style.use('seaborn')

weekday_alcohol_consumption = student_df['weekday_alcohol_usage'].value_
weekend_alcohol_consumption = student_df['weekend_alcohol_usage'].value_

labels = ['very low', 'low', 'medium', 'high', 'very high']

fig, ax = plt.subplots(nrows=1, ncols=2)

explode = (0, 0,0,0,0.2)

plt.tight_layout()
ax[0].pie(weekday_alcohol_consumption.iloc[0], startangle=90, wedgeprops
ax[0].set_title('Weekday Alcohol Consumption pattern', fontweight='bold'

ax[1].pie(weekend_alcohol_consumption.iloc[0], startangle=90, wedgeprops
ax[1].set_title('Weekend Alcohol Consumption pattern', fontweight='bold'

fig.legend(labels=labels, loc='lower right',fontsize='medium')
plt.suptitle('Alcohol Consumption patterns for weekdays vs weekends', fo
#plt.savefig('images/alcohol_consumption_weekday_weekend')
plt.show()
```



It's good to see that 70% of the student drink very low alcohol on weekdays.As expected alcohol usage increases over the weekend.Let's now look at the impact of alcohol consumption on academic performance.

In [68]:

```
fig, ax = plt.subplots(nrows=2, ncols=1)
b1 = sns.boxplot(y="weekday_alcohol_usage", x="final_score", orient='h',
b1.set_title("Weekday Alcohol Consumption vs Final Score")
b1.set_yticklabels(['very low', 'low', 'medium', 'high', 'very high'])

b2 = sns.boxplot(y="weekend_alcohol_usage", x="final_score", orient='h',
b2.set_title("Weekend Alcohol Consumption vs Final Score")
b2.set_yticklabels(['very low', 'low', 'medium', 'high', 'very high'])

plt.tight_layout()


plt.suptitle("Distribution of final scores with respect to frequency of
#plt.savefig('images/dist_final_frequency_consumption')
plt.show()
```



In both the cases, I see that alcohol consumption leads to decline in `final_score` .Let's run a hypothesis test using ANOVA to check for significance of this effect.I will first look at weekend drinking.

In [69]:

```
very_low_weekend_drinking = student_df[student_df['weekend_alcohol_usage
low_weekend_drinking = student_df[student_df['weekend_alcohol_usage']==2
medium_weekend_drinking = student_df[student_df['weekend_alcohol_usage']
high_weekend_drinking = student_df[student_df['weekend_alcohol_usage']==
very_high_weekend_drinking = student_df[student_df['weekend_alcohol_usag
```

I will run a [One-Way ANOVA test (https://www.simplypsychology.org/anova.html)](https://www.simplypsychology.org/anova.html) to check if there

exists a statistically signisignificant difference between the mean score of different alcohol consumption pattern.

In [70]:
```
# Running a one-way ANOVA test https://docs.scipy.org/doc/scipy/referenc
t_value, p_value = scipy.stats.f_oneway(very_low_weekend_drinking, low_w

print('Test statistic is %f'%float("{:.6f}".format(t_value)))
print('p-value for ANOVA test is %f'%p_value)
```

```
Test statistic is 3.780943
p-value for ANOVA test is 0.004641
```

In [71]:
```
alpha = 0.05
# testing for significance
if p_value<=alpha:
    print('Difference between final scores for different alcohol consump
else:
    print('Difference between final scores for different alcohol consump
```

```
Difference between final scores for different alcohol consumption patterns
over the weekend is statistically significant
```

In [72]:
```
pd.DataFrame({'Weekend Alcohol Usage':['very low', 'low', 'medium', 'hig
    'Avarage Final Score':student_df.groupby('weekend_alcohol_usage').me
              })
```

Out[72]:

	Weekend Alcohol Usage	Avarage Final Score
0	very low	11.743719
1	low	11.472340
2	medium	11.290000
3	high	10.536232
4	very high	10.397260

So, increase in weekend alcohol consumption has a significant nagetive impact on final score.Let's test the same for weekday alcohol consumption as well.

In [73]:
```
very_low_weekday_drinking = student_df[student_df['weekday_alcohol_usage
low_weekday_drinking = student_df[student_df['weekday_alcohol_usage']==2
medium_weekday_drinking = student_df[student_df['weekday_alcohol_usage']
high_weekday_drinking = student_df[student_df['weekday_alcohol_usage']==
very_high_weekday_drinking = student_df[student_df['weekday_alcohol_usag
```

In [74]:
```
# Running a one-way ANOVA test https://docs.scipy.org/doc/scipy/referenc
t_value, p_value = scipy.stats.f_oneway(very_low_weekday_drinking, low_w

print('Test statistic is %f'%float("{:.6f}".format(t_value)))
print('p-value for ANOVA test is %f'%p_value)
```

```
Test statistic is 6.241569
p-value for ANOVA test is 0.000058
```

In [75]:
```
alpha = 0.05
# testing for significance
if p_value<=alpha:
    print('Difference between final scores for different alcohol consump
else:
    print('Difference between final scores for different alcohol consump
```

```
Difference between final scores for different alcohol consumption patterns
over the weekday is statistically significant
```

In [76]:
```
pd.DataFrame({'Weekday Alcohol Usage':['very low', 'low', 'medium', 'hig
    'Avarage Final Score':student_df.groupby('weekday_alcohol_usage').me
             })
```

Out[76]:

	Weekday Alcohol Usage	Avarage Final Score
0	very low	11.704264
1	low	10.556122
2	medium	10.898551
3	high	9.269231
4	very high	10.384615

I see an even more significant negative impact of increase in alchocol consumption on final score when done during workdays.

There is also an inconsistency here that the mean final score is higher for `very high` alcohol consumption compared to `high consumption` on weekdays. This could be due to the very low number of students who consume `very high` alcohol during weekdays.

Nevertheless, the trend is very clear.Increasing alcohol consumption among high school students will lead to decreace in academic performance.

Parent's Education

Many studies indicate that parent;s education levels have critical impact on a child's academic and long-term career success.(Dubow et al.,2009 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2853053/))

Educated parents can often help students with their studies and also help instill studying discipline in their children.They are also better equipped to deal with school curricullum and help guide their children to their chosen careers(Source (https://degree.lamar.edu/articles/undergraduate/parents-education-level-and-childrens-success/)).

Many schools woldwide consider parent's education as an important criteria while admitting a student into their programs. I will now run an analysis on this dataset to see if this criteria holds any merit or is based on archaic idead of student success.

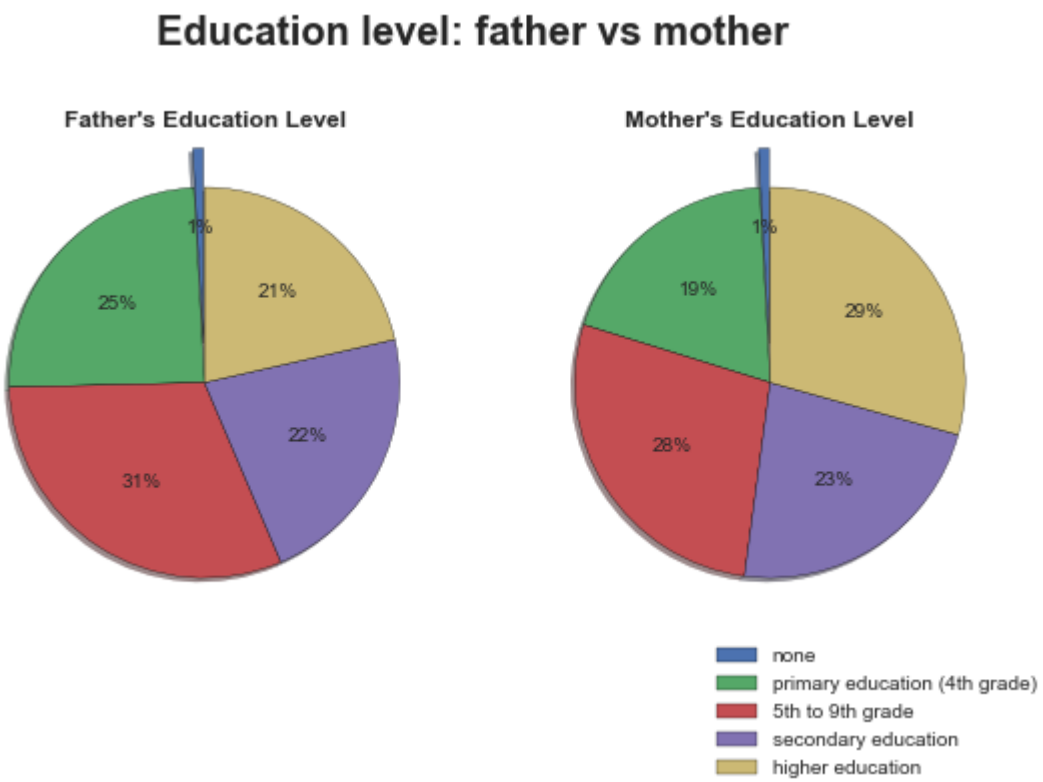
There are two columns related to parent's education in this dataset:

- `mother_education` : 0 - none, 1 - primary education(4th grade), 2 - 5th to 9th grade, 3 - secondary education, 4 - higher education
- `father_education` : 0 - none, 1 - primary education(4th grade), 2 - 5th to 9th grade, 3 - secondary education, 4 - higher education

  Let's first look at the distribution of education for both mothers and fathers.

In [77]:

```
# Draw pie charts to see frequency distribution of fathers and mothers

plt.style.use('seaborn')

father_edu = student_df['father_education'].value_counts().sort_index().
mother_edu = student_df['mother_education'].value_counts().sort_index().

labels = ['none','primary education (4th grade)','5th to 9th grade','sec

fig, ax = plt.subplots(nrows=1, ncols=2)

explode = (0.2, 0,0,0,0)

plt.tight_layout()
ax[0].pie(father_edu.iloc[0], startangle=90, wedgeprops={'edgecolor': 'b
ax[0].set_title("Father's Education Level", fontweight='bold')

ax[1].pie(mother_edu.iloc[0], startangle=90, wedgeprops={'edgecolor': 'b
ax[1].set_title("Mother's Education Level", fontweight='bold')

fig.legend(labels=labels, loc='lower right',fontsize='medium')
plt.suptitle('Education level: father vs mother', fontsize=20, fontweigh
#plt.savefig('images/edu_level_father_mother')
plt.show()
```

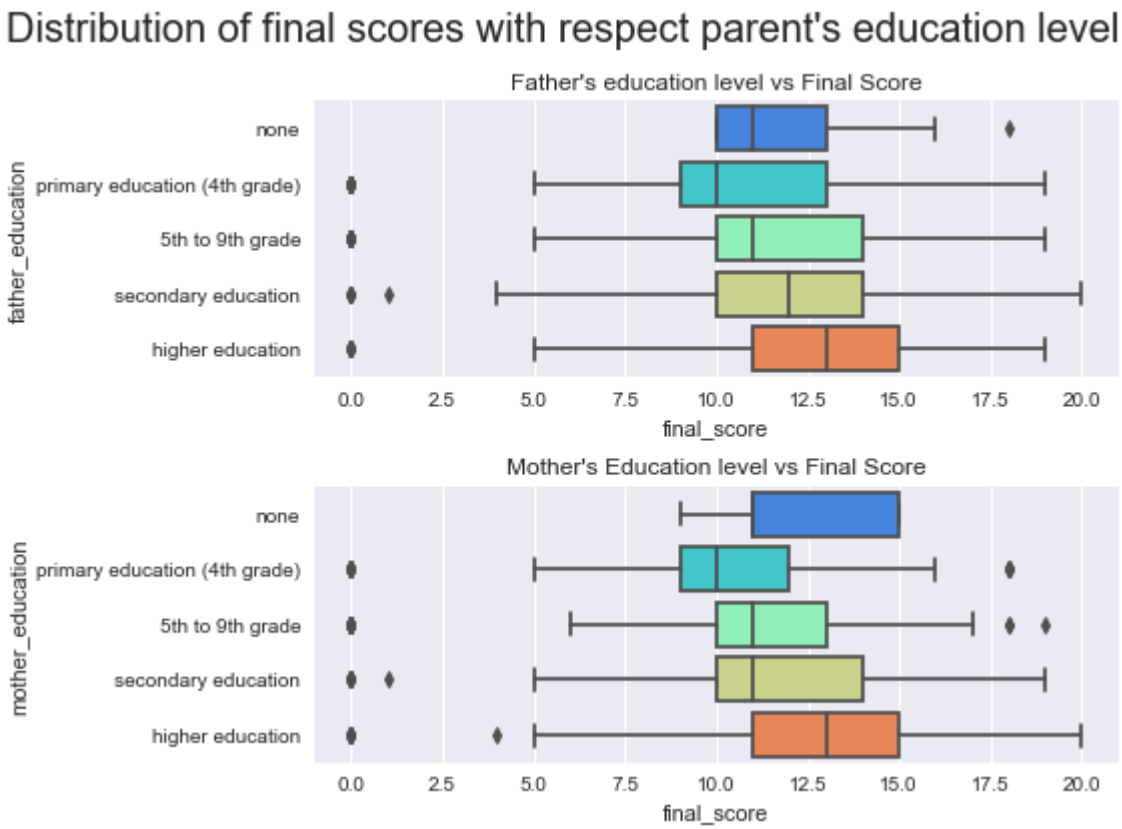


The none values indicate no education seem to be very low in number.These could be rows for which I have no info about parent's education or the students are orphans. I will ignore these values for our further analysis.

I see that overall mothers seem to have better education than fathers.Let's look at the impact of parent's education on final score:

In [78]:

```
fig, ax = plt.subplots(nrows=2, ncols=1)
b1 = sns.boxplot(y="father_education", x="final_score", orient='h',data =
b1.set_title("Father's education level vs Final Score")
b1.set_yticklabels(['none','primary education (4th grade)','5th to 9th g

b2 = sns.boxplot(y="mother_education", x="final_score", orient='h',data =
b2.set_title("Mother's Education level vs Final Score")
b2.set_yticklabels(['none','primary education (4th grade)','5th to 9th g

plt.tight_layout()


plt.suptitle("Distribution of final scores with respect parent's educati
#plt.savefig('images/dist_final_parent_edu_level')
plt.show()
```



It can be seen that for both parents, having better education will lead to better academic success for the student.

Interestingly, this effect seems to be more profound for fathers.This is consistent with literature that suggests that father's education level is very strong factor for child's success at school(Source (https://www.theguardian.com/society/2014/sep/23/fathers-education-child-success-school)).

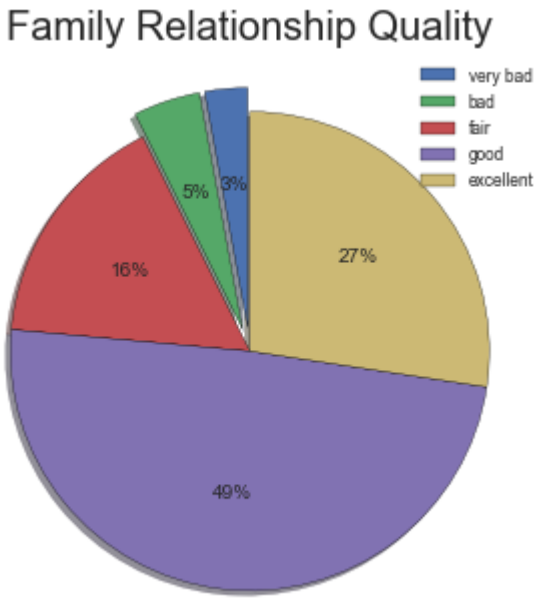
Family Quality

Anna J Egalite (https://www.educationnext.org/how-family-background-influences-student-

achievement/) covered the impact family background can have on a student's academic success.While factors are equally important, the family that you are born into, the socio-economic factors, the home environment and your parents support does impact your life very significantly.

In this dataset, I have column called `family_quality` which I will use to analze impact of family quality on a student's grade.It captures the quality of family relationships, going from 1(very bad) to 5(excellent).

In [79]:
```
# Draw Pie-Chart of frequency distribution for family relationship quali

plt.style.use('seaborn')

family_relationship = student_df['family_quality'].value_counts().sort_i
labels = 'very bad', 'bad', 'fair', 'good', 'excellent'
colors = ['#008fd5', '#fc4f30']
explode = (0.1, 0.1, 0, 0, 0)

plt.tight_layout()
plt.pie(family_relationship.iloc[0], startangle=90, wedgeprops={'edgecol
plt.legend(loc='best', labels=labels, fontsize='small')
plt.title('Family Relationship Quality', fontsize = 20)
#plt.savefig('images/family_rel_quality')
plt.show()
```

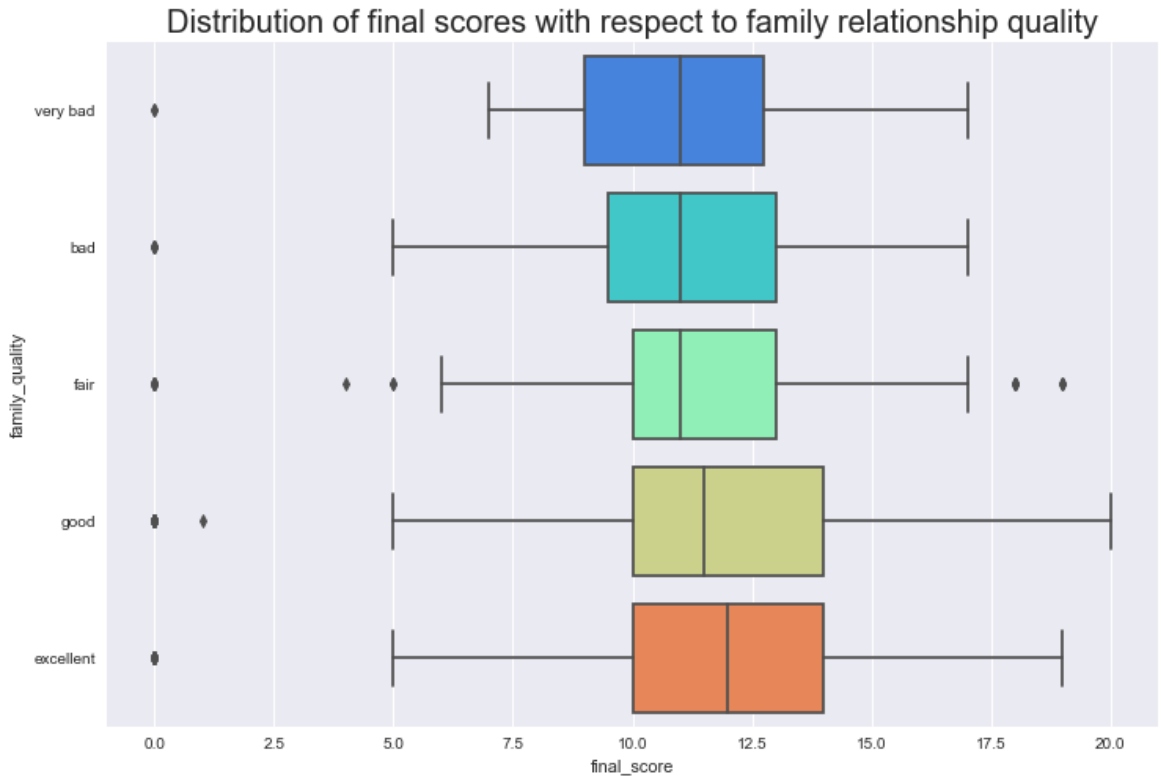


This variable was recorded through surveys filled by students themselves.It is refreshing to see that only 8% students feel that they have a bad/very bad family situation.More than 75% of the student rated their family relationship quality as good as or excellent.

Let's now see the impact of family quality on final score:

In [80]:

```
plt.figure(figsize=(12,8))
plt.title("Distribution of final scores with respect to family relations
plot = sns.boxplot(y="family_quality", x="final_score", orient='h', data
plot.set_yticklabels(labels)
#plt.savefig('images/dist_final_family_rel_quality')
plt.show()
```

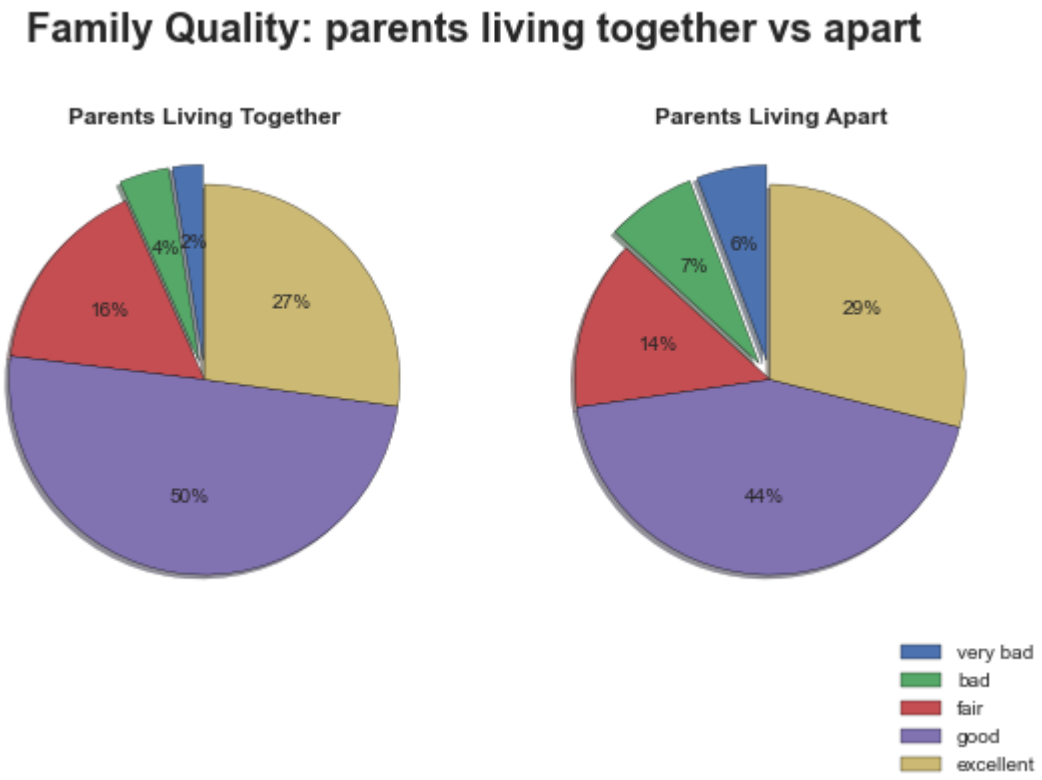


I see a consistent linear trend here.As quality of family relationship increase, a student's final score increases as well.

Let's look at a factor that might impact the quality of family relationship - `parent status` .This column contains information on parent's cohabitation status (**"T" -living together or "A" -apart**).

In [81]:

```
# Draw pie charts to see frequency distribution of parent status

plt.style.use('seaborn')

parents_living_together = student_df[student_df['parent_status']=='T']['
parents_living_apart = student_df[student_df['parent_status']=='A']['fam

labels = 'very bad', 'bad', 'fair', 'good', 'excellent'

explode = (0.1, 0.1, 0, 0, 0)

fig, ax = plt.subplots(nrows=1, ncols=2)

plt.tight_layout()
ax[0].pie(parents_living_together.iloc[0], startangle=90, wedgeprops={'e
ax[0].set_title("Parents Living Together", fontweight='bold')

ax[1].pie(parents_living_apart.iloc[0], startangle=90, wedgeprops={'edge
ax[1].set_title("Parents Living Apart", fontweight='bold')

fig.legend(labels=labels, loc='lower right',fontsize='medium')
plt.suptitle('Family Quality: parents living together vs apart', fontsiz
#plt.savefig('images/pie_parent_live_together_vs_apart')
plt.show()
```



I see that in this dataset, the quality of family relationship is reduced when parents live apart.

Important Insights from EDA

1. Student age or gender do not affect academic grade.
2. Increasing study time improves final results.
3. Increased alcohol consumption decreaces final results.
4. Relationships do impact students' academic score.
5. Family plays a major role in child's academic success whether it is parent's education or their relationship status.
6. Parents should look for a school closer to their home as increased commute time affects academic performance negatively.

Modeling

Preparing dataset for Machine Learning

Now, I will prepare our dataset for machine learning. I have already cleaned our dataset and removed filler values.

I now need to handle all the categorical columns and do some feature engineering.

As the target variable is continuous, this will be a regression task. I will evaluate our models using commonly used accuracy metrics like R-squared or coefficient of determination (https://www.investopedia.com/terms/r/r-squared.asp), RMSE (https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/) and MAE (https://en.wikipedia.org/wiki/Mean_absolute_error).

While discussing the target variable final_score during EDA, we discussed the case of many zero values. I showed that there could be multiple reasons for it and there is no clarification from the authors of this dataset on what exactly does the zero final score represent for each row.

This is why I chose to consider them as outliers and remove them from the dataset.

In [82]:
```
# removing all rows where final score is 0
student_df = student_df[student_df['final_score']!=0]
print(len(student_df))
```

```
991
```

So, the new dataset now has 991 rows and no students with zero score.

In [83]:
```
# splitting the data into features (X) and target variable(y)
X = student_df.copy()

#dropping both the irrelevant columns and the target variable
X = X.drop(['new_study_time','final_score'], axis=1)
y = student_df['final_score']
```

All ordinal variables in the dataset are already encoded. All nominal categorical variables in the data are of object type. I can directly call `pandas.get_dummmies` to convert them into one-hot encoded columns suitable for machine learning.

In [84]:
```
# setting drop_first = True to decrease dimensionality and remove one fi
X = pd.get_dummies(X, drop_first=True)
X
```

Out[84]:

	age	mother_education	father_education	commute_time	study_time	failures	family_qu
0	18	4	4	2	2	0	
1	17	1	1	1	2	0	
2	15	1	1	1	2	3	
3	15	4	2	1	3	0	
4	16	3	3	1	2	0	
...	...	...	...	...	...	...	
1039	19	2	3	1	3	1	
1040	18	3	1	1	2	0	
1041	18	1	1	2	2	0	
1042	17	3	1	2	1	0	
1043	18	3	2	3	1	0	

991 rows × 41 columns

I have a total of 41 columns and 991 data points in our X.

In [85]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X),columns = X.columns)
```

Now I will split the dataset into two parts: a training dataset and a testing dataset.

I chose a 80-20 split and I will use sklearn's `train_test_split` function to do so.

In [86]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_si
```

In [87]:
```
X_train.head()
```

Out[87]:

	age	mother_education	father_education	commute_time	study_time	failures	family
910	0.142857	0.25	0.50	0.000000	0.000000	0.0	
47	0.142857	1.00	0.75	0.000000	1.000000	0.0	
517	0.000000	0.25	0.50	0.000000	0.333333	0.0	
631	0.428571	0.50	0.25	0.333333	0.666667	0.0	
19	0.142857	1.00	0.75	0.000000	0.000000	0.0	

5 rows × 41 columns

In [88]:
```
y_train.head()
```

Out[88]:
```
951    11
47     20
555    11
670    11
19     10
Name: final_score, dtype: int64
```

In [89]:
```
print('Size of training data:',len(y_train))
print('Size of testing data:',len(y_test))
```

```
Size of training data: 792
Size of testing data: 199
```

Baseline Model

The first step would be to build a naive model to get a baseline level of performance.The target then would be to build a model that does better than at least this model.

I will use Sklearn's `DummyRegressor` here with the `mean` strategy. This would only output the mean of the training data as it's prediction.Let's evaluate this model.

In [90]:
```
# create a dummy regressor
dummy_reg = DummyRegressor(strategy='mean')

# fit it on the training set
dummy_reg.fit(X_train, y_train)

# make predictions on the test set
y_pred = dummy_reg.predict(X_test)

# calculate root mean squared error on the test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for Dummy Regressor:", rmse)
print("MAE for Dummy Regressor:", mae)
print("R-Squared for Dummy Regressor:", r2)
```

```
RMSE for Dummy Regressor: 2.859872873921179
MAE for Dummy Regressor: 2.365324095223593
R-Squared for Dummy Regressor: -0.032955765537687975
```

As expected, this is a very bad model. A negative R-Squared means that this model is predicting worse than the mean of the test data. This will serve as our naive baseline.

Let's now jump to our first model. I will start things with a simple linear regression.

Linear Regression

I will start with a linear regression model (https://www.ibm.com/in-en/topics/linear-regression). There are a lot of variables that have clear statistical relationship with the target variable so I am

expecting a good performance from this model.

In [91]:

```
# create a linear regression model
lin_reg = LinearRegression()
# fit on the training data
lin_reg.fit(X_train, y_train)
# make predictions on the test set
y_pred = lin_reg.predict(X_test)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for Linear Regressor:", rmse)
print("MAE for Linear Regressor:", mae)
print("R-Squared for Linear Regressor:", r2)
```

```
RMSE for Linear Regressor: 0.8598858292632797
MAE for Linear Regressor: 0.6733399854361957
R-Squared for Linear Regressor: 0.9066165636896389
```

I see a fairly good performance even without any optimization on the linear regression model. This tells me that we have some really good features.

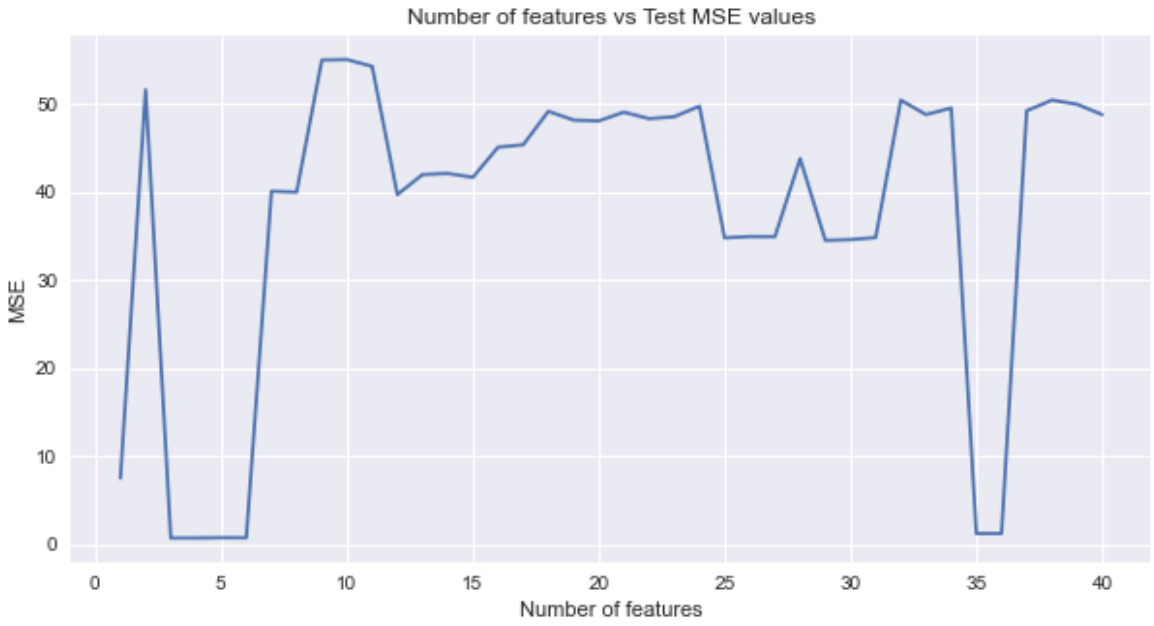
Now I will do some feature selection (https://en.wikipedia.org/wiki/Feature_selection). I will be using the chi-square test (https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/8-chi-squared-tests) to measure the dependence of the target variable on all the features.

Then I will pick the k-best features from the lot. I will tune the value of k by runninng a grid search on all plausible values for k to find the optimal k.

In [92]:

```
# to find optimal number of features to use in the model
# we are using chi-square test from sklearn https://scikit-learn.org/sta
from sklearn.feature_selection import SelectKBest, chi2

test_mses=[]

for i in range(1,len(X_train.columns)):
    # select k best features
    sk = SelectKBest(chi2, k=i)

    # filter only the k best features
    X_new = sk.fit_transform(X_train,y_train)
    X_new_test=sk.fit_transform(X_test,y_test)

    # fit linear regression on the k best features
    temp_lr = lin_reg.fit(X_new, y_train)

    # find mse on the test set
    temp_y_pred = temp_lr.predict(X_new_test)
    test_mse = mean_squared_error(y_test, temp_y_pred)

    # append mse to list
    test_mses.append(test_mse)

# value of k for which mse is lowest
optimal_k = np.argmin(test_mses) +1
```

In [93]:
```
# k vs mse
plt.figure(figsize=(10,5))
plt.plot(range(1,len(X_train.columns)), test_mses)
plt.xlabel('Number of features')
plt.ylabel('MSE')
plt.title('Number of features vs Test MSE values')
#plt.savefig('images/number_features_vs_test_MSE')
plt.show()
print('Optimal k:',optimal_k)
```




```
Optimal k: 4
```

I will now use the optimal k for our final model.

In [94]:
```
# final model
sk = SelectKBest(chi2, k=optimal_k)
X_new = sk.fit_transform(X_train,y_train)
X_new_test=sk.fit_transform(X_test,y_test)
```

In [95]:

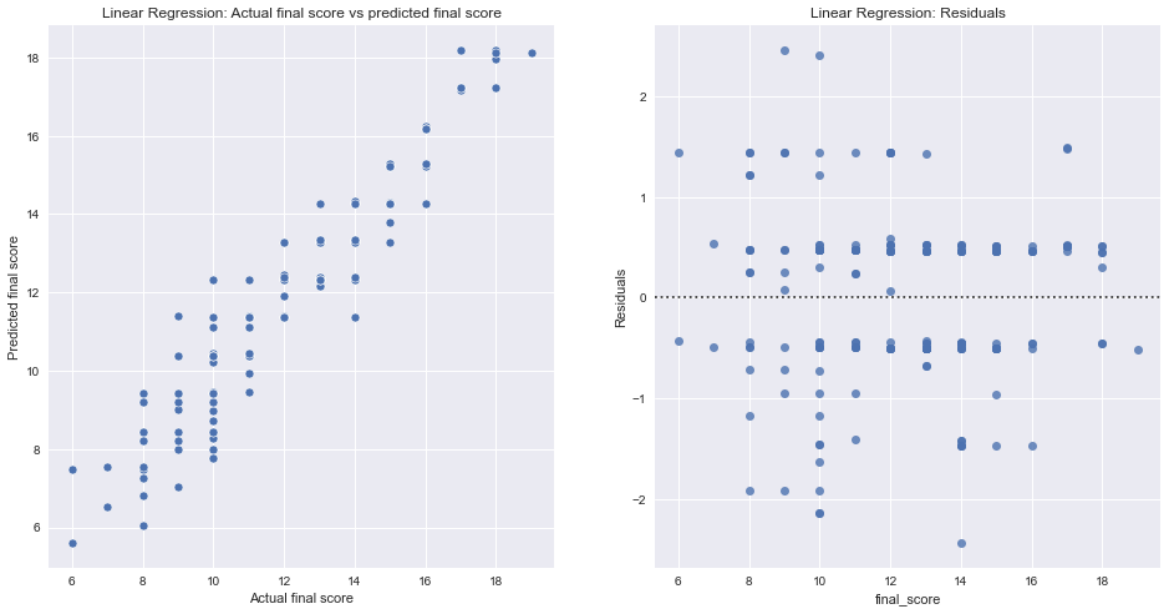
```
# create a linear regression model
lin_reg = LinearRegression()
# fit on the training data
lin_reg.fit(X_new, y_train)
# make predictions on the test set
y_pred = lin_reg.predict(X_new_test)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for Linear Regressor:", rmse)
print("MAE for Linear Regressor:", mae)
print("R-Squared for Linear Regressor:", r2)
```

```
RMSE for Linear Regressor: 0.8525840803275168
MAE for Linear Regressor: 0.681837184150793
R-Squared for Linear Regressor: 0.9081957672420428
```

I receive a strong R-Squared of 0.908. Let's look at the residual plots to evaluate model even further.

In [96]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('Linear Regression: Actual final score vs predicted final s
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('Linear Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/Linear_regression')
plt.show()
```



The residual plots seem ideal. I am happy with this model. I will now move to more complex models. Let's try KNN.

K-Nearest Neighbor

With knn, I will reuse the feature selection process, only the model being used will be a basic knn with 5 neighbors (arbitary choice for neighbors right now).

In [97]:
```
# finding optimal number of features to use in the model
# we are using chi-square test from sklearn https://scikit-learn.org/sta
knn_model = KNeighborsRegressor(n_neighbors=5)

test_mses=[]

for i in range(1,len(X_train.columns)):
    # select k best features
    sk = SelectKBest(chi2, k=i)

    # filter only the k best features
    X_new = sk.fit_transform(X_train,y_train)
    X_new_test=sk.fit_transform(X_test,y_test)

    # fit linear regression on the k best features
    temp_lr = knn_model.fit(X_new, y_train)

    # find mse on the test set
    temp_y_pred = temp_lr.predict(X_new_test)
    test_mse = mean_squared_error(y_test, temp_y_pred)

    # append mse to list
    test_mses.append(test_mse)

# number of features for which mse is lowest
optimal_num_features = np.argmin(test_mses) +1
```

In [98]:
```
print('The optimal number of features for the knn model are',optimal_num
```

```
The optimal number of features for the knn model are 4
```

In [99]:
```
# dataset to use
sk = SelectKBest(chi2, k=optimal_num_features)
X_new = sk.fit_transform(X_train,y_train)
X_new_test=sk.fit_transform(X_test,y_test)
```

Now I will optimize on a different k: the number of neighbors.

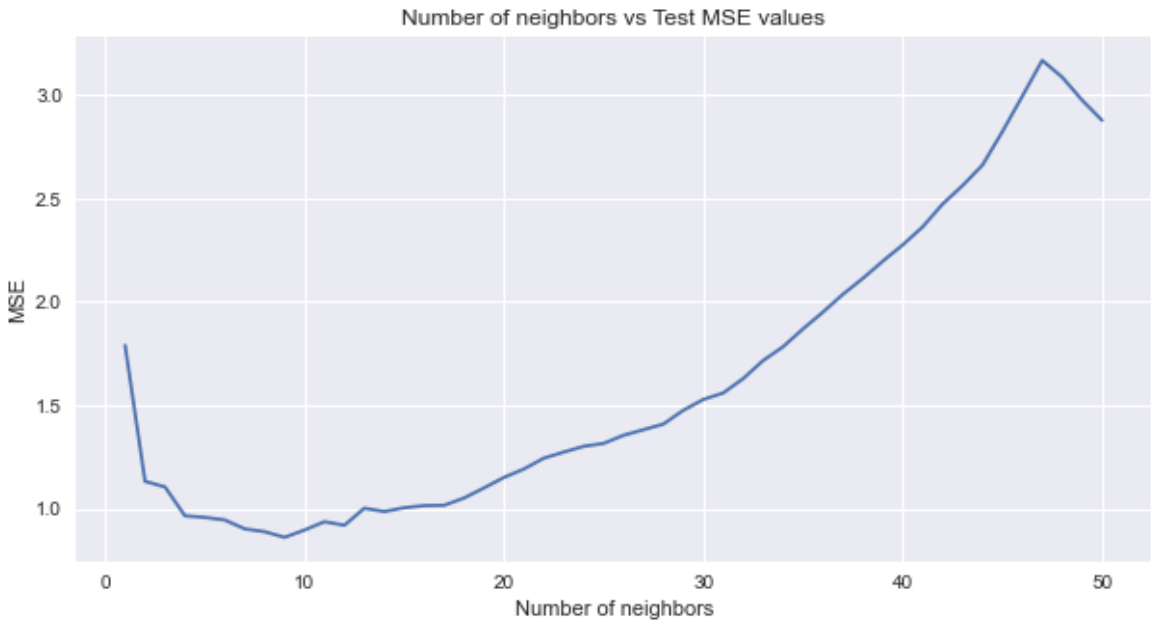
Let's run a grid search on k and find the optimal number of neighbors for the kNN model.

In [100]:

```
knn_mse = {}
# running a grid search for k between 1 and 50
for k in range(1,51):

    # defining a knn with k neighbors
    knn_model = KNeighborsRegressor(n_neighbors=k)
    knn_model.fit(X_new, y_train)
    y_pred = knn_model.predict(X_new_test)

    #Storing MSE
    mse = mean_squared_error(y_test,y_pred)
    knn_mse[k] = mse

# value of k for which mse is lowest
optimal_k = np.argmin(list(knn_mse.values())) +1
```

In [101]:

```
# k vs mse
plt.figure(figsize=(10,5))
plt.plot(knn_mse.keys(), knn_mse.values())
plt.xlabel('Number of neighbors')
plt.ylabel('MSE')
plt.title('Number of neighbors vs Test MSE values')
#plt.savefig('images/Number_of_neighbor_test_MSE')
plt.show()
```



Using the elbow method, I chose 8 as our k value. Now running our final knn model:

In [102]:

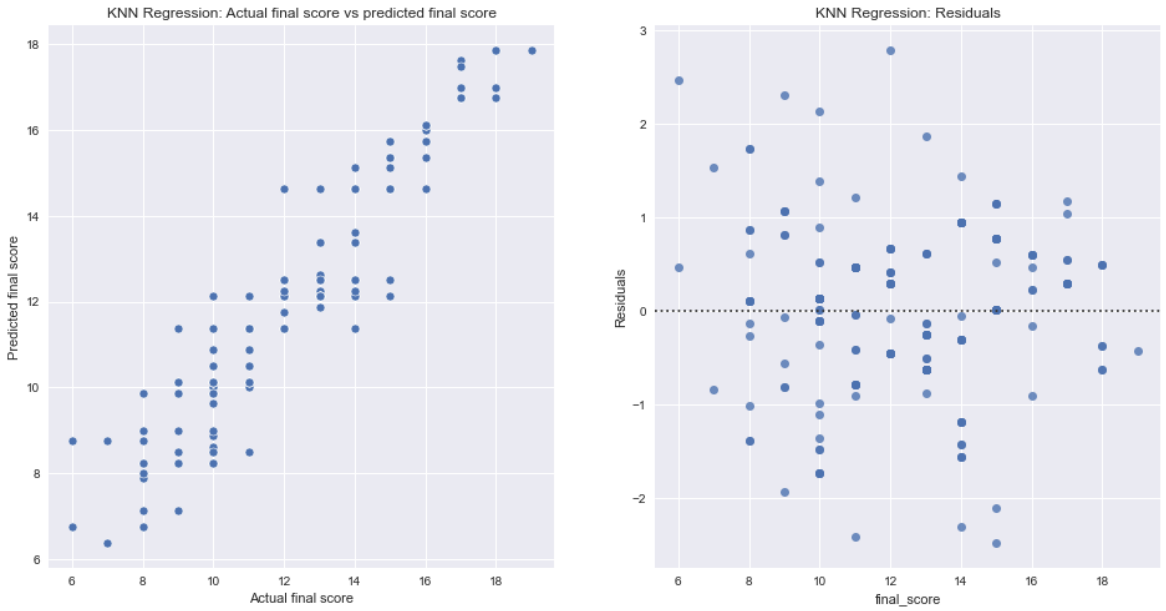
```
# create a knn regressor model
knn_model = KNeighborsRegressor(n_neighbors=8)
# fit on the training data
knn_model.fit(X_new, y_train)
# make predictions on the test set
y_pred = knn_model.predict(X_new_test)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for KNN Regressor:", rmse)
print("MAE for KNN Regressor:", mae)
print("R-Squared for KNN Regressor:", r2)
```

```
RMSE for KNN Regressor: 0.9426054450965303
MAE for KNN Regressor: 0.7173366834170855
R-Squared for KNN Regressor: 0.887785712691113
```

Again I receive a good R-Squared, slightly lower than linear regression though.

In [103]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('KNN Regression: Actual final score vs predicted final scor
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('KNN Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/KNN_regression')
plt.show()
```



Feature Selection using Random Forest Model

I will now jump to tree models. But before that I will create a standard set of features to use for all the tree based models.

I will use random forest to select our features. This is because each tree of the random forest can calculate the importance of a feature according to its ability to increase the pureness of the leaves. This makes them excellent models for understanding feature importance.

Features that I select using random forest will now be used for all the remaining models

In [104]:

```
# First we build and train our Random Forest Model
# adding an arbitrary amount of trees
# we need to have a good enough number of trees to model the dataset, bu
# adding max_depth to avoid overfitting
rf = RandomForestRegressor(max_depth=10, random_state=42, n_estimators =

# calculate feature importance
feature_importances = pd.DataFrame(rf.feature_importances_, index =X_tra
feature_importances['cummulative_importance'] = feature_importances['imp
feature_importances
```

Out[104]:

	importance	cummulative_importance
period2_score	0.901942	0.901942
period1_score	0.012079	0.914022
absences	0.009044	0.923065
age	0.005706	0.928772
health	0.005128	0.933899
weekday_alcohol_usage	0.004852	0.938751
family_quality	0.004815	0.943566
go_out	0.004604	0.948170
free_time	0.004477	0.952647
commute_time	0.003902	0.956550
weekend_alcohol_usage	0.003329	0.959879
failures	0.002838	0.962716
father_education	0.002768	0.965484
mother_education	0.002673	0.968158
father_job_services	0.002610	0.970767
paid_classes_yes	0.002339	0.973107
father_job_other	0.002189	0.975296
study_time	0.002062	0.977358
reason_other	0.001631	0.978989
school_MS	0.001546	0.980534
nursery_yes	0.001499	0.982033
school_support_yes	0.001490	0.983523
family_size_LE3	0.001486	0.985010
mother_job_other	0.001444	0.986453
mother_job_services	0.001173	0.987627
romantic_yes	0.001154	0.988781

	importance	cummulative_importance
activities_yes	0.001127	0.989908
guardian_mother	0.001085	0.990993
family_support_yes	0.001084	0.992077
sex_M	0.000997	0.993075
reason_reputation	0.000904	0.993979
address_U	0.000862	0.994841
reason_home	0.000760	0.995601
parent_status_T	0.000727	0.996329
guardian_other	0.000637	0.996965
internet_yes	0.000618	0.997583
mother_job_teacher	0.000615	0.998198
desire_higher_edu_yes	0.000525	0.998722
mother_job_health	0.000521	0.999244
father_job_health	0.000435	0.999678
father_job_teacher	0.000322	1.000000

As `period1_score` and `period2_score` have undue advantage in predicting final_score, we will remove them while calculating feature importance and add them back later on

In [105]:
```
from sklearn.feature_selection import SelectFromModel

sel = SelectFromModel(RandomForestRegressor(max_depth=10, random_state=4

X_1 = X_train.drop(['period2_score','period1_score'],axis=1)
sel.fit(X_1, y_train)

selected_feat= list(X_1.columns[(sel.get_support())]) + ['period1_score'
```

In [106]:
```
print('Final columns selected for use with tree-based models are:',selec
```

```
Final columns selected for use with tree-based models are: ['age', 'mother_
education', 'father_education', 'study_time', 'failures', 'free_time', 'go_
out', 'weekday_alcohol_usage', 'weekend_alcohol_usage', 'health', 'absence
s', 'school_support_yes', 'paid_classes_yes', 'desire_higher_edu_yes', 'per
iod1_score', 'period2_score']
```

In [107]:
```
# finalizing training and test set for tree based models
X_train_new = X_train[selected_feat]
X_test_new = X_test[selected_feat]
```

Decision Tree Regsessor

I will run a decision tree regressor with the default setting, just to get an idea of the accuracy levels

In [108]:
```
# create a dTree regressor model
dTree = DecisionTreeRegressor(random_state=1)
# fit on the training data
dTree.fit(X_train_new, y_train)
# make predictions on the test set
y_pred = dTree.predict(X_test_new)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for dTree Regressor:", rmse)
print("MAE for dTree Regressor:", mae)
print("R-Squared for dTree Regressor:", r2)
```

```
RMSE for dTree Regressor: 1.205098048514171
MAE for dTree Regressor: 0.9095477386934674
R-Squared for dTree Regressor: 0.816585767226478
```

Not bad. Let's try hyperparameter optimization using grid search and see where we can take this model.

First we will define the grid.

In [109]:
```
# parameters for grid search dTree
param_dict = {
    "criterion":["squared_error", "friedman_mse", "absolute_error", "poi
    "max_depth":range(1,15),
    "min_samples_split":range(1,10),
    "min_samples_leaf":range(1,10)
}
```

In [110]:
```
grid_dtree = GridSearchCV(DecisionTreeRegressor(), param_dict,verbose=1,
grid_dtree.fit(X_train_new,y_train)
grid_dtree.best_params_
```

```
Fitting 5 folds for each of 4536 candidates, totalling 22680 fits
```

Out[110]:
```
{'criterion': 'poisson',
 'max_depth': 4,
 'min_samples_leaf': 9,
 'min_samples_split': 2}
```

Now I will use these hyperparameters and fit the final dTree Model.

The best hyperparameters that we could find for Decision Tree from grid search are as follows:

- criterion : 'squared_error',
- max_depth : 4,
- min_samples_leaf : 9
- min_samples_split : 2

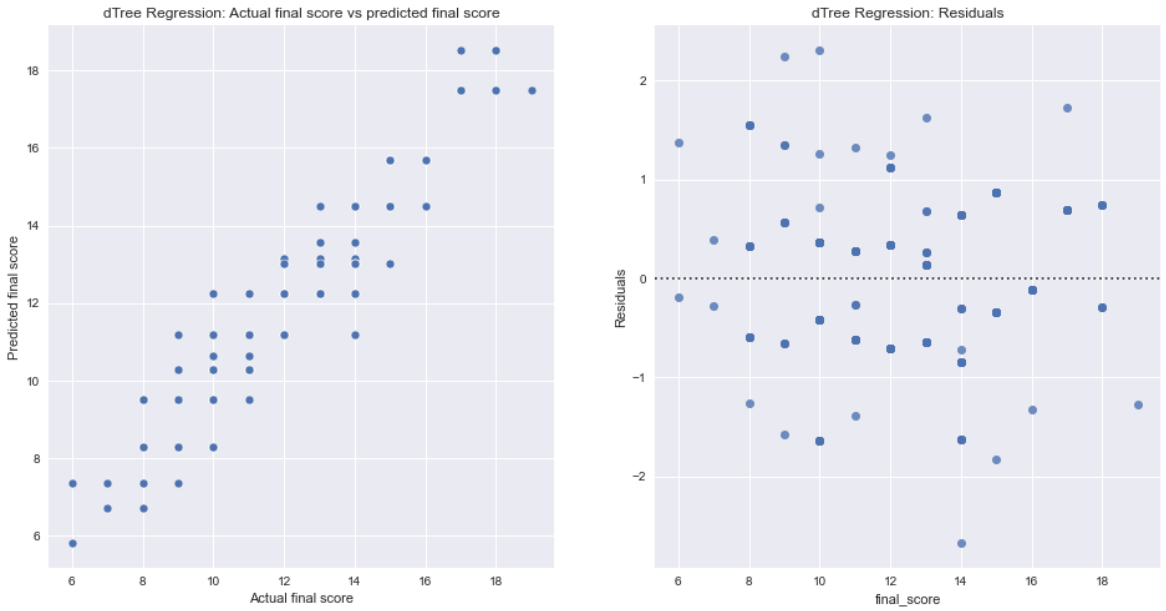
In [111]:
```
# create a dTree regressor model
dTree = DecisionTreeRegressor(criterion='squared_error',max_depth=4,min_
# fit on the training data
dTree.fit(X_train_new, y_train)
# make predictions on the test set
y_pred = dTree.predict(X_test_new)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for dTree Regressor:", rmse)
print("MAE for dTree Regressor:", mae)
print("R-Squared for dTree Regressor:", r2)
```

```
RMSE for dTree Regressor: 0.8351363771549553
MAE for dTree Regressor: 0.6794526992463412
R-Squared for dTree Regressor: 0.9119147746723426
```

I receive our best ever R-Squared so far of 0.91

In [112]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('dTree Regression: Actual final score vs predicted final sc
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('dTree Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/Decision_tree')
plt.show()
```



I can also look at the importance of the features using the trained decision tree model. Let's look at the five most important features according to our decision tree model.

In [113]:
```
# Importance calculation
pd.DataFrame(dTree.feature_importances_, columns = ["Imp"], index = X_tr
```

Out[113]:

	Imp
period2_score	0.994800
period1_score	0.002296
health	0.001607
age	0.000702
failures	0.000595

Interestingly, number of failures seem to be an important factor as well.

Random Forest Regressor

Now, I will use Random Forest to model our dataset. I will now optimize the hyper-parameters.

As Random Forest take considerable time to train, I will run a Randomized Search instead of a grid search for hyper-parameter tuning. Let's define the parameters that will be considered.

In [114]:

```
# parameters for grid search dTree
param_dict = {
    "criterion":["squared_error", "friedman_mse", "absolute_error", "poi
    "max_depth":range(1,15),
    "min_samples_split":range(1,10),
    "min_samples_leaf":range(1,10)
}

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, num

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 80, num = 10)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Method of selecting samples for training each tree
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap
              }

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf_model = RandomForestRegressor()

# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = RandomizedSearchCV(estimator = rf_model, param_distributions
rf_random.fit(X_train_new, y_train)
print(rf_random.best_params_)
```

In [115]:

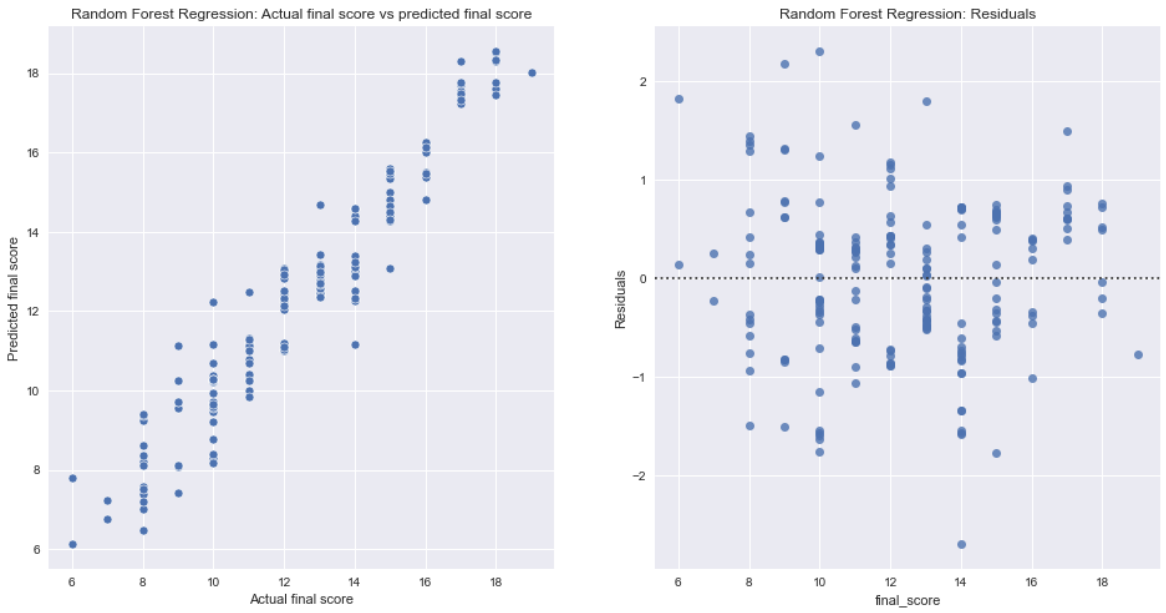
```python
# create a random forest regressor model
rf_model = RandomForestRegressor(n_estimators=90,
                                 max_depth=5,
                                 min_samples_leaf=4,
                                 min_samples_split=5,
                                 max_features='auto',
                                 bootstrap=True,
                                 random_state=1)
# fit on the training data
rf_model.fit(X_train_new, y_train)
# make predictions on the test set
y_pred = rf_model.predict(X_test_new)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for Random Forest Regressor:", rmse)
print("MAE for Random Forest Regressor:", mae)
print("R-Squared for Random Forest Regressor:", r2)
```

```
RMSE for Random Forest Regressor: 0.8125775380679985
MAE for Random Forest Regressor: 0.6502622099198602
R-Squared for Random Forest Regressor: 0.9166092474278689
```

I improve our R-Squared even further to 0.9167. Let's look at the residual plots:

In [116]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('Random Forest Regression: Actual final score vs predicted
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('Random Forest Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/Random_forest')
plt.show()
```



Five most important features according to the Random Forest model:

In [117]:
```
# Importance calculation for Random Forest
pd.DataFrame(rf_model.feature_importances_, columns = ["Imp"], index = X
```

Out[117]:

	Imp
period2_score	0.970247
period1_score	0.007183
health	0.002956
absences	0.002771
go_out	0.002597

Ada Boost Regressor

As tree-based ensemble models are doing well, let's try Ada Boost. Let's run a grid search to optimize on the hyper-parameters.

In [*]:

```
ada_boost_model = AdaBoostRegressor(base_estimator=DecisionTreeRegressor

parameters = {'base_estimator__max_depth':[i for i in range(2,11,2)],
              'base_estimator__min_samples_leaf':[i for i in range(4,14,
              'n_estimators':[50,100,250,300,350],
              'learning_rate':[0.01, 0.025, 0.05, 0.075, 0.1],
              'base_estimator__max_features':['auto']
             }

ada_boost_grid = GridSearchCV(ada_boost_model, parameters,verbose=100,sc
ada_boost_grid.fit(X_train,y_train)
ada_boost_grid.best_params_
```

```
Fitting 5 folds for each of 625 candidates, totalling 3125 fits
[CV 1/5; 1/625] START base_estimator__max_depth=2, base_estimator__ma
x_features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.
01, n_estimators=50
[CV 1/5; 1/625] END base_estimator__max_depth=2, base_estimator__max_
features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.0
1, n_estimators=50;, score=0.819 total time=   0.0s
[CV 2/5; 1/625] START base_estimator__max_depth=2, base_estimator__ma
x_features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.
01, n_estimators=50
[CV 2/5; 1/625] END base_estimator__max_depth=2, base_estimator__max_
features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.0
1, n_estimators=50;, score=0.784 total time=   0.0s
[CV 3/5; 1/625] START base_estimator__max_depth=2, base_estimator__ma
x_features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.
01, n_estimators=50
[CV 3/5; 1/625] END base_estimator__max_depth=2, base_estimator__max_
features=auto, base_estimator__min_samples_leaf=4, learning_rate=0.0
1, n_estimators=50;, score=0.850 total time=   0.0s
```

I will now use the best parameters received to create our final ada boost model and report the accuracies.

In [*]:

```
ada_boost_model = AdaBoostRegressor(base_estimator=DecisionTreeRegressor


                                    learning_rate=0.075,
                                    n_estimators=250)

# fit on the training data
ada_boost_model.fit(X_train_new, y_train)
# make predictions on the test set
y_pred = ada_boost_model.predict(X_test_new)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE for AdaBoost Regressor:", rmse)
print("MAE for AdaBoost Regressor:", mae)
print("R-Squared for AdaBoost Regressor:", r2)
```

AdaBoost performed well but not as well as Random Forest. I received a R-Square of 0.906

In [*]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('Random Forest Regression: Actual final score vs predicted
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('Random Forest Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/ada_boost')
plt.show()
```

Let's move to our final ensemble model XGBoost Regressor

XGBoost Regsessor

Let's try XGBoost Regressor now. First we will do hyperparameter optimization.

In [*]:

```
## Hyper Parameter Optimization
n_estimators = [100, 200, 300, 400,500]
max_depth = [2, 3, 5, 10, 15]
learning_rate=[0.05,0.1,0.15,0.20]
min_child_weight=[1,2,3,4]

# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
    'min_child_weight':min_child_weight,
    "gamma": np.linspace(0, 5,10),
    "eta":np.linspace(0.1,0.3,10)
    }

# Set up the random search with 5-fold cross validation
random_xgb = RandomizedSearchCV(estimator=xgb.XGBRegressor(),
            param_distributions=hyperparameter_grid,
            cv=5,
            n_iter=100,
            scoring = 'neg_mean_absolute_error',
            n_jobs = 4,
            verbose = 100,
            return_train_score = True,
            random_state=42)

random_xgb.fit(X_train_new,y_train)
print(random_xgb.best_params_)
```

In [*]:

```
# the optimized model
xgb_reg = xgb.XGBRegressor(n_estimators=400, eta=0.18, max_depth=2, gamm
                           min_child_weight=4, learning_rate=0.2)
xgb_reg.fit(X_train_new, y_train)

# fit on the training data
xgb_reg.fit(X_train_new, y_train)

# make predictions on the test set
y_pred = xgb_reg.predict(X_test_new)

# calculate root mean squared error on test set
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# calculate mean absolute error on test set
mae = mean_absolute_error(y_test, y_pred)

# calculate r2-squared on test set
r2 = r2_score(y_test, y_pred)

print("RMSE for XGBoost Regressor:", rmse)
print("MAE for XGBoost Regressor:", mae)
print("R-Squared for XGBoost Regressor:", r2)
```

XGBoost gave us a very good accuracy but considering the time it takes to train and factors like interpretability of the features, I will prefer Random Forest.

In [*]:

```
fig, ax =plt.subplots(figsize=(16,8),nrows=1,ncols=2)

b1 = sns.scatterplot(x=y_test, y=y_pred, ax=ax[0])
b1.set_title('XGBoost Regression: Actual final score vs predicted final
b1.set_xlabel('Actual final score')
b1.set_ylabel('Predicted final score')

b2 = sns.residplot(x=y_test, y=y_pred, ax=ax[1])
b2.set_title('XGBoost Regression: Residuals')
b2.set_ylabel('Residuals')
#plt.savefig('images/XGBoost')
plt.show()
```

These were all the models that we use. Let's compare their performance now.

Evaluation

Here are the testing accuracies for all the 6 hyper-parameter tuned models and the baseline dummy model. I tested each one 5 times(5 different samples of testing set) and presented the mean accuracy metrics for all of these models.

Model Name	RMSE	MAE	R-Squared
Baseline Regressor	2.86	2.37	-0.033
Linear Regression	0.85	0.68	0.908
KNN	0.94	0.71	0.888
Decision Tree	0.83	0.68	0.912
Random Forest	**0.81**	**0.65**	**0.917**
AdaBoost	0.86	0.69	0.906
XGBoost	0.82	0.65	0.915

Fig.2 Mean Accuracy Metrics for all of our ML models

The best accuracy was received with the Random Forest Model. Random Forest are excellent for analysis as they are very robust models and also help provide key insights from the model.

For example, Random Forest gave us the subset of features that are most useful in predicting student's final score. These are:

1. `study_time`
2. `mother_education`
3. `father_education`
4. `failures`
5. `free_time`
6. `go_out`
7. `weekday_alcohol_usage`
8. `weekend_alcohol_usage`
9. `health`
10. `absences`
11. `school_support`
12. `paid_classes`
13. `desire_higher_edu`
14. `period1_score`
15. `period2_score`
16. `age`

Conclusion

Student's academic success is usually thought to be dependent on student's dedication and hardwork. This analyses pointed out that while those parameters are important, there are other background factors that impact student's final score.

The Exploratory Data Analysis revealed many interesting ideas from the data such as:

1. Father's education status seems to be more relevant to student's success.
2. Students living with both their parents or stable households seem to be perform better.
3. Students who wish to go for higher education perform better in school.
4. Increased commute time has a negative impact on academic performance.
5. Increasing weekly study time perform better in final score.
6. Student’s who has better internet access, perform better.
7. Alcohol consumption also has a negative impact on academic performance

These are just some of the insights that were found in the data. These ideas were than supported by the machine learning modeling stage which showed that the same features were important for predicting final score.

I selected Random Forest Regressor as our final model where we received a mean R-Squared of 0.92.

Future Work

While I have covered a lot of bases in this analysis, more can be done to improve the analysis further.

It would be interesting to implement an online system where we look at these factors while a student is studying and then conduct experiments to see, if additional school support can help students who are predicted to perform poorly.

Many of the analyses presented in this report can be used as guidelines and recommendations for school districts. For example, providing students, who don't have access to internet at home, with mobile hotspots.

I can also study school dropouts in more details and analyze the factors that cause dropout in more detail.

In the next iteration, I would also like to deploy our ML model as web application which is integrated with the school district rostering systems. This will allow for schools to continuously monitor the progress of its students and take corrective measures if required.