# Project 2 - Bayesian Linear Regression for the Prediction of Student Grades


## Name: Tatenda Kanyere

## Student Number: 19021756


## Date: 02/04/20

## Abstract

This report discovers the merit of Bayesian linear regression as a predictive model, comparing it to ordinary least squares regression, ridge regression and random forest. The Portugal student grades dataset was used, specifically the maths grades dataset. It was found that Bayesian linear regression only marginally outperformed the OLS regression and performed comparably on this dataset to ridge regression. Performing a similar model comparison in the future, with data containing a larger output spread and more correlated variables, may bear more conclusive results as to which model has the best predictive value.


## Introduction

Bayesian linear regression is a combination of Bayesian inference and traditional linear regression. Rather than just providing a single predictive figure as a regular linear regression would, Bayesian linear regression returns a distribution, highlighting the likelihood of a particular outcome being in that range, thus giving us a level of confidence that the true figure will lie within a specific range. This can be extremely useful in situations such as investments, where although the exact return might not be possible to predict, having a distribution based on priors gives the investor a distribution for their return, allowing them to see the chances of having both a higher and lower return than expected, as well as being able to retrospectively judge their actual returns against their projected BLR distribution, and updating the model. This is not something that is available when you implement traditional methods of regression. In order to test just how well Bayesian linear regression performs, it will be put up against three regression models (ridge regression, OLS regression and random forest regression), and using mean absolute error and root mean squared error as model metrics, we aim to determine which of the regression methods is the most effective. We hypothesise that the Bayesian linear regression is going to have a lower mean absolute error and root mean squared error than the other regression models. The null hypothesis is that there will be no significant difference between BLR and the other three regression models.

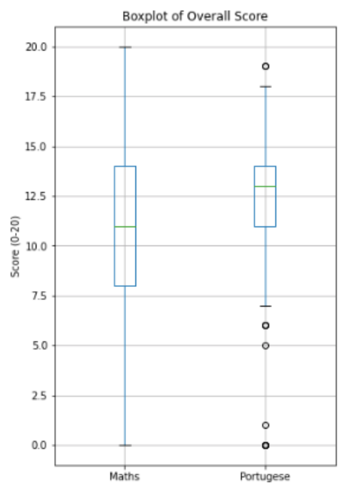
## Main Results

### Task 1

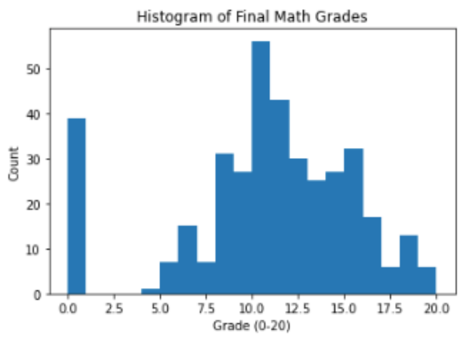
The first thing to look at is the data itself, in order to ensure that the data chosen is correct, it needs to be viewed and examined. This was done by showing a pre-view of the data (figure 1), then producing a boxplot (figure 2) to analyse the overall spread of grades amongst the students for both mathematics and Portuguese grades. Two histograms were also produced for the maths grades (figure 3) and Portuguese grades (figure 4) to see the details of the spread, such as whether or not there was more than one mode, or if the measures of central tendency are affected by outliers.

| | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob | Fjob | reason | guardian_x | traveltime_x | studytime_x | failures_x | schoolsup_x | fan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | GP | F | 18 | U | GT3 | A | 4 | 4 | at_home | teacher | course | mother | 2 | 2 | 0 | yes | |
| **1** | GP | F | 17 | U | GT3 | T | 1 | 1 | at_home | other | course | father | 1 | 2 | 0 | no | |
| **2** | GP | F | 15 | U | LE3 | T | 1 | 1 | at_home | other | other | mother | 1 | 2 | 3 | yes | |
| **3** | GP | F | 15 | U | GT3 | T | 4 | 2 | health | services | home | mother | 1 | 3 | 0 | no | |
| **4** | GP | F | 16 | U | GT3 | T | 3 | 3 | other | other | home | father | 1 | 2 | 0 | no | |
| **5** | GP | M | 16 | U | LE3 | T | 4 | 3 | services | other | reputation | mother | 1 | 2 | 0 | no | |
| **6** | GP | M | 16 | U | LE3 | T | 2 | 2 | other | other | home | mother | 1 | 2 | 0 | no | |
| **7** | GP | F | 17 | U | GT3 | A | 4 | 4 | other | teacher | home | mother | 2 | 2 | 0 | yes | |
| **8** | GP | M | 15 | U | LE3 | A | 3 | 2 | services | other | home | mother | 1 | 2 | 0 | no | |
| **9** | GP | M | 15 | U | GT3 | T | 3 | 4 | other | other | home | mother | 1 | 2 | 0 | no | |

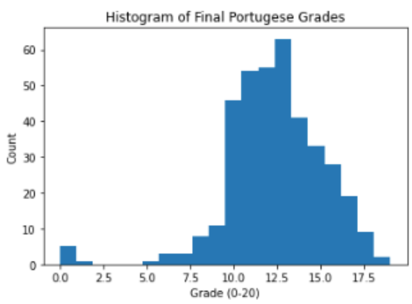
(figure 1: preview of merged data.)




(figure 2: boxplot of final grade for mathematics and Portuguese)




(figure 3: histogram of final math grades)




(figure 4: histogram of final Portuguese grades)

By implementing the built-in function '.head()' in python, it allows for a preview of the data to be read, allowing us to ensure that the data in the data frame is as expected, and allows us to see the column titles for future reference. The boxplot gives us the minimum, maximum and the 25% 50% and 75% percentile scores,  however this can easily be skewed by outliers, so it is necessary to also view this data in the form of a histogram so we can have an idea of how the data is spread according to the number of students achieving each grade. As you can see from figure 1, the maths figures show that 40 of the students scored 0 on their maths test, showing it to be the third most attained grade. The same is not true for the Portuguese grades, as according to the boxplot, any grade lower than 7 is technically classed as an outlier. This level of detail is not available with just the boxplot alone, so combining the boxplot with the histograms gives us a more nuanced view of the data being handled.

Following the reading and examining of data, the statistical details of the data, particularly pertaining to the final grades of mathematics student were examined. Although it is possible to view the statistical details of all the columns in both datasets, the only data set that was used to conduct the rest of the analysis was the maths dataset and so therefore this was the most relevant dataset to provide statistical details.

```
count    395.000000
mean      10.415190
std        4.581443
min        0.000000
25%        8.000000
50%       11.000000
75%       14.000000
max       20.000000
Name: G3, dtype: float64
```

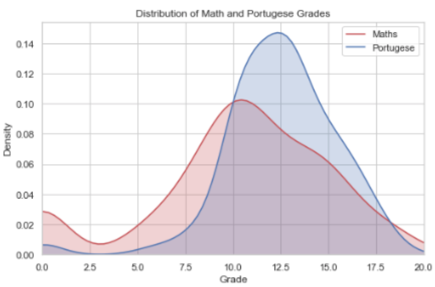
(figure 5: statistical details of the final grades)

| | age | Medu | Fedu | traveltime | studytime | failures | famrel | freetime |
|---|---|---|---|---|---|---|---|---|
| count | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 |
| mean | 16.696203 | 2.749367 | 2.521519 | 1.448101 | 2.035443 | 0.334177 | 3.944304 | 3.235443 |
| std | 1.276043 | 1.094735 | 1.088201 | 0.697505 | 0.839240 | 0.743651 | 0.896659 | 0.998862 |
| min | 15.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 |
| 25% | 16.000000 | 2.000000 | 2.000000 | 1.000000 | 1.000000 | 0.000000 | 4.000000 | 3.000000 |
| 50% | 17.000000 | 3.000000 | 2.000000 | 1.000000 | 2.000000 | 0.000000 | 4.000000 | 3.000000 |
| 75% | 18.000000 | 4.000000 | 3.000000 | 2.000000 | 2.000000 | 0.000000 | 5.000000 | 4.000000 |
| max | 22.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 3.000000 | 5.000000 | 5.000000 |

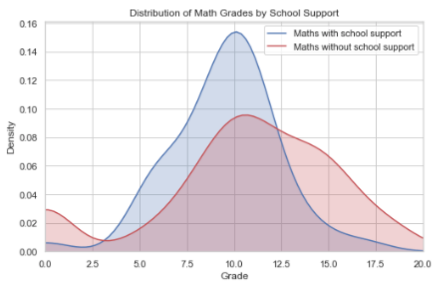
(figure 6: statistical details of all columns in the maths grades dataset)

By implementing the built-in function '.describe()' in Python, Figures 5 and 6 above provide us with the count, mean, standard deviation and percentiles for all of the numeric variables in the mathematics grades dataset. The percentiles coincide with the boxplot provided earlier in figure 2 so the analysis is consistent.

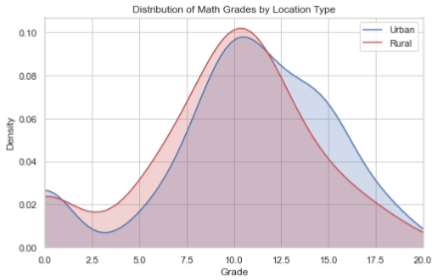
After viewing the statistical details of the data, the distributions of various features were then plotted, highlighting how different features, both numerical and categorical, affected the final grade achieved by students. Three of the most insightful distributions are highlighted in figures 7, 8 and 9.




(figure 7: distribution of math and Portuguese grades)




(figure 8: distribution of math grades by school support)



(figure 9: distribution of math grades by location type)

In order to achieve this visualisation, the seaborn module was used to produce a density plot, which allows for the comparison between different categorical variables with regards to the math grades achieved. In figure 7, we can see that the density of the Portuguese grades between 11 and 17 is much higher than the density for the same range in the maths grades. Overall you are more likely to get a higher grade in Portuguese than mathematics.
Figure 8 shows us that in mathematics, you are more likely to get a grade between 3 and 12.5 with school support than without school support, however grades above 12.5 are likely to be achieved by those who do not have school support. This may be due to those individuals who are capable do not need school support and are therefore more likely to achieve a higher grade. The distribution difference between urban and rural students is quite close, as rural and urban address students both have the same mode, however the urban students seem to perform better than rural students at the higher end of the grade distribution.

Following the plotting of various features, the next step was to produce a list of all the variables and see how they all correlate with our target feature, the final maths grade. This is shown in figures 10 and 11.

```
d1.corr()['G3'].sort_values()

failures     -0.360415
age          -0.161579
goout        -0.132791
traveltime   -0.117142
health       -0.061335
Dalc         -0.054660
Walc         -0.051939
freetime      0.011307
absences      0.034247
famrel        0.051363
studytime     0.097820
Fedu          0.152457
Medu          0.217147
G1            0.801468
G2            0.904868
G3            1.000000
Name: G3, dtype: float64
```
(figure 10: numerical variable correlations with final maths grade)

```
cat_d1=d1.select_dtypes(object)
dummy_d1=pd.get_dummies(cat_d1)
dummy_d1['G3']=d1['G3']
dummy_d1.corr()['G3'].sort_values()
```

```
higher_no           -0.182465
romantic_yes        -0.129970
Mjob_at_home        -0.115634
address_R           -0.105756
sex_F               -0.103456
paid_no             -0.101996
reason_course       -0.098950
internet_no         -0.098483
Mjob_other          -0.096477
guardian_other      -0.087774
schoolsup_yes       -0.082788
famsize_GT3         -0.081407
Pstatus_T           -0.058009
Fjob_other          -0.053483
nursery_no          -0.051568
school_MS           -0.045017
famsup_yes          -0.039157
reason_home         -0.021359
Fjob_services       -0.016108
activities_no       -0.016100
Fjob_at_home        -0.013385
activities_yes       0.016100
guardian_mother      0.022338
guardian_father      0.032493
famsup_no            0.039157
school_GP            0.045017
nursery_yes          0.051568
reason_other         0.052008
Fjob_health          0.057111
Mjob_teacher         0.057712
Pstatus_A            0.058009
Mjob_services        0.078429
famsize_LE3          0.081407
schoolsup_no         0.082788
Fjob_teacher         0.095374
reason_reputation    0.095692
internet_yes         0.098483
paid_yes             0.101996
sex_M                0.103456
address_U            0.105756
Mjob_health          0.116158
romantic_no          0.129970
higher_yes           0.182465
G3                   1.000000
```

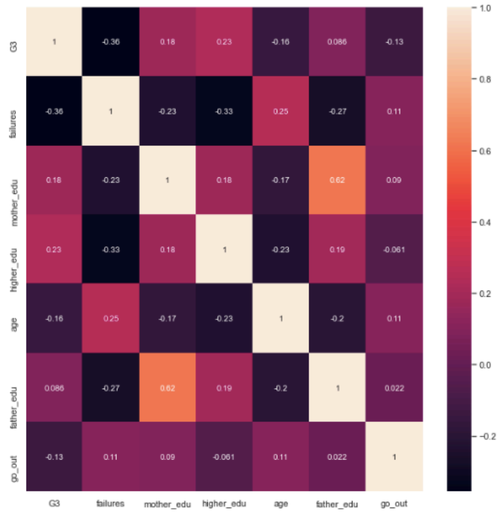
(figure 11: categorical variable correlations with final grade)

Producing the correlations for the numerical variables was fairly simple as it only required a simple python function, however in order to get the categorical variables, they had to be on-hot encoded using the pandas function, 'pd.get_dummies()', which changes the columns containing categorical variables to 1 (present) or 0 (missing), and each categorical input is classed as its own column, allowing us to determine how far a particular categorical variable such as not being in a relationship, correlates with the final maths grade achieved by students.
The next step in the process was to determine which six variables were the highest correlated with the final grade, regardless of direction (positive or negative). A function was implemented to complete this, as shown in figure 12.

```
# Function for the highest correlated variables
def most_correlated(d1):
    #Unnecessary or too highly correlted variables
    d1 = d1.drop(columns=['school', 'G1', 'G2'])

    # One-Hot Encoding of Categorical Variables
    d1 = pd.get_dummies(d1)

    # Find correlations with the Grade regardless of direction
    most_correlated = d1.corr().abs()['G3'].sort_values(ascending=False)

    #Pull the top 6 correlations
    most_correlated = most_correlated[:8]

    #Add the categorical variables and remove the binary opposite variable
    d1 = d1.loc[:, most_correlated.index]
    d1 = d1.drop(columns = ['higher_no'])

    return d1

d1=most_correlated(d1)
```

(figure 12: function for selecting the highest correlated variables)

The first step in this function was to remove the variables that are not useful in the models we will be implementing, so the school and mock grades of the students were removed from the data frame. Following this, the one-hot encoded categorical variables replace the previous categorical columns in the data frame, allowing us to rank the correlations of all the necessary columns. The top 6 most correlated variables were then used to replace the previous data frame, allowing us to proceed with only the top 6. Following this, it was noticed that the 'higher_no' variable was too highly correlated with the 'higher_yes' variable, which caused collinearity problems when implementing the Bayesian linear regression, so it had to be retrospectively removed from the data frame. A correlation matrix (figure 13) was produced to analyse the correlation and collinearity of all of the chosen variables.



(figure 14: correlation matrix of the top 6 most correlated variables)

As you can see in this figure, the most correlated variables to the final maths grade are failures, mother's education, intention to study at university, age, father's education and how often the students go out.

**Task 2**

In order to test the Bayesian linear regression results, it is important to measure it against some other statistical models to ensure that the results we gather are either more or less useful than other models already being used. In order to achieve this, a function was implemented (figure 15) to show the mean absolute error and root mean squared error of 3 statistical models (linear regression, ridge regression and random forest regression).

```
def compare(X_train, X_test, y_train, y_test):
    # Models to test
    model_name = ['Linear Regression','Random Forest',
                  'Ridge']
    X_train = X_train.drop(columns='G3')
    X_test = X_test.drop(columns='G3')

    model1 = LinearRegression()
    model2 = RandomForestRegressor(n_estimators=50)
    model3 = Ridge(alpha=1.0)

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

    # Train and test models for comparison
    for i, model in enumerate([model1, model2, model3]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

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

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

    return results
```
(figure 15: testing statistical models)

In this function, the mean absolute error and root mean squared error are produced for all three models, through training them then comparing the results against the test sets. The results gathered from the three models are provided in figure 16.

| | mae | rmse |
|---|---|---|
| Linear Regression | 3.55577 | 4.49314 |
| Random Forest | 3.8253 | 4.83371 |
| Ridge | 3.55137 | 4.48777 |

(figure 16: mean absolute error and root mean squared error of models)

Now that the statistical models have been evaluated, it was then time to implement the Bayesian linear regression. This process is outlined in figure 16.

```
# Formula for Bayesian Linear Regression
formula = 'G3 ~ ' + ' + '.join(['%s' % variable for variable in X_train.columns[1:]])
formula
```

```
'G3 ~ failures + mother_edu + higher_edu + age + father_edu + go_out'
```

```
# Context for the model
with pm.Model() as normal_model:

    # The prior for the model parameters will be a normal distribution
    family = pm.glm.families.Normal()

    # Creating the model requires the BLR formula and data
    pm.GLM.from_formula(formula, data = X_train, family = family)

    # Perform Markov Chain Monte Carlo sampling
    n_trace = pm.sample(draws=2000, chains = 2, tune = 500, target_accept=0.9)
```

(figure 16: implementing the Bayesian linear regression)

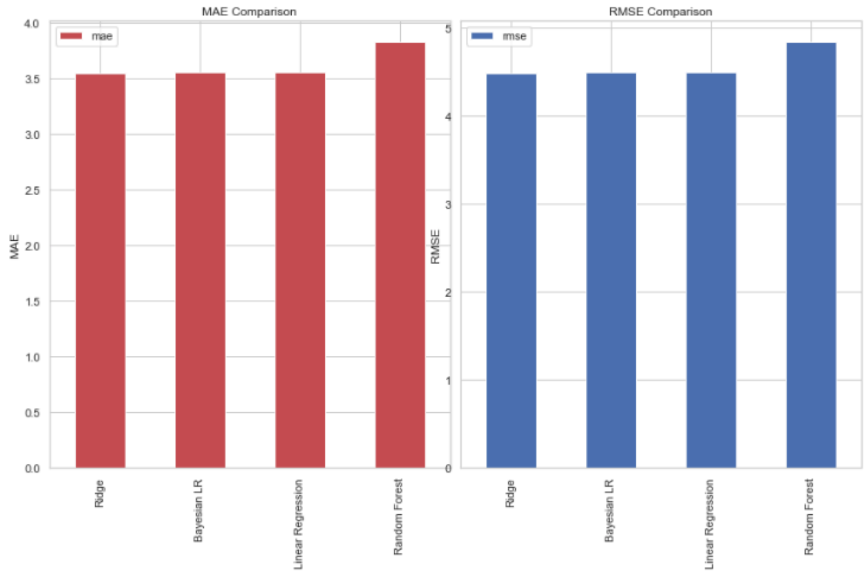
The first step was to provide a formula for the Bayesian linear regression, this simply consisted of the target variable 'G3' being determined by the 6 predictor variables as defined by our most correlated variables function. The prior for the model is a normal distribution, then the formula and data are implemented into the Bayesian linear regression. Following this, the trace for each variable is produced by Markov Chain Monte Carlo (MCMC) sampling. The general idea behind MCMC in this context stems from the fact that Bayesian linear regression return a distribution as a prediction, rather than a single figure. For Bayesian linear regression, your aim is to produce an accurate posterior distribution; so, the larger number of samples you have, the more accurate the posterior distribution is likely to be. Our trace is produced after 2000 samples, providing us with mean variable values from the trace (figure 17), this trace allows us to test the Bayesian linear regression on unseen (test) data.

```
Variable: Intercept      Mean weight in model: 12.1396
Variable: failures       Mean weight in model: -1.7785
Variable: mother_edu     Mean weight in model: 0.7221
Variable: higher_edu     Mean weight in model: 2.1258
Variable: age            Mean weight in model: -0.1504
Variable: father_edu     Mean weight in model: -0.5326
Variable: go_out         Mean weight in model: -0.3983
Variable: sd_log__       Mean weight in model: 1.4261
Variable: sd             Mean weight in model: 4.1663
```

(figure 17: mean weights of variables in the model)

From just looking at the figures produced by the trace, we can tell that the model we've produced is at least somewhat accurate as the negatively correlated variables have a negative mean weight, and the opposite is true for the variables that are positive correlated with the target variable.

To evaluate our Bayesian linear regression model, the standard mean absolute error and root mean squared error parameters were used, and the results were compared to the three models tested on the same dataset earlier, this is shown in figure 18.



(figure 18: mean of model parameters including Bayesian linear regression)

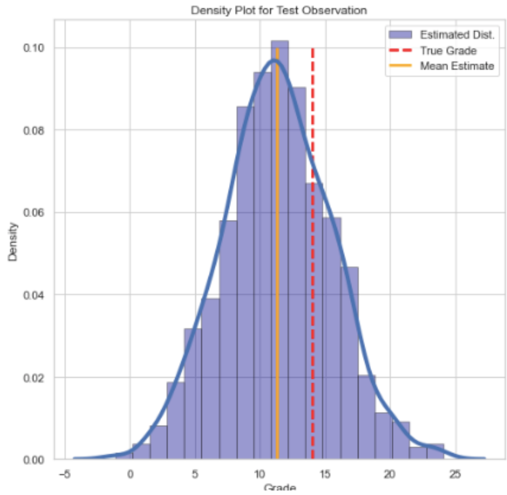
As you can see from figure 18, Bayesian linear regression outperforms random forest regression and ordinary least squares regression (although only slightly) in both MAE and RMSE scores but is narrowly bested by ridge regression on both metrics. The exact results from all four models are as detailed in figure 19.

| | mae | rmse |
|---|---|---|
| **Linear Regression** | 3.55577 | 4.49314 |
| **Random Forest** | 3.8253 | 4.83371 |
| **Ridge** | 3.55137 | 4.48777 |
| **Bayesian LR** | 3.55563 | 4.49228 |

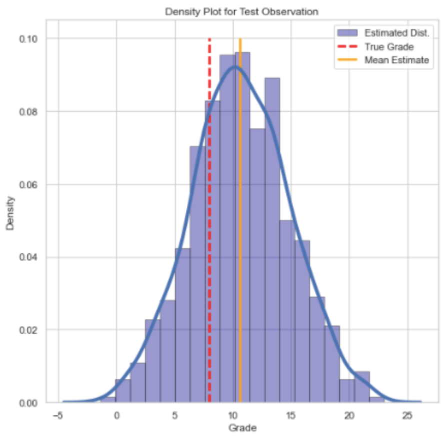
(figure 19: RMSE and MAE metric results)

Considering that the difference between the highest and lowest MAE and RMSE scores are within less than 1% of one another, in this instance it may be fair to say that although Bayesian linear regression does provide some inherent benefits as opposed to the other three statistical models, it doesn't provide us with a noticeably improved MAE or RMSE score. Perhaps in this particular instance, it may be apt to say that there you can only get more favourable MAE and RMSE scores with input data that is more correlated to the output, than the data provided by this dataset.

The final procedure to measure the accuracy of our Bayesian linear regression was to test the model on unseen instances of data and see what the distribution would look like when compared to the mean weights in the model and true grade. The results are provided in figures 20 and 21.




(figure 20: Prediction from unseen data)




(figure 21: prediction from unseen data)

Figure 20 is an example of the distribution underestimating the predicted grade of a particular student, whereas figure 21 is an example of the distribution overestimating the predicted grade of a particular student, although the same mean estimate is provided in both cases. Given the MAE and RMSE scores provided in figure 19, these visualisations appear to be reliable.

**Conclusions and future works**:

Analysing the performance of Bayesian linear regression throughout this report has required a number of skills from different areas of data science including machine learning, data visualisation. Perhaps the most useful skill when producing the report was feature selection, which allowed for the data that was used to be as useful as possible, weeding out data that was unnecessary due to low correlation, or introduced issues due to collinearity. One major issue that was confronted during the process of analysing the data was the fact that only one of the two maths and Portuguese datasets were able to be processed due to lack of technical skill. There was an attempt to incorporate both datasets into the Bayesian linear regression through merging, but as a result of a lack of technical skill to incorporate both, it was decided to only incorporate the maths dataset for the majority of the report. In future works, it may be of benefit to try and incorporate both datasets if it were to be used for the purpose of determining the usefulness of Bayesian linear regression.

The data visualisation allowed us to produce the results from the various analyses in an easily digestible manner, making the results easily interpreted by individuals who may not have specific domain knowledge.

This report has been able to conclude that in this particular example, Bayesian linear regression finds itself to be just as useful as ridge regression and ordinary least squares regression. This means we fail to reject our null hypothesis. The limits to the full potential of Bayesian linear regression seem to stem from the fact that the input data is not very correlated with the target data, so it can only be so accurate. There may have also been some benefit in selecting a dataset that has a wider spread in output range, as this may provide a more exaggerated difference between the performance of the regression models in MAE and RMSE scores. Although this report may be somewhat inconclusive as to the benefits of Bayesian linear regression, in future research, to truly measure the benefits of BLR as opposed to other statistical models, better input and output data correlations as well as a greater output range may amplify the performance of BLR by comparison.

**Appendix I**:

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

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

# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, median
_absolute_error

# Distributions
import scipy
```

In [54]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import pymc3 as pm
```

In [77]:

```
# Math Grades
d1=pd.read_csv("/Users/tatendakanyere/SDA/student/student-mat.csv",sep=None
,engine='python')
# Portugese Grades
d2=pd.read_csv("/Users/tatendakanyere/SDA/student/student-por.csv",sep=None
,engine='python')
# Merged data
d3=pd.merge(d1,d2, on = ["school","sex","age","address","famsize","Pstatus"
,"Medu","Fedu","Mjob","Fjob","reason","nursery","internet",])
```

```
pd.set_option('display.max_columns', None)
d3.head(10)
```

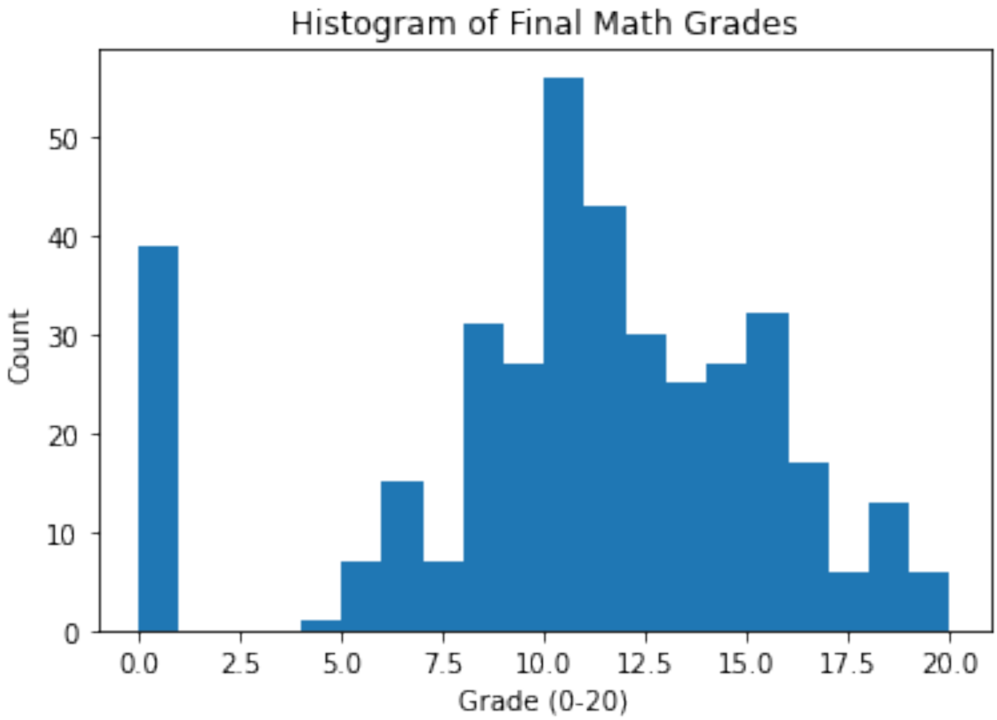
Out[77]:

| | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob | Fjob | reason | guardian_x | traveltime_x | studytime_x | failures_x | schoolsup_x | famsup_x | paid_x | activities_x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | GP | F | 18 | U | GT3 | A | 4 | 4 | at_home | teacher | course | mother | 2 | 2 | 0 | yes | no | no | no |
| **1** | GP | F | 17 | U | GT3 | T | 1 | 1 | at_home | other | course | father | 1 | 2 | 0 | no | yes | no | no |
| **2** | GP | F | 15 | U | LE3 | T | 1 | 1 | at_home | other | other | mother | 1 | 2 | 3 | yes | no | yes | no |
| **3** | GP | F | 15 | U | GT3 | T | 4 | 2 | health | services | home | mother | 1 | 3 | 0 | no | yes | yes | yes |
| **4** | GP | F | 16 | U | GT3 | T | 3 | 3 | other | other | home | father | 1 | 2 | 0 | no | yes | yes | no |
| **5** | GP | M | 16 | U | LE3 | T | 4 | 3 | services | other | reputation | mother | 1 | 2 | 0 | no | yes | yes | yes |
| **6** | GP | M | 16 | U | LE3 | T | 2 | 2 | other | other | home | mother | 1 | 2 | 0 | no | no | no | no |
| **7** | GP | F | 17 | U | GT3 | A | 4 | 4 | other | teacher | home | mother | 2 | 2 | 0 | yes | yes | no | no |
| **8** | GP | M | 15 | U | LE3 | A | 3 | 2 | services | other | home | mother | 1 | 2 | 0 | no | yes | yes | no |
| **9** | GP | M | 15 | U | GT3 | T | 3 | 4 | other | other | home | mother | 1 | 2 | 0 | no | yes | yes | yes |

The number of rows matches the number of students according to the merge document. _x= maths, _y=portugese

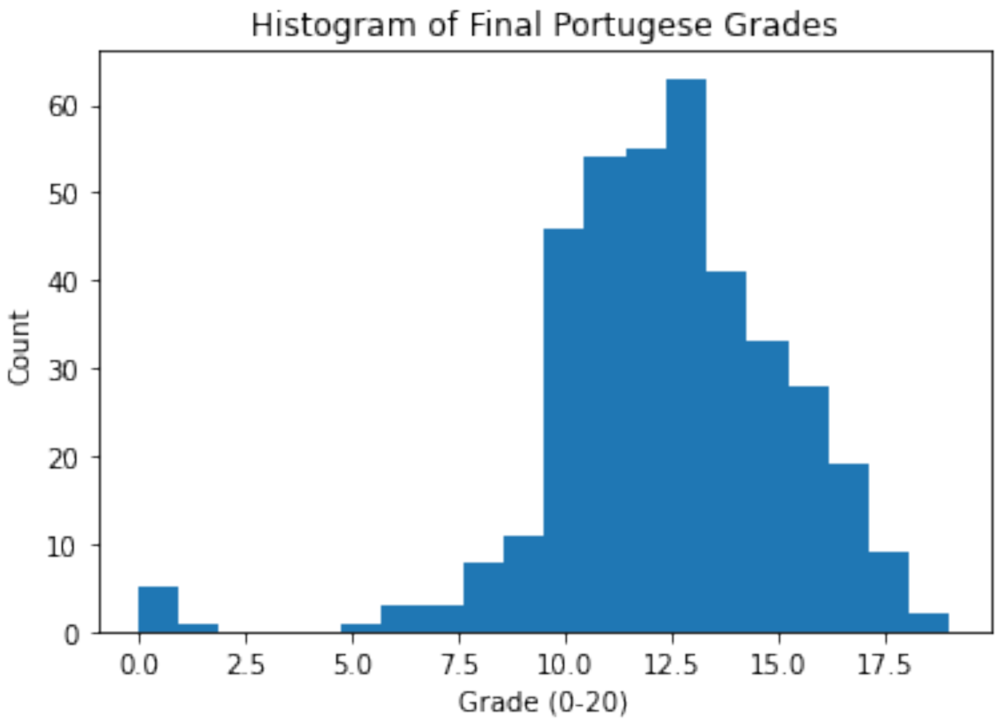
In [4]:

```
import matplotlib.pyplot as plt
# Histogram of math grades
plt.hist(d3['G3_x'], bins = 20)
plt.xlabel('Grade (0-20)')
plt.ylabel('Count')
plt.title('Histogram of Final Math Grades')
plt.show()
```




In [5]:

```
import matplotlib.pyplot as plt
# Histogram of portugese grades
plt.hist(d3['G3_y'], bins = 20)
plt.xlabel('Grade (0-20)')
plt.ylabel('Count')
plt.title('Histogram of Final Portugese Grades')
plt.show()
```

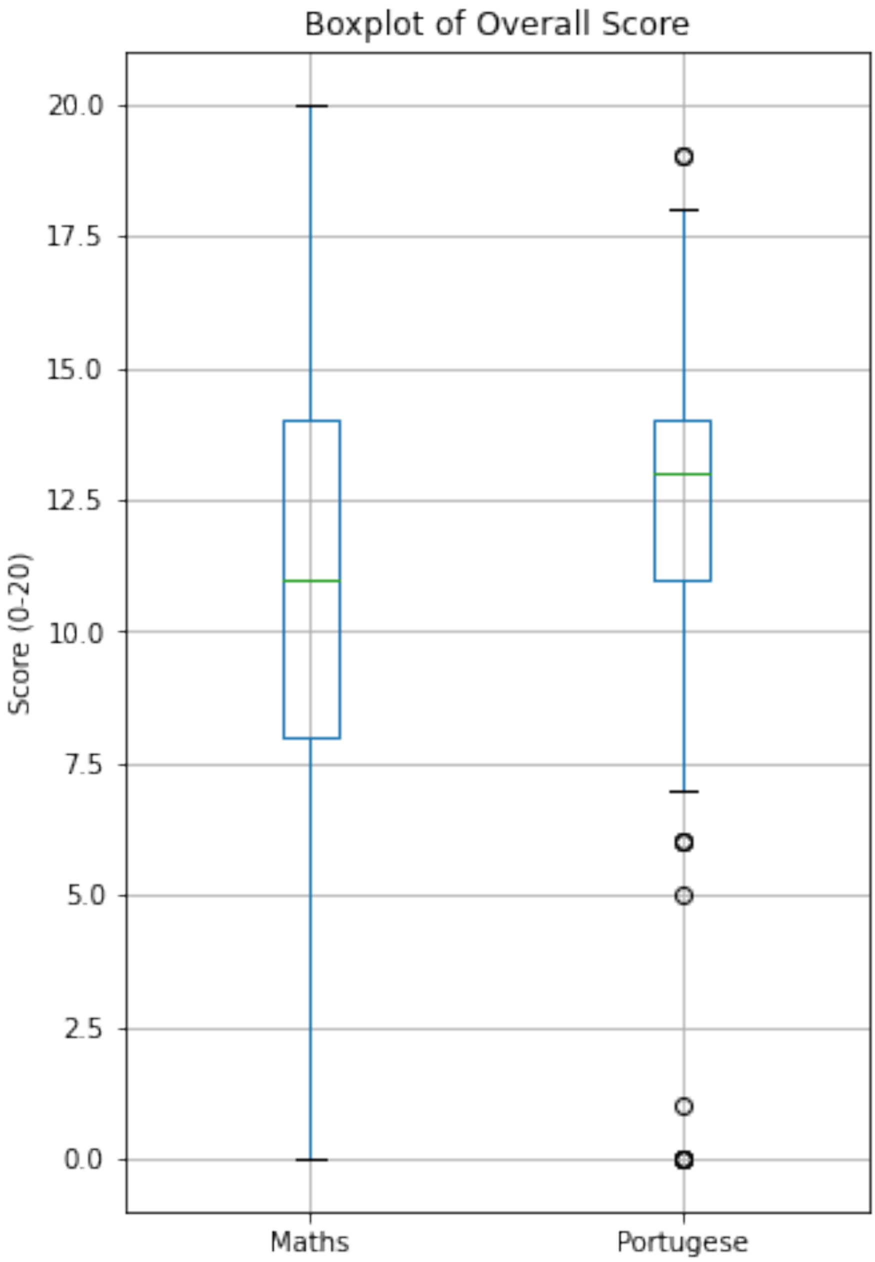



In [6]:

```
plt.figure(figsize=(5,8))
bp=d3.boxplot(column=['G3_x','G3_y'])
bp.set_xticklabels(['Maths','Portugese'])
bp.set_ylabel('Score (0-20)')
bp.set_title('Boxplot of Overall Score')
```

Out[6]:

```
Text(0.5, 1.0, 'Boxplot of Overall Score')
```

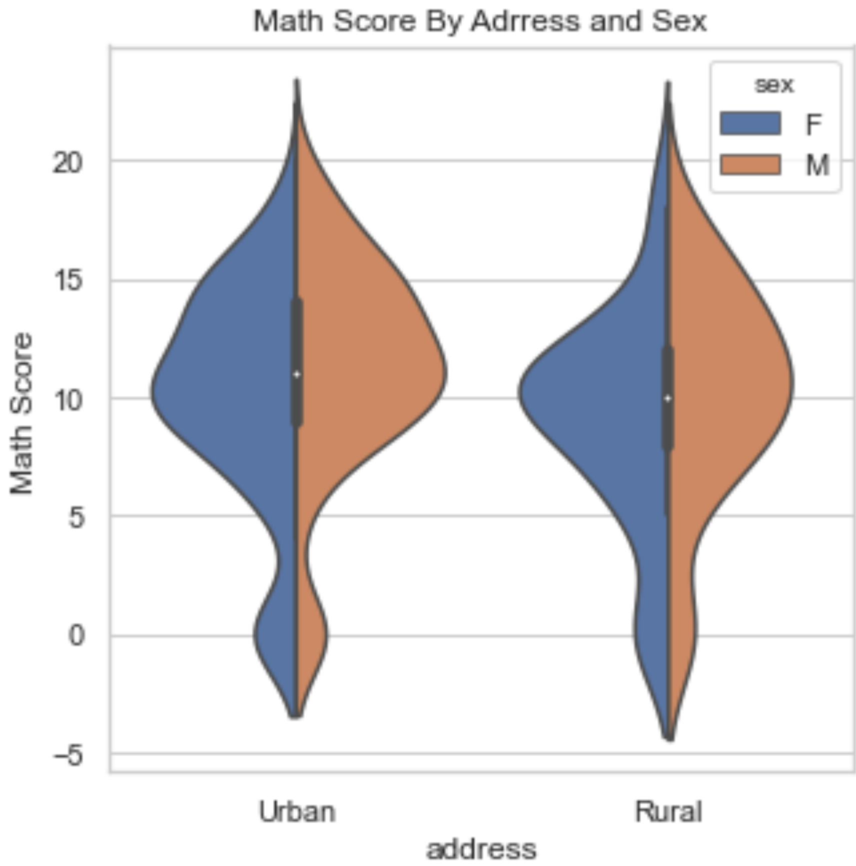



In [7]:

```
sns.set(style='whitegrid')
plt.figure(figsize=(5,5))
vp=sns.violinplot(y='G3_x',x='address',hue='sex',data=d3,split=True)
vp.set_xticklabels(['Urban','Rural'])
vp.set_ylabel('Math Score')
vp.set_title('Math Score By Adrress and Sex')
```

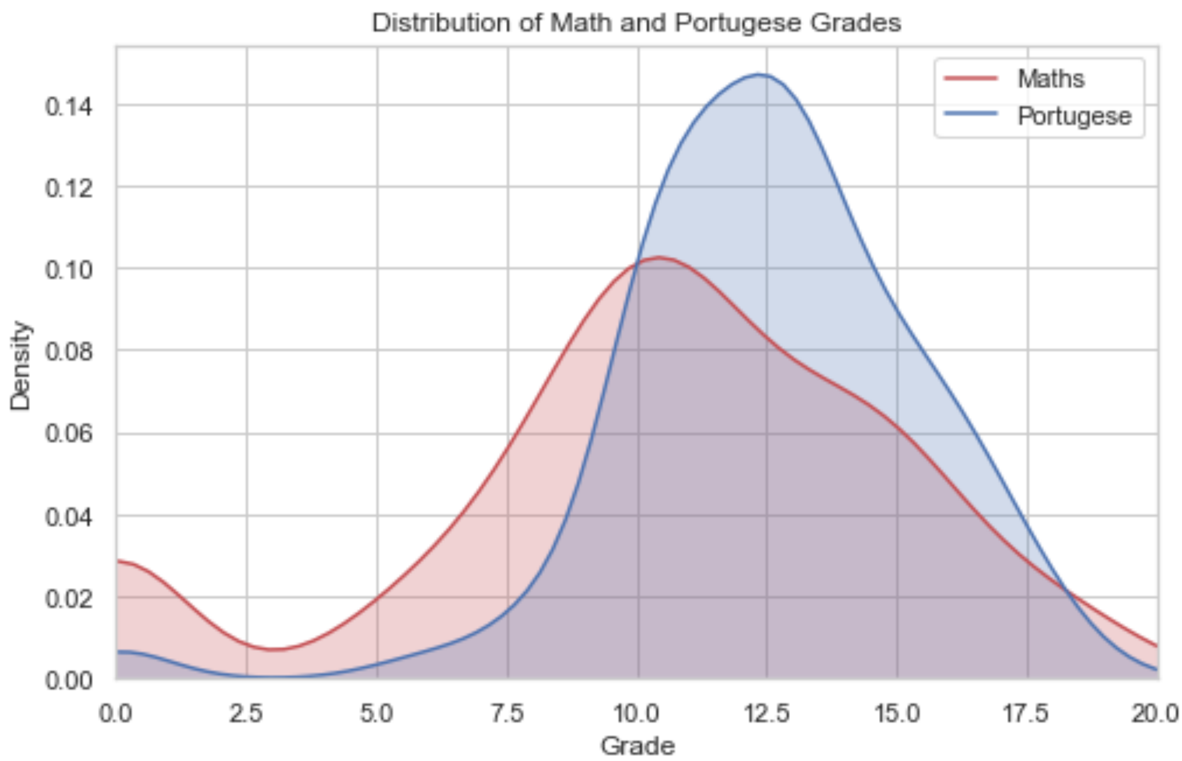
Out[7]:

```
Text(0.5, 1.0, 'Math Score By Adrress and Sex')
```



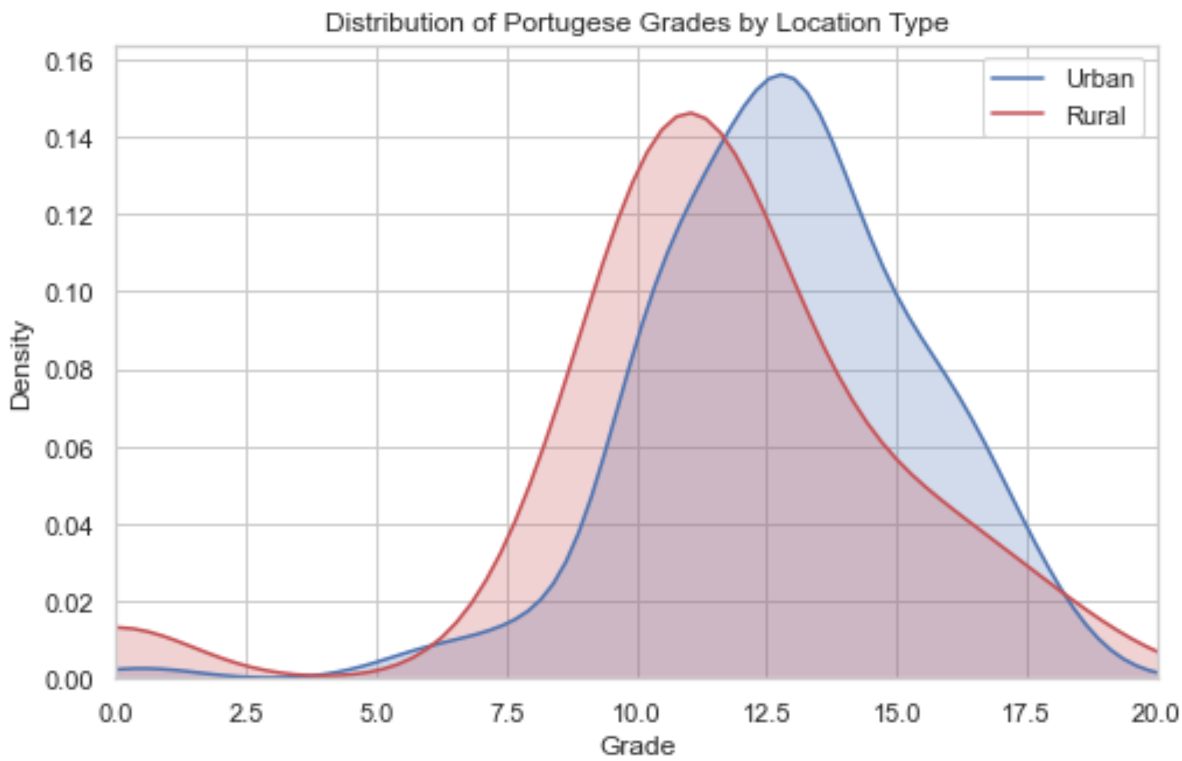

In [8]:

```
import seaborn as sns
#Plot distributions for a range of features (cat. and num.)
#Plot by Subject
plt.figure(figsize=(8,5))
plt.xlim(0,20)
plt.xlabel('Grade')
plt.ylabel('Density')
p1=sns.kdeplot(d3['G3_x'],shade=True,label='Maths',color='r')
p1=sns.kdeplot(d3['G3_y'],shade=True,label='Portugese',color='b')
plt.title('Distribution of Math and Portugese Grades')
plt.show()
```



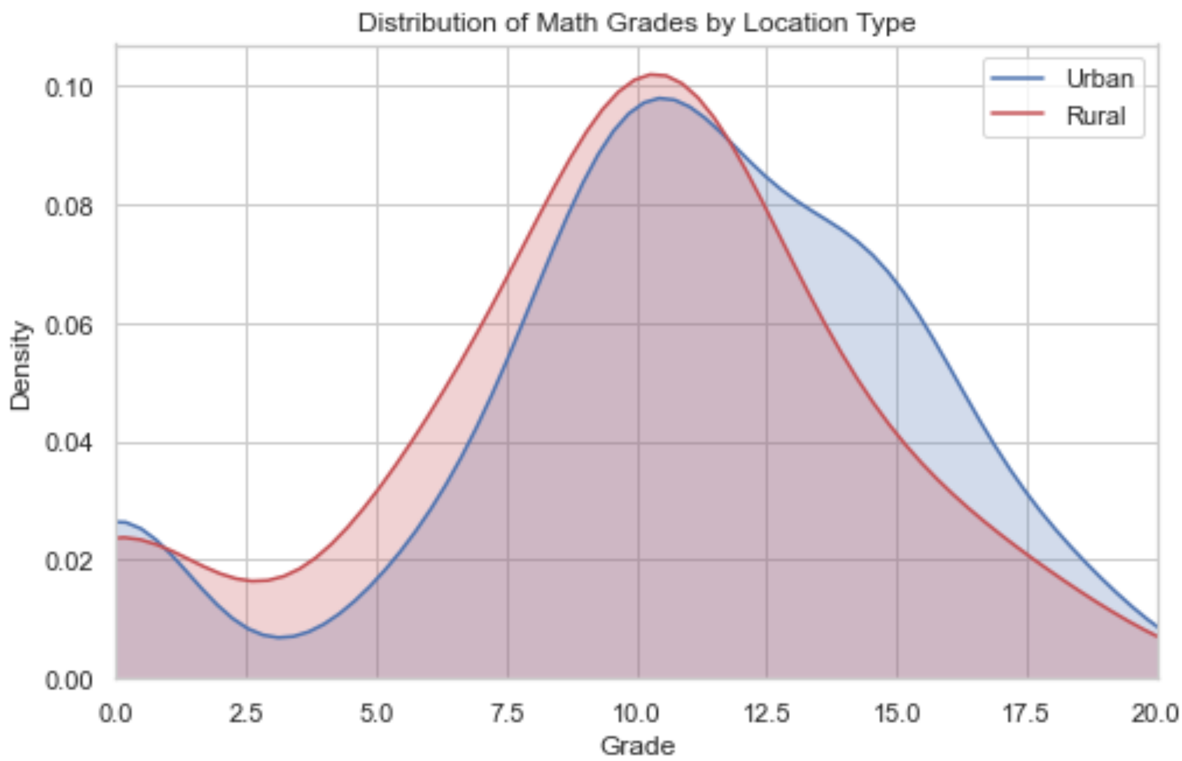

In [9]:

```
#Plot by Location Type
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['address']=='U'),'G3_y'],color='b',label='Urban',sha
de=
            True)
sns.kdeplot(d3.loc[(d3['address']=='R'),'G3_y'],color='r',label='Rural',sha
de=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Portugese Grades by Location Type')
plt.show()
```



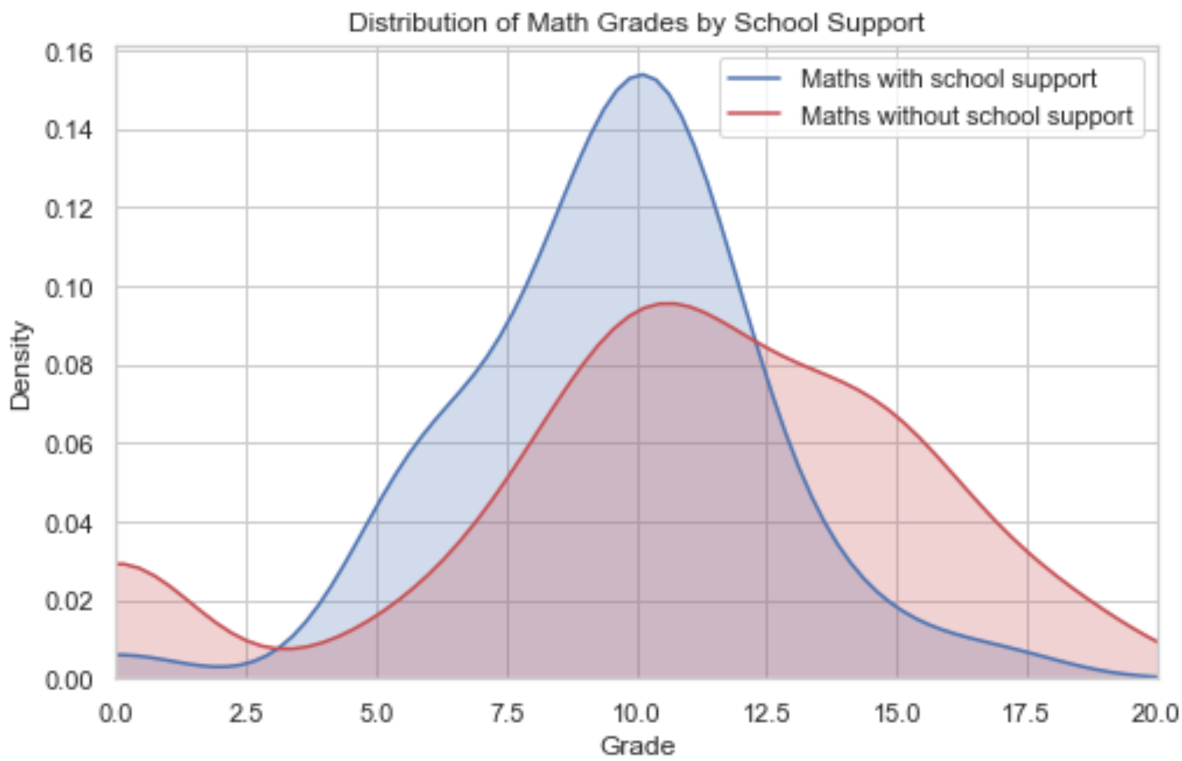

In [10]:

```
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['address']=='U'),'G3_x'],color='b',label='Urban',sha
de=
            True)
sns.kdeplot(d3.loc[(d3['address']=='R'),'G3_x'],color='r',label='Rural',sha
de=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Math Grades by Location Type')
plt.show()
```



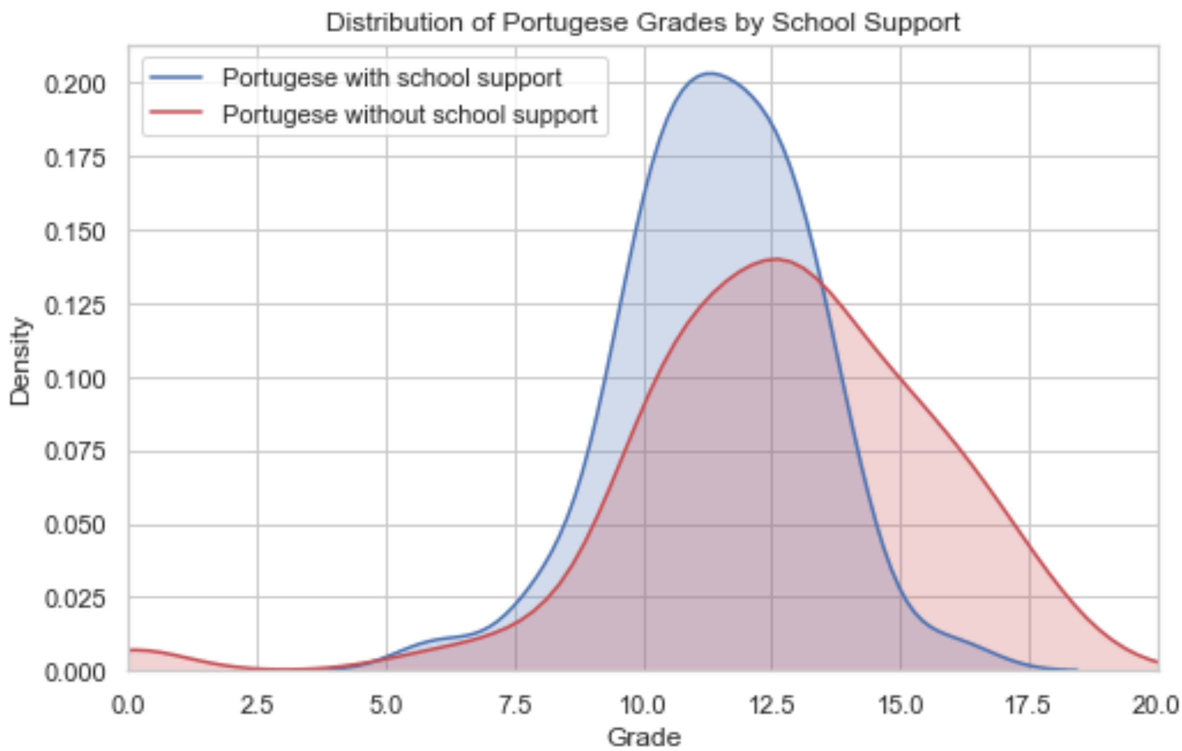

In [11]:

```
#Effect of School Support on Both Subjects
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['schoolsup_x']=='yes'),'G3_x'],color='b',label='Math
s with school support',shade=
            True)
sns.kdeplot(d3.loc[(d3['schoolsup_y']=='no'),'G3_x'],color='r',label='Maths
without school support',shade=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Math Grades by School Support')
plt.show()
```



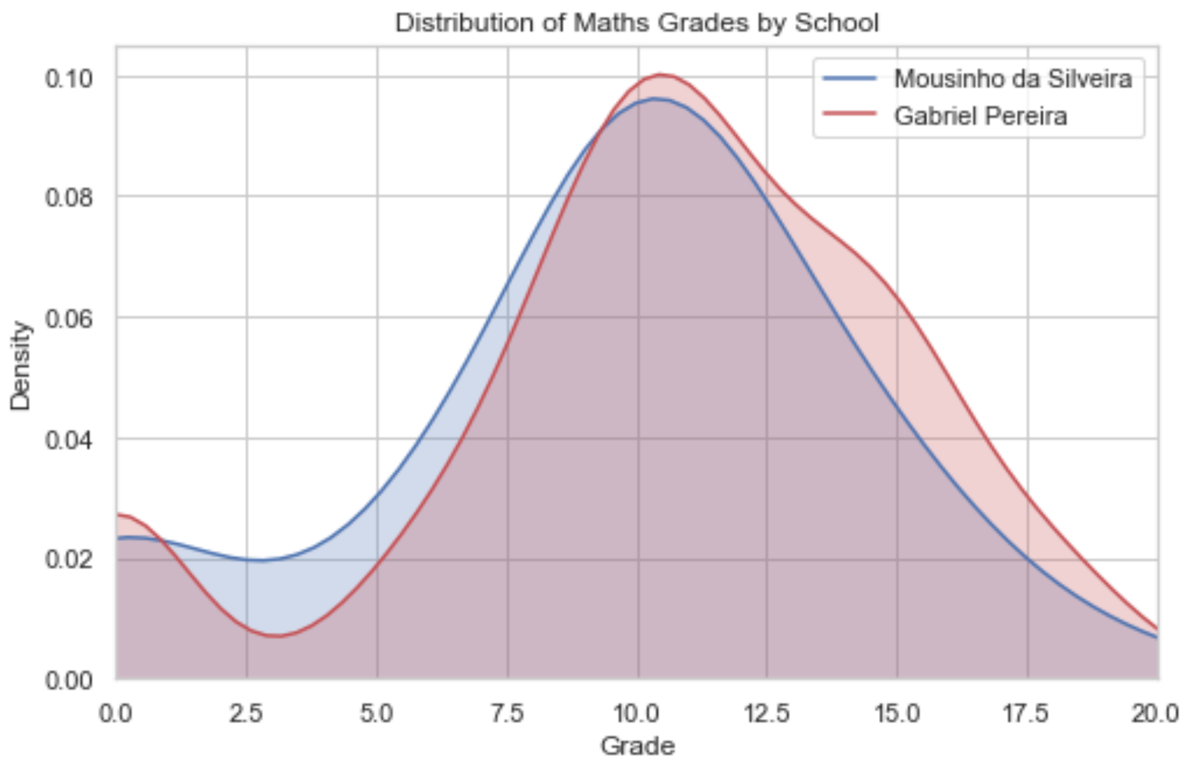

In [12]:

```
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['schoolsup_x']=='yes'),'G3_y'],color='b',label='Port
ugese with school support',shade=
            True)
sns.kdeplot(d3.loc[(d3['schoolsup_y']=='no'),'G3_y'],color='r',label='Portu
gese without school support',shade=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Portugese Grades by School Support')
plt.show()
```



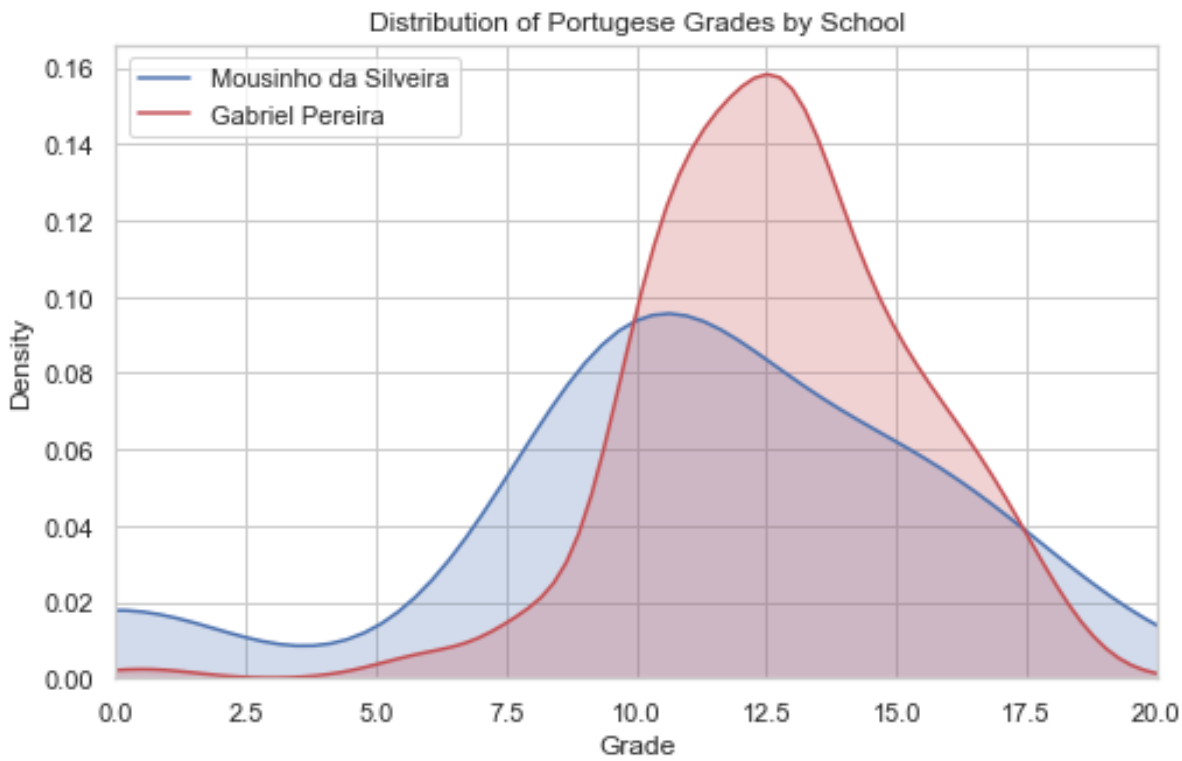

In [13]:

```
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['school']=='MS'),'G3_x'],color='b',label='Mousinho d
a Silveira',shade=
            True)
sns.kdeplot(d3.loc[(d3['school']=='GP'),'G3_x'],color='r',label='Gabriel Pe
reira',shade=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Maths Grades by School')
plt.show()
```




In [14]:

```
plt.figure(figsize=(8,5))
sns.kdeplot(d3.loc[(d3['school']=='MS'),'G3_y'],color='b',label='Mousinho d
a Silveira',shade=
            True)
sns.kdeplot(d3.loc[(d3['school']=='GP'),'G3_y'],color='r',label='Gabriel Pe
reira',shade=
            True)
plt.xlim(0,20) #X-axis limit set as grades can only be between 0-20
plt.xlabel('Grade')
plt.ylabel('Density')
plt.title('Distribution of Portugese Grades by School')
plt.show()
```




Now that we've produced a number of distributions for our features, let's decide on which of them correlate the most with grades. Unfortunately this is not possible with both sets of data, the dataset to be used from now will be the maths dataset.

In [78]:

```
d1.describe()
```

Out[78]:

| | age | Medu | Fedu | traveltime | studytime | failures | famrel | freetime | goout | Dalc | Walc | health | absences | G1 | G2 | G3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 | 395.000000 |
| **mean** | 16.696203 | 2.749367 | 2.521519 | 1.448101 | 2.035443 | 0.334177 | 3.944304 | 3.235443 | 3.108861 | 1.481013 | 2.291139 | 3.554430 | 5.708861 | 10.908861 | 10.713924 | 10.415190 |
| **std** | 1.276043 | 1.094735 | 1.088201 | 0.697505 | 0.839240 | 0.743651 | 0.896659 | 0.998862 | 1.113278 | 0.890741 | 1.287897 | 1.390303 | 8.003096 | 3.319195 | 3.761505 | 4.581443 |
| **min** | 15.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 3.000000 | 0.000000 | 0.000000 |
| **25%** | 16.000000 | 2.000000 | 2.000000 | 1.000000 | 1.000000 | 0.000000 | 4.000000 | 3.000000 | 2.000000 | 1.000000 | 1.000000 | 3.000000 | 0.000000 | 8.000000 | 9.000000 | 8.000000 |
| **50%** | 17.000000 | 3.000000 | 2.000000 | 1.000000 | 2.000000 | 0.000000 | 4.000000 | 3.000000 | 3.000000 | 1.000000 | 2.000000 | 4.000000 | 4.000000 | 11.000000 | 11.000000 | 11.000000 |
| **75%** | 18.000000 | 4.000000 | 3.000000 | 2.000000 | 2.000000 | 0.000000 | 5.000000 | 4.000000 | 4.000000 | 2.000000 | 3.000000 | 5.000000 | 8.000000 | 13.000000 | 13.000000 | 14.000000 |

| | age | Medu | Fedu | traveltime | studytime | failures | famrel | freetime | goout | Dalc | Walc | health | absences | G1 | G2 | G3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **max** | 22.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 3.000000 | 5.000000 | 5.000000 | 5.000000 | 5.000000 | 5.000000 | 5.000000 | 75.000000 | 19.000000 | 19.000000 | 20.000000 |

In [56]:

```
d1.corr()['G3'].sort_values()
```

Out[56]:

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

In [57]:

```
cat_d1=d1.select_dtypes(object)
dummy_d1=pd.get_dummies(cat_d1)
dummy_d1['G3']=d1['G3']
dummy_d1.corr()['G3'].sort_values()
```

Out[57]:

```
higher_no          -0.182465
romantic_yes       -0.129970
Mjob_at_home       -0.115634
address_R          -0.105756
sex_F              -0.103456
paid_no            -0.101996
reason_course      -0.098950
internet_no        -0.098483
Mjob_other         -0.096477
guardian_other     -0.087774
schoolsup_yes      -0.082788
famsize_GT3        -0.081407
Pstatus_T          -0.058009
Fjob_other         -0.053483
nursery_no         -0.051568
school_MS          -0.045017
```

```
famsup_yes           -0.039157
reason_home          -0.021359
Fjob_services        -0.016108
activities_no        -0.016100
Fjob_at_home         -0.013385
activities_yes        0.016100
guardian_mother       0.022338
guardian_father       0.032493
famsup_no             0.039157
school_GP             0.045017
nursery_yes           0.051568
reason_other          0.052008
Fjob_health           0.057111
Mjob_teacher          0.057712
Pstatus_A             0.058009
Mjob_services         0.078429
famsize_LE3           0.081407
schoolsup_no          0.082788
Fjob_teacher          0.095374
reason_reputation     0.095692
internet_yes          0.098483
paid_yes              0.101996
sex_M                 0.103456
address_U             0.105756
Mjob_health           0.116158
romantic_no           0.129970
higher_yes            0.182465
G3                    1.000000
Name: G3, dtype: float64
```

Up next, feature selection!

In [79]:

```python
# Function for the highest correlated variables
def most_correlated(d1):
    #Unnecessary or too highly correlted variables
    d1 = d1.drop(columns=['school', 'G1', 'G2'])

    # One-Hot Encoding of Categorical Variables
    d1 = pd.get_dummies(d1)

    # Find correlations with the Grade regardless of direction
    most_correlated = d1.corr().abs()['G3'].sort_values(ascending=False)

    #Pull the top 6 correlations
    most_correlated = most_correlated[:8]

    #Add the categorical variables and remove the binary opposite variable
    d1 = d1.loc[:, most_correlated.index]
    d1 = d1.drop(columns = ['higher_no'])
```

```
    return d1

d1=most_correlated(d1)
```

In [80]:

```
# Split into train/test, labels= target data
def train_test(d1):
    labels = d1['G3']
    X_train, X_test, y_train, y_test = train_test_split(d1, labels,
                                                          test_size = 0.25,
                                                          random_state=42)

    return X_train, X_test, y_train, y_test
train_test(d1)
```

Out[80]:

```
(     G3  failures  Medu  higher_yes  age  Fedu  goout
 16   14         0     4           1   16     4      3
 66   12         0     4           1   15     4      3
 211  13         0     4           1   17     4      5
 7     6         0     4           1   17     4      4
 19   10         0     4           1   16     3      3
 ..   ..       ...   ...         ...  ...   ...    ...
 71   10         0     4           1   15     2      3
 106   8         0     2           1   15     2      2
 270   9         2     3           1   19     3      5
 348  15         0     4           1   17     3      3
 102  14         0     4           1   15     4      3

 [296 rows x 7 columns],      G3  failures  Medu  higher_yes  age  Fedu  go
out
 78   10         3     2           0   17     1      1
 371  12         0     1           0   18     2      3
 248   5         1     3           1   18     3      3
 55   10         0     2           1   16     1      4
 390   9         2     2           1   20     2      4
 ..   ..       ...   ...         ...  ...   ...    ...
 367   0         1     1           1   17     1      1
 210   8         0     3           1   19     3      3
 75   10         0     4           1   15     3      3
 104  18         0     3           1   15     4      4
 374  19         0     4           1   18     4      4

 [99 rows x 7 columns], 16     14
 66     12
 211    13
 7       6
 19     10
```

```
        ..
71      10
106      8
270      9
348     15
102     14
Name: G3, Length: 296, dtype: int64, 78      10
371     12
248      5
55      10
390      9
        ..
367      0
210      8
75      10
104     18
374     19
Name: G3, Length: 99, dtype: int64)
```

In [60]:

```
X_train, X_test, y_train, y_test = train_test(d1)
X_train.head()
```

Out[60]:

| | G3 | failures | Medu | higher_yes | age | Fedu | goout |
|---|---|---|---|---|---|---|---|
| **16** | 14 | 0 | 4 | 1 | 16 | 4 | 3 |
| **66** | 12 | 0 | 4 | 1 | 15 | 4 | 3 |
| **211** | 13 | 0 | 4 | 1 | 17 | 4 | 5 |
| **7** | 6 | 0 | 4 | 1 | 17 | 4 | 4 |
| **19** | 10 | 0 | 4 | 1 | 16 | 3 | 3 |

In [61]:

```
# Rename variables in train and test
X_train = X_train.rename(columns={'higher_yes': 'higher_edu',
                                  'Medu': 'mother_edu',
                                  'Fedu': 'father_edu',
                                 'goout': 'go_out'})

X_test = X_test.rename(columns={'higher_yes': 'higher_edu',
                                  'Medu': 'mother_edu',
                                  'Fedu': 'father_edu',
                                 'goout': 'go_out'})
```

In [62]:

```
X_train.head()
```

Out[62]:

| | G3 | failures | mother_edu | higher_edu | age | father_edu | go_out |
|---|---|---|---|---|---|---|---|
| **16** | 14 | 0 | 4 | 1 | 16 | 4 | 3 |
| **66** | 12 | 0 | 4 | 1 | 15 | 4 | 3 |
| **211** | 13 | 0 | 4 | 1 | 17 | 4 | 5 |
| **7** | 6 | 0 | 4 | 1 | 17 | 4 | 4 |
| **19** | 10 | 0 | 4 | 1 | 16 | 3 | 3 |

In [29]:

```
print(X_train.shape)
print(X_test.shape)
(296, 7)
(99, 7)
```

In [82]:

```
#Correlation matrix showing the Pearson's correlation coefficient of all va
riable relationships
plt.figure(figsize=(11,11))
corrMatrix = X_train.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```


In [63]:

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

    return mae, rmse
```

In [64]:

```
def compare(X_train, X_test, y_train, y_test):
    # Models to test
    model_name = ['Linear Regression','Random Forest',
                  'Ridge']
    X_train = X_train.drop(columns='G3')
    X_test = X_test.drop(columns='G3')

    model1 = LinearRegression()
```

```
    model2 = RandomForestRegressor(n_estimators=50)
    model3 = Ridge(alpha=1.0)

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

    # Train and test models for comparison
    for i, model in enumerate([model1, model2, model3]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

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

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

    return results
```

In [65]:

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

In [66]:

```
# Formula for Bayesian Linear Regression
formula = 'G3 ~ ' + ' + '.join(['%s' % variable for variable in X_train.col
umns[1:]])
formula
```

Out[66]:

```
'G3 ~ failures + mother_edu + higher_edu + age + father_edu + go_out'
```

In [67]:

```
# Context for the model
with pm.Model() as normal_model:

    # The prior for the model parameters will be a normal distribution
    family = pm.glm.families.Normal()

    # Creating the model requires the BLR formula and data
    pm.GLM.from_formula(formula, data = X_train, family = family)

    # Perform Markov Chain Monte Carlo sampling
    n_trace = pm.sample(draws=2000, chains = 2, tune = 500, target_accept=0
.9)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [sd, go_out, father_edu, age, higher_edu, mother_edu, failures, Inter
cept]
```

```
Sampling 2 chains, 0 divergences: 100%|██████████| 5000/5000 [00:45<00:00, 108.94draws/s]
```

In [70]:

```
# Print out the mean variable weight from the trace
for variable in n_trace.varnames:
    print('Variable: {:15} Mean weight in model: {:.4f}'.format(variable, 
                                                                np.mean(n_t
race[variable])))
```

```
Variable: Intercept       Mean weight in model: 12.1396
Variable: failures        Mean weight in model: -1.7785
Variable: mother_edu      Mean weight in model: 0.7221
Variable: higher_edu      Mean weight in model: 2.1258
Variable: age             Mean weight in model: -0.1504
Variable: father_edu      Mean weight in model: -0.5326
Variable: go_out          Mean weight in model: -0.3983
Variable: sd_log__        Mean weight in model: 1.4261
Variable: sd              Mean weight in model: 4.1663
```

In [71]:

```
pm.summary(n_trace)
```

Out[71]:

| | mean | sd | hpd_3% | hpd_97% | mcse_mean | mcse_sd | ess_mean | ess_sd | ess_bulk | ess_tail | r_hat |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | 12.140 | 3.914 | 5.113 | 19.811 | 0.097 | 0.070 | 1632.0 | 1583.0 | 1642.0 | 2129.0 | 1.0 |
| **failures** | -1.779 | 0.372 | -2.495 | -1.106 | 0.007 | 0.005 | 2639.0 | 2622.0 | 2641.0 | 2676.0 | 1.0 |
| **mother_edu** | 0.722 | 0.293 | 0.198 | 1.310 | 0.006 | 0.004 | 2634.0 | 2634.0 | 2635.0 | 2521.0 | 1.0 |
| **higher_edu** | 2.126 | 1.112 | -0.028 | 4.176 | 0.021 | 0.015 | 2817.0 | 2602.0 | 2819.0 | 2128.0 | 1.0 |
| **age** | -0.150 | 0.206 | -0.530 | 0.236 | 0.005 | 0.004 | 1707.0 | 1707.0 | 1718.0 | 1947.0 | 1.0 |
| **father_edu** | -0.533 | 0.291 | -1.079 | 0.010 | 0.006 | 0.004 | 2714.0 | 2714.0 | 2726.0 | 2978.0 | 1.0 |
| **go_out** | -0.398 | 0.222 | -0.836 | 0.003 | 0.003 | 0.003 | 4076.0 | 3773.0 | 4066.0 | 2762.0 | 1.0 |

| | mean | sd | hpd_3% | hpd_97% | mcse_mean | mcse_sd | ess_mean | ess_sd | ess_bulk | ess_tail | r_hat |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **sd** | 4.166 | 0.176 | 3.854 | 4.516 | 0.003 | 0.002 | 3446.0 | 3429.0 | 3452.0 | 2799.0 | 1.0 |

In [86]:

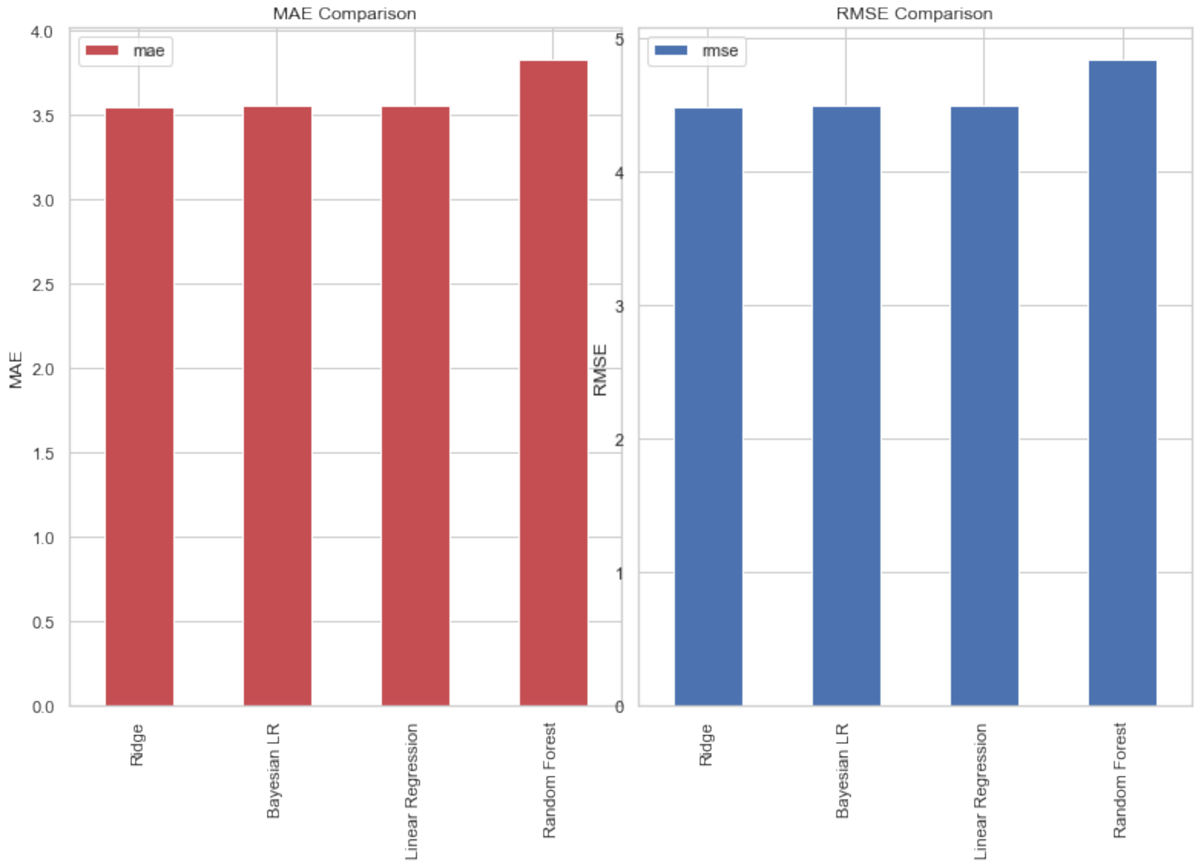
```
# Evalute the MCMC trace and compare to regression models
def evaluate_trace(trace, X_train, X_test, y_train, y_test, model_results):

    # Dictionary of all sampled values for each parameter
    var_dict = {}
    for variable in trace.varnames:
        var_dict[variable] = trace[variable]

    # Results into a dataframe
    var_weights = pd.DataFrame(var_dict)

    # Means for all the weights
    var_means = var_weights.mean(axis=0)

    # Create an intercept column
    X_test['Intercept'] = 1

    # Align names of the test observations and means
    names = X_test.columns[1:]
    X_test = X_test.loc[:, names]
    var_means = var_means[names]

    # Calculate estimate for each test observation using the average weight
s
    results = pd.DataFrame(index = X_test.index, columns = ['estimate'])

    for row in X_test.iterrows():
        results.loc[row[0], 'estimate'] = np.dot(np.array(var_means), np.ar
ray(row[1]))

    # Metrics
    actual = np.array(y_test)
    errors = results['estimate'] - actual
    mae = np.mean(abs(errors))
    rmse = np.sqrt(np.mean(errors ** 2))

    print('BLR  MAE: {:.4f}\nBLR RMSE: {:.4f}'.format(mae, rmse))

    # Add the results to the comparison dataframe
    model_results.loc['Bayesian LR', :] = [mae, rmse]
```

```
    plt.figure(figsize=(11, 8))

    # Plot median absolute percentage error of all models
    ax = plt.subplot(1, 2, 1)
    model_results.sort_values('mae', ascending = True).plot.bar(y = 'mae',
color = 'r', ax = ax)
    plt.title('MAE Comparison'); plt.ylabel('MAE');
    plt.tight_layout()

    # Plot root mean squared error of all models
    ax = plt.subplot(1, 2, 2)
    model_results.sort_values('rmse', ascending = True).plot.bar(y = 'rmse'
, color = 'b', ax = ax)
    plt.title('RMSE Comparison'); plt.ylabel('RMSE')

    return model_results
```

In [87]:

```
all_model_results = evaluate_trace(n_trace, X_train, X_test, y_train, y_tes
t, results)
```

```
BLR  MAE: 3.5556
BLR RMSE: 4.4923
```




In [75]:

```
all_model_results
```

Out[75]:

| | mae | rmse |
|---|---|---|
| **Linear Regression** | 3.55577 | 4.49314 |
| **Random Forest** | 3.8253 | 4.83371 |
| **Ridge** | 3.55137 | 4.48777 |
| **Bayesian LR** | 3.55563 | 4.49228 |

In [46]:

```
print('The Bayesian linear regression MAE score is {:0.2f}% higher than the OLS regression score.'.format(
    (100 * abs(results.loc['Bayesian LR', 'mae'] - results.loc['Linear Regression', 'mae'])) / results.loc['Linear Regression', 'mae']))

print('The Bayesian linear regression RMSE score is {:0.2f}% lower than the OLS regression score.'.format(
    (100 * abs(results.loc['Bayesian LR', 'rmse'] - results.loc['Linear Regression', 'rmse'])) / results.loc['Linear Regression', 'rmse']))
```

```
The Bayesian linear regression MAE score is 0.00% higher than the OLS regression score.
The Bayesian linear regression RMSE score is 0.01% lower than the OLS regression score.
```

In [48]:

```
X_test.head()
```

Out[48]:

| | G3 | failures | mother_edu | higher_edu | age | father_edu | go_out | Intercept |
|---|---|---|---|---|---|---|---|---|
| **78** | 10 | 3 | 2 | 0 | 17 | 1 | 1 | 1 |
| **371** | 12 | 0 | 1 | 0 | 18 | 2 | 3 | 1 |
| **248** | 5 | 1 | 3 | 1 | 18 | 3 | 3 | 1 |
| **55** | 10 | 0 | 2 | 1 | 16 | 1 | 4 | 1 |
| **390** | 9 | 2 | 2 | 1 | 20 | 2 | 4 | 1 |

In [49]:

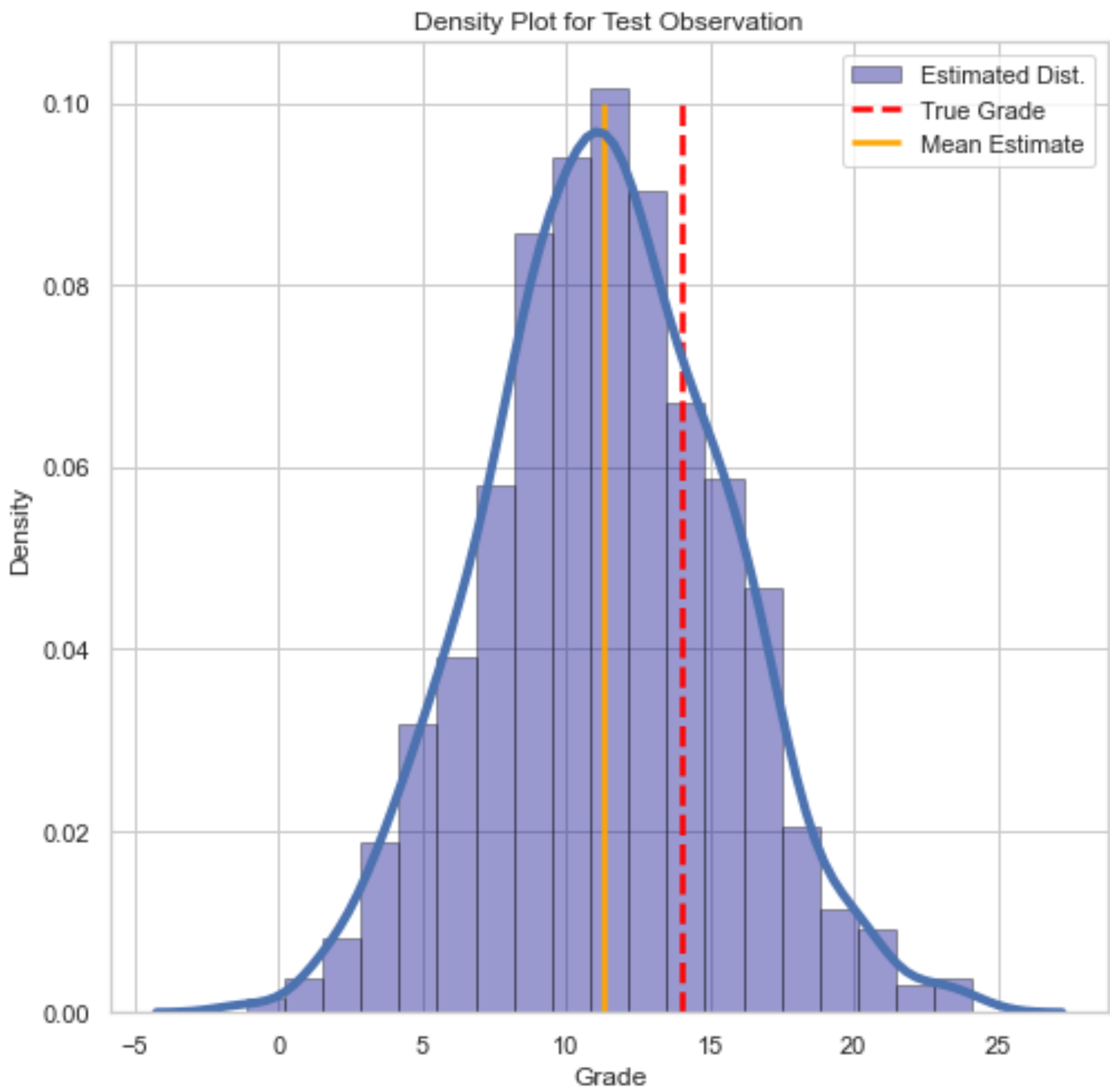
```
# Make a new prediction from the test set and compare to actual value
def test_model(trace, test_observation):
    from statistics import mean
```

```
    # Print out the test observation data
    print('Test Observation:')
    print(test_observation)
    var_dict = {}
    for variable in trace.varnames:
        var_dict[variable] = trace[variable]

    # Results into a dataframe
    var_weights = pd.DataFrame(var_dict)

    # Standard deviation of the likelihood
    sd_value = var_weights['sd'].mean()

    # Actual Value
    actual = test_observation['G3']

    # Add in intercept term
    test_observation['Intercept'] = 1
    test_observation = test_observation.drop('G3')

    # Align weights and test observation
    var_weights = var_weights[test_observation.index]

    # Means for all the weights
    var_means = var_weights.mean(axis=0)


    # Location of mean for observation
    mean_loc = np.dot(var_means, test_observation)

    # Estimates of grade
    estimates = np.random.normal(loc = mean_loc, scale = sd_value, size=100
0)

    # Plot all the estimates
    plt.figure(figsize=(8, 8))
    sns.distplot(estimates, hist = True, kde = True, bins = 19,
                 hist_kws = {'edgecolor': 'k', 'color': 'darkblue'},
                kde_kws = {'linewidth' : 4},
                label = 'Estimated Dist.')
    # Plot the actual grade
    plt.vlines(x = actual, ymin = 0, ymax = 0.1,
               linestyles = '--', colors = 'red',
               label = 'True Grade',
              linewidth = 2.5)
    # Plot the mean estimate
    plt.vlines(x = mean_loc, ymin = 0, ymax = 0.1,
```

```
               linestyles = '-', colors = 'orange',
               label = 'Mean Estimate',
              linewidth = 2.5)

    plt.legend(loc = 1)
    plt.title('Density Plot for Test Observation');
    plt.xlabel('Grade'); plt.ylabel('Density');

    # Prediction information
    print('True Grade = %d' % actual)
    print('Average Estimate = %0.4f' % mean_loc)
    print('5%% Estimate = %0.4f    95%% Estimate = %0.4f' % (np.percentile(
estimates, 5),
                                       np.percentile(estimates, 95)))
```

In [50]:

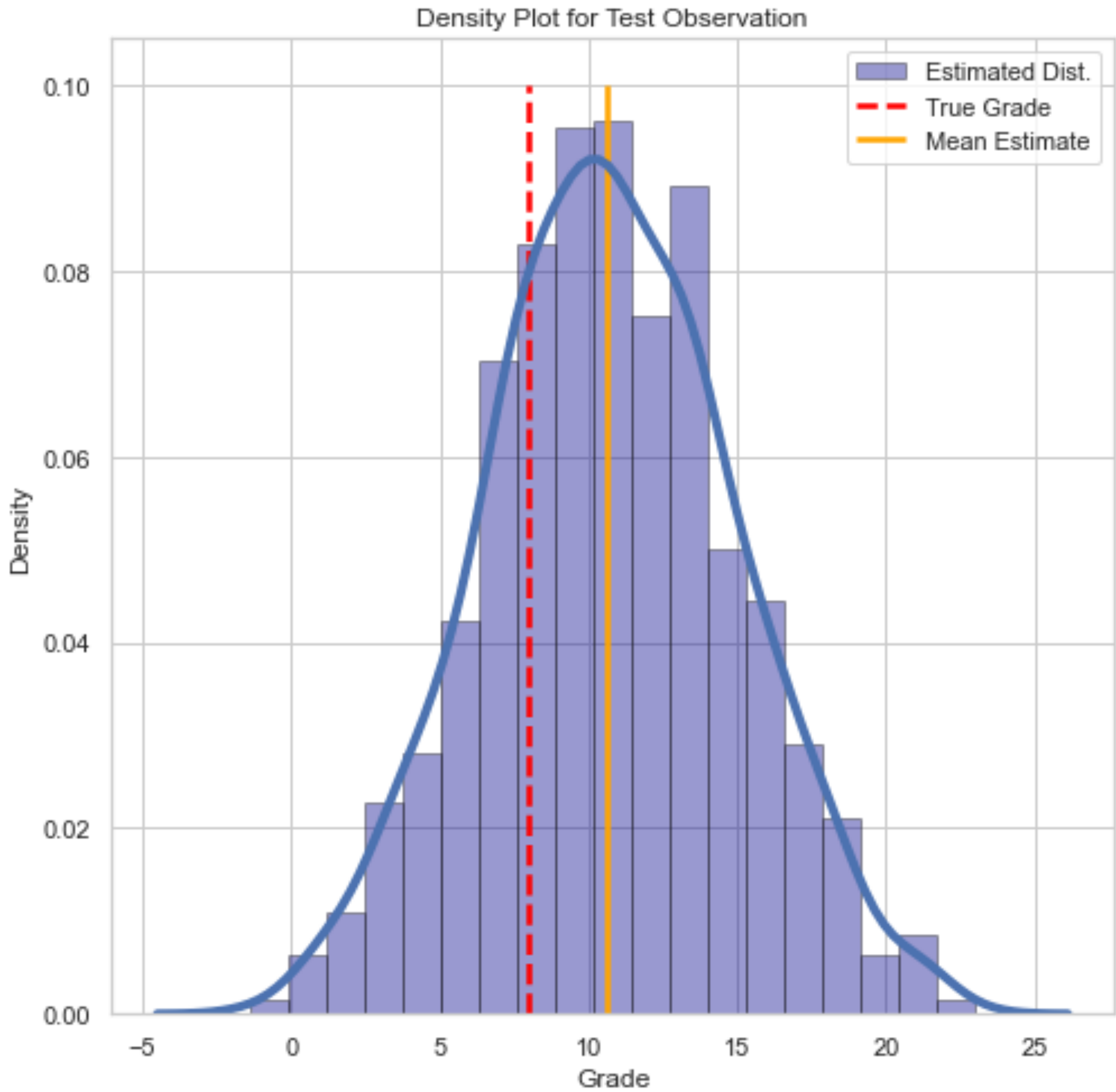
```
test_model(n_trace, X_test.iloc[42])
Test Observation:
G3              14
failures         0
mother_edu       4
higher_edu       1
age             17
father_edu       4
go_out           3
Intercept        1
Name: 329, dtype: int64
True Grade = 14
Average Estimate = 11.2583
5% Estimate = 4.5526    95% Estimate = 17.9737
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packag
es/ipykernel_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```




In [52]:

```
test_model(n_trace, X_test.iloc[16])
Test Observation:
G3             8
failures       0
mother_edu     2
higher_edu     1
age           16
father_edu     2
go_out         4
Intercept      1
Name: 124, dtype: int64
True Grade = 8
Average Estimate = 10.6413
5% Estimate = 3.6658    95% Estimate = 17.7550
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packag
es/ipykernel_launcher.py:22: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```




**Appendix II**:
I believe I should be marked an 80 out of 100 because I provided answers for all 10 tasks, with accompanying visualisations, explanations and clear logical steps from one task to the next. Some of the answers provided may not include the necessary amount of details, however I believe that the work I have provided has a sufficient level of detail and demonstration of knowledge to warrant an 80/100 mark.