Question 1

(a) Load the male_stud.csv dataset into Python as a pandas DataFrame

In [1]:
```
import pandas as pd

male_df = pd.read_csv("C:/Users/chels/Downloads/male_stud.csv")

#Python was able to successfully load the male_stud.csv dataset as a pandas DataFrame.
```

(b) Inspect the data

In [2]:
```
# Display the first few rows of the DataFrame
print(male_df.head())

# Get information about the dataset
print(male_df.info())

#Check if any missing values in the dataset
Missing_Values = male_df.isnull().any().any()
print("\n Check whether the data set contains any missing values?", Missing_Values)
```

```
   large_family  lives_in_city  traveltime  studytime  failures  paid  \
0             0              1           1          2         0     1
1             0              1           1          2         0     0
2             0              1           1          2         0     1
3             1              1           1          2         0     1
4             0              1           1          1         0     1

   activities  internet  romantic  famrel  freetime  goout  absences  \
0           1         1         0       5         4      2        10
1           0         1         0       4         4      4         0
2           0         1         0       4         2      2         0
3           1         1         0       5         5      1         0
4           1         1         0       4         3      3         2

   final_grade
0           15
1           11
2           19
3           15
4           14
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187 entries, 0 to 186
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   large_family   187 non-null    int64
 1   lives_in_city  187 non-null    int64
 2   traveltime     187 non-null    int64
 3   studytime      187 non-null    int64
 4   failures       187 non-null    int64
 5   paid           187 non-null    int64
 6   activities     187 non-null    int64
 7   internet       187 non-null    int64
 8   romantic       187 non-null    int64
 9   famrel         187 non-null    int64
 10  freetime       187 non-null    int64
 11  goout          187 non-null    int64
 12  absences       187 non-null    int64
 13  final_grade    187 non-null    int64
dtypes: int64(14)
memory usage: 20.6 KB
None

 Check whether the data set contains any missing values? False
```

In the dataset, there are 14 indicators and 187 students. Among the indicators are binary variables such as living in a city, having a large family, being paid, having activities, having internet access, and being romantic. There are also numerical factors such as study time, travel time, failures, family, free time, going out, absences, and final grade. There are no missing values in this dataset.

(c) Perform exploratory data analysis

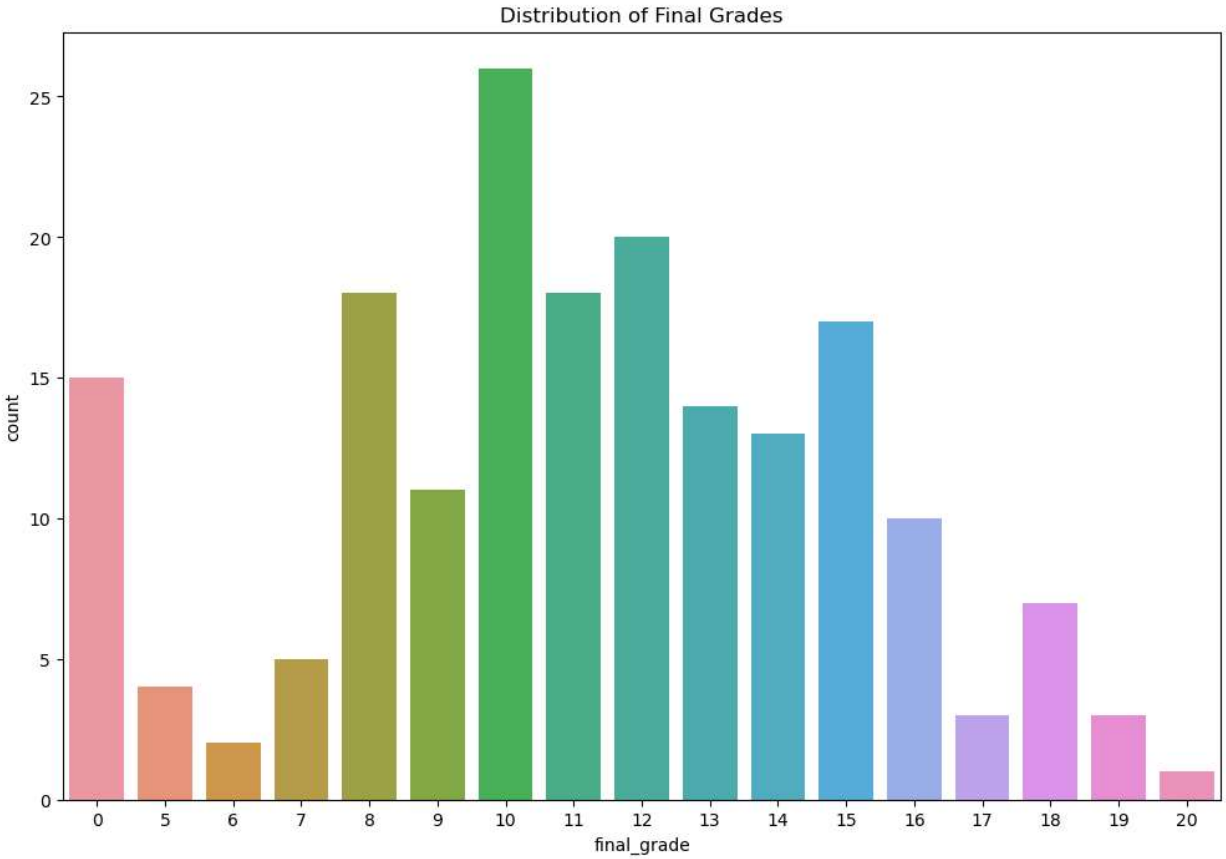
In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Summary statistics
print(male_df.describe())
```

```
       large_family  lives_in_city  traveltime   studytime    failures  \
count    187.000000     187.000000  187.000000  187.000000  187.000000
mean       0.668449       0.764706    1.491979    1.764706    0.368984
std        0.472034       0.425321    0.750405    0.808713    0.788152
min        0.000000       0.000000    1.000000    1.000000    0.000000
25%        0.000000       1.000000    1.000000    1.000000    0.000000
50%        1.000000       1.000000    1.000000    2.000000    0.000000
75%        1.000000       1.000000    2.000000    2.000000    0.000000
max        1.000000       1.000000    4.000000    4.000000    3.000000

              paid  activities    internet    romantic      famrel    freetime  \
count  187.000000  187.000000  187.000000  187.000000  187.000000  187.000000
mean     0.390374    0.561497    0.850267    0.283422    4.000000    3.486631
std      0.489144    0.497536    0.357767    0.451870    0.898027    1.001924
min      0.000000    0.000000    0.000000    0.000000    1.000000    1.000000
25%      0.000000    0.000000    1.000000    0.000000    4.000000    3.000000
50%      0.000000    1.000000    1.000000    0.000000    4.000000    3.000000
75%      1.000000    1.000000    1.000000    1.000000    5.000000    4.000000
max      1.000000    1.000000    1.000000    1.000000    5.000000    5.000000

             goout    absences  final_grade
count  187.000000  187.000000   187.000000
mean     3.197861    5.144385    10.914439
std      1.135164    5.980749     4.495297
min      1.000000    0.000000     0.000000
25%      2.000000    0.000000     9.000000
50%      3.000000    4.000000    11.000000
75%      4.000000    8.000000    14.000000
max      5.000000   38.000000    20.000000
```

Numerical summaries revealed the indicators' means and standard deviations. Based on the data, the majority of students commute to school for a short distance, taking an average of 1.49 hours. The average final grade is approximately 10.91, with a 4.50 standard deviation.

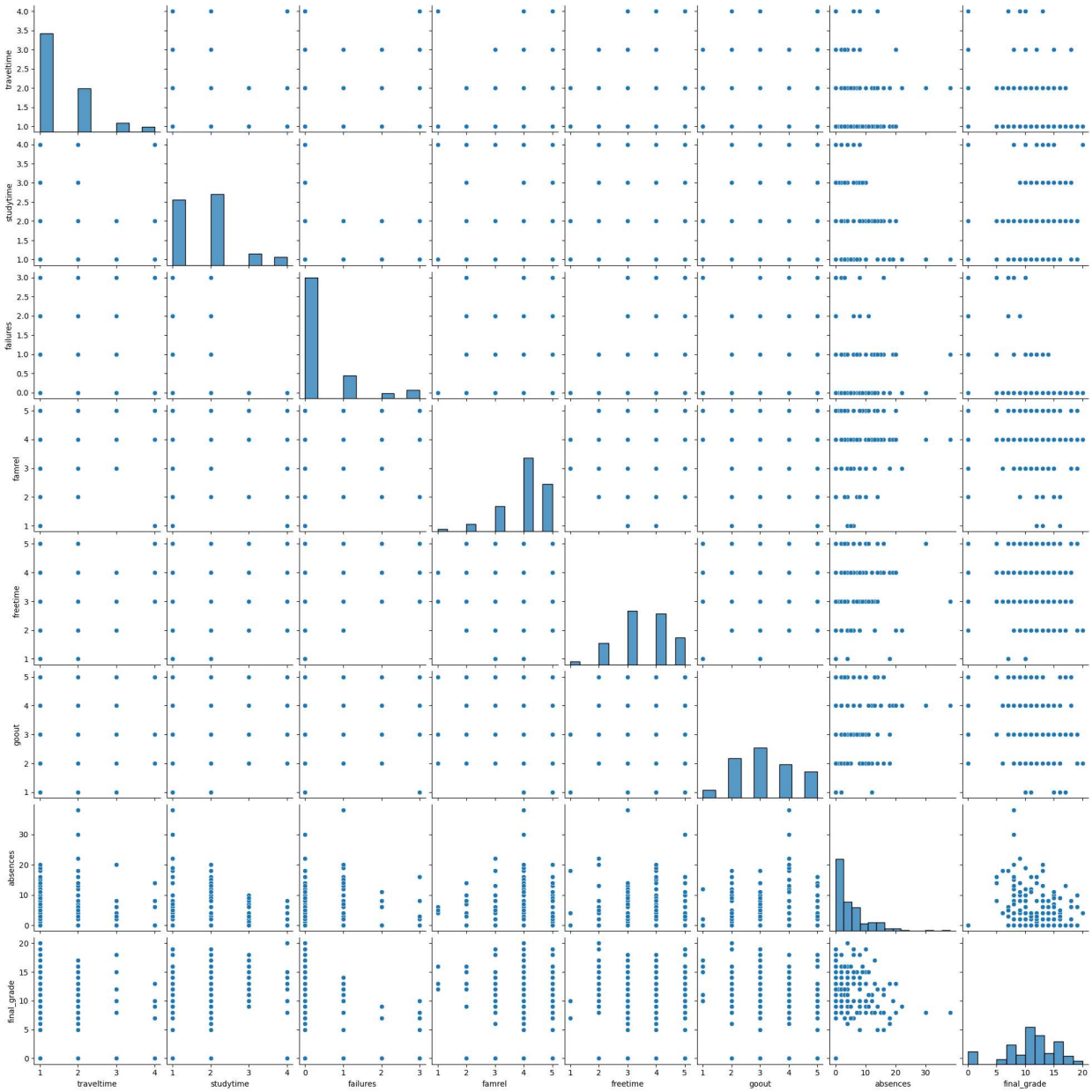
Countplot for Categorical Variables: To provide insight into the number of students achieving each grade, a countplot was created for the final grade distribution. With a concentration in the grades 10 and 11, the countplot showed a varied distribution

In [4]:
```
# Countplot for categorical variables
plt.figure(figsize=(12, 8))
sns.countplot(x='final_grade', data=male_df)
plt.title('Distribution of Final Grades')
plt.show()
```



Pairplot for Numerical Variables The relationships between numerical variables were visualized using a pairplot. In order to better understand the relationships between variables, the pairplot helps to identify potential correlations or patterns in the data.

In [5]:
```
# Pairplot for numerical variables
sns.pairplot(male_df, vars=['traveltime', 'studytime', 'failures', 'famrel', 'freetime
plt.show()
```



Question 2

(a) Load the female_stud.csv dataset into Python as a pandas DataFrame

In [6]:
```
female_df = pd.read_csv("C:/Users/chels/Downloads/female_stud.csv")
```

It has been successful to import the female_stud.csv dataset into Python as a pandas DataFrame

(b) Inspect the data

In [7]:
```
# Display the first few rows of the DataFrame
print(female_df.head())

# Get information about the datas et
print(female_df.info())

#Check if any missing values in the dataset
Missing_Values = female_df.isnull().any().any()
print("\n Check whether the data set contains any missing values?", Missing_Values)
```

```
   large_family  lives_in_city  traveltime  studytime  failures  paid  \
0             1              1           2          2         0     0
1             1              1           1          2         0     0
2             0              1           1          2         3     1
3             1              1           1          3         0     1
4             1              1           1          2         0     1

   activities  internet  romantic  famrel  freetime  goout  absences  \
0           0         0         0       4         3      4         6
1           0         1         0       5         3      3         4
2           0         1         0       4         3      2        10
3           1         1         1       3         2      2         2
4           0         0         0       4         3      2         4

   final_grade
0            6
1            6
2           10
3           15
4           10
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208 entries, 0 to 207
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   large_family   208 non-null    int64
 1   lives_in_city  208 non-null    int64
 2   traveltime     208 non-null    int64
 3   studytime      208 non-null    int64
 4   failures       208 non-null    int64
 5   paid           208 non-null    int64
 6   activities     208 non-null    int64
 7   internet       208 non-null    int64
 8   romantic       208 non-null    int64
 9   famrel         208 non-null    int64
 10  freetime       208 non-null    int64
 11  goout          208 non-null    int64
 12  absences       208 non-null    int64
 13  final_grade    208 non-null    int64
dtypes: int64(14)
memory usage: 22.9 KB
None

 Check whether the data set contains any missing values? False
```

The dataset, which includes 208 female students, includes 14 indicators: large family, lives in a city, travel time, study time, activities, paid, internet, romantic, family, free time, going out, absences, and final. This dataset has no missing values, just like the male group did

(c) Perform a t-test for each measurement to test whether any of the indicators differ between the male and female groups

In [8]:
```
from scipy.stats import ttest_ind

# Assuming 'final_grade' is the target variable
alpha = 0.01
```

```
# Loop through each column (excluding 'final_grade') and perform t-test
for column in female_df.columns[:-1]:
    t_stat, p_value = ttest_ind(male_df[column], female_df[column])

    print(f"\nT-test for '{column}':")
    print(f"T-score: {t_stat}")
    print(f"P-value: {p_value}")

    # Check if the p-value is less than the significance level
    if p_value < alpha:
        print("Reject the null hypothesis: There is a significant difference between m
    else:
        print("Fail to reject the null hypothesis: No significant difference between m
```

```
T-test for 'large_family':
T-score: -1.7886773632900528
P-value: 0.07443689700483698
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'lives_in_city':
T-score: -0.5653036030213555
P-value: 0.5721898495135425
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'traveltime':
T-score: 1.1860551557953753
P-value: 0.2363170824741324
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'studytime':
T-score: -6.378011403261443
P-value: 5.045044371289513e-10
Reject the null hypothesis: There is a significant difference between male and female
groups.

T-test for 'failures':
T-score: 0.8817780076175071
P-value: 0.37843584788424967
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'paid':
T-score: -2.58142674087509
P-value: 0.010200689886658792
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'activities':
T-score: 1.989052552579219
P-value: 0.0473883165761214
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'internet':
T-score: 0.8753563404920145
P-value: 0.38191471671030264
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'romantic':
T-score: -2.0331361835287765
P-value: 0.04271031909944724
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'famrel':
T-score: 1.1710913475656488
P-value: 0.24227162046982076
Fail to reject the null hypothesis: No significant difference between male and female
groups.
```

```
T-test for 'freetime':
T-score: 4.873860224579706
P-value: 1.59070521252592e-06
Reject the null hypothesis: There is a significant difference between male and female
groups.

T-test for 'goout':
T-score: 1.5089596455705667
P-value: 0.13211259546899726
Fail to reject the null hypothesis: No significant difference between male and female
groups.

T-test for 'absences':
T-score: -1.3304512483459712
P-value: 0.1841411182775425
Fail to reject the null hypothesis: No significant difference between male and female
groups.
```

Question 3

(a) Combine the two datasets into a single DataFrame

In [9]:
```
# Combining the two datasets
full_df = pd.concat([male_df, female_df], ignore_index=True)
print(full_df)
```

```
     large_family  lives_in_city  traveltime  studytime  failures  paid  \
0               0              1           1          2         0     1
1               0              1           1          2         0     0
2               0              1           1          2         0     1
3               1              1           1          2         0     1
4               0              1           1          1         0     1
..            ...            ...         ...        ...       ...   ...
390             1              0           2          3         0     1
391             1              0           3          1         0     1
392             1              0           1          3         1     0
393             0              1           1          2         0     1
394             1              1           2          2         1     0

     activities  internet  romantic  famrel  freetime  goout  absences  \
0             1         1         0       5         4      2        10
1             0         1         0       4         4      4         0
2             0         1         0       4         2      2         0
3             1         1         0       5         5      1         0
4             1         1         0       4         3      3         2
..          ...       ...       ...     ...       ...    ...       ...
390           0         0         0       5         3      3         2
391           1         1         1       4         4      3         7
392           1         1         0       5         4      2         0
393           0         1         0       4         3      4         0
394           1         0         0       1         1      1         0

     final_grade
0             15
1             11
2             19
3             15
4             14
..           ...
390           10
391            6
392            0
393            8
394            0

[395 rows x 14 columns]
```

After successfully merging the male_stud.csv and female_stud.csv datasets into a single DataFrame, a thorough analysis of the data pertaining to both male and female students is now possible.

(b) Compute the Pearson correlation coefficient between each of the measurements

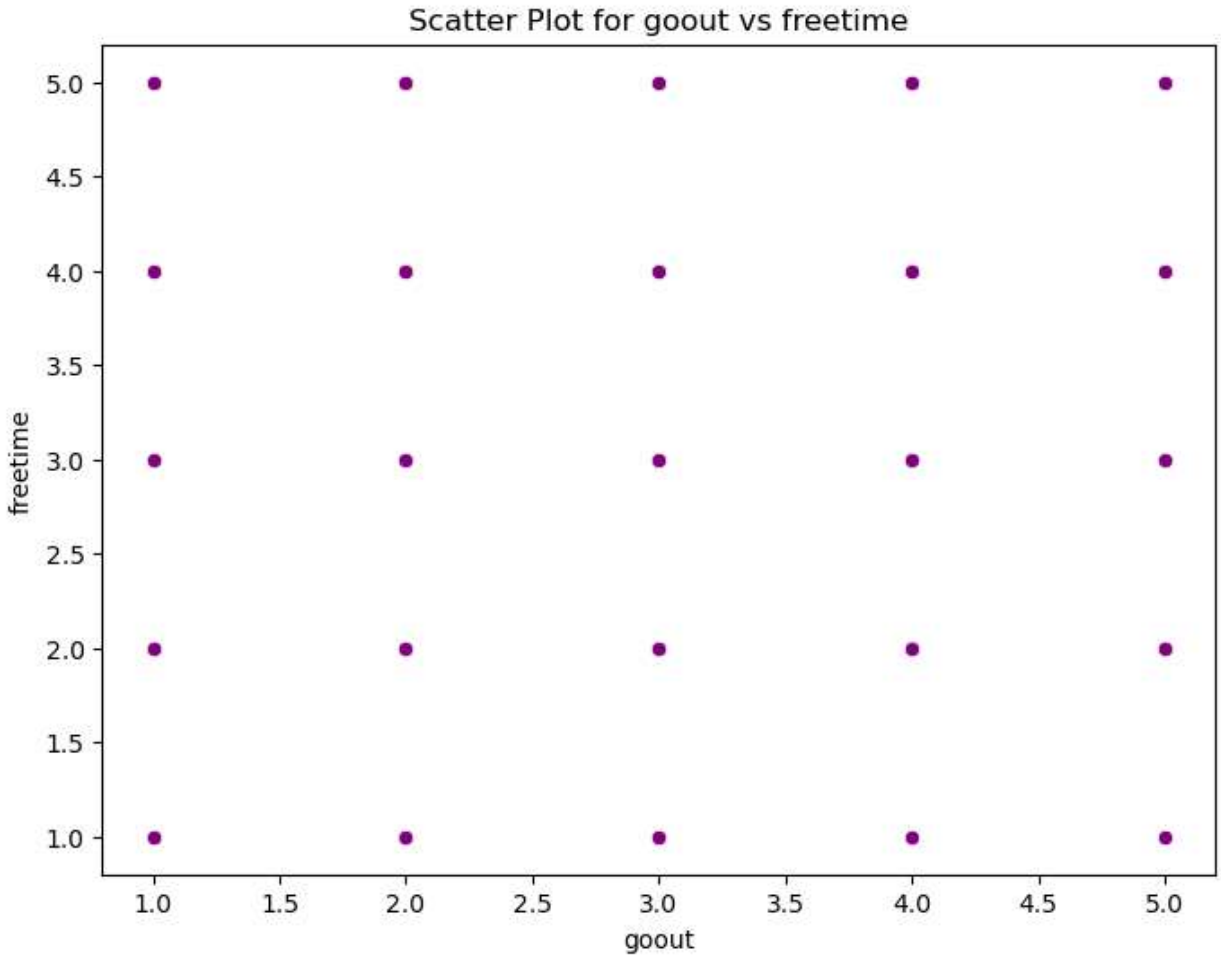
In [10]:
```
# Compute the Pearson correlation coefficient
correlation_matrix = full_df.corr()

# Identify the most correlated pairs
most_correlated_pairs = correlation_matrix.unstack().sort_values(ascending=False).drop

# List the four most strongly correlated pairs
top_correlations = most_correlated_pairs[1:5]
print(top_correlations)
```

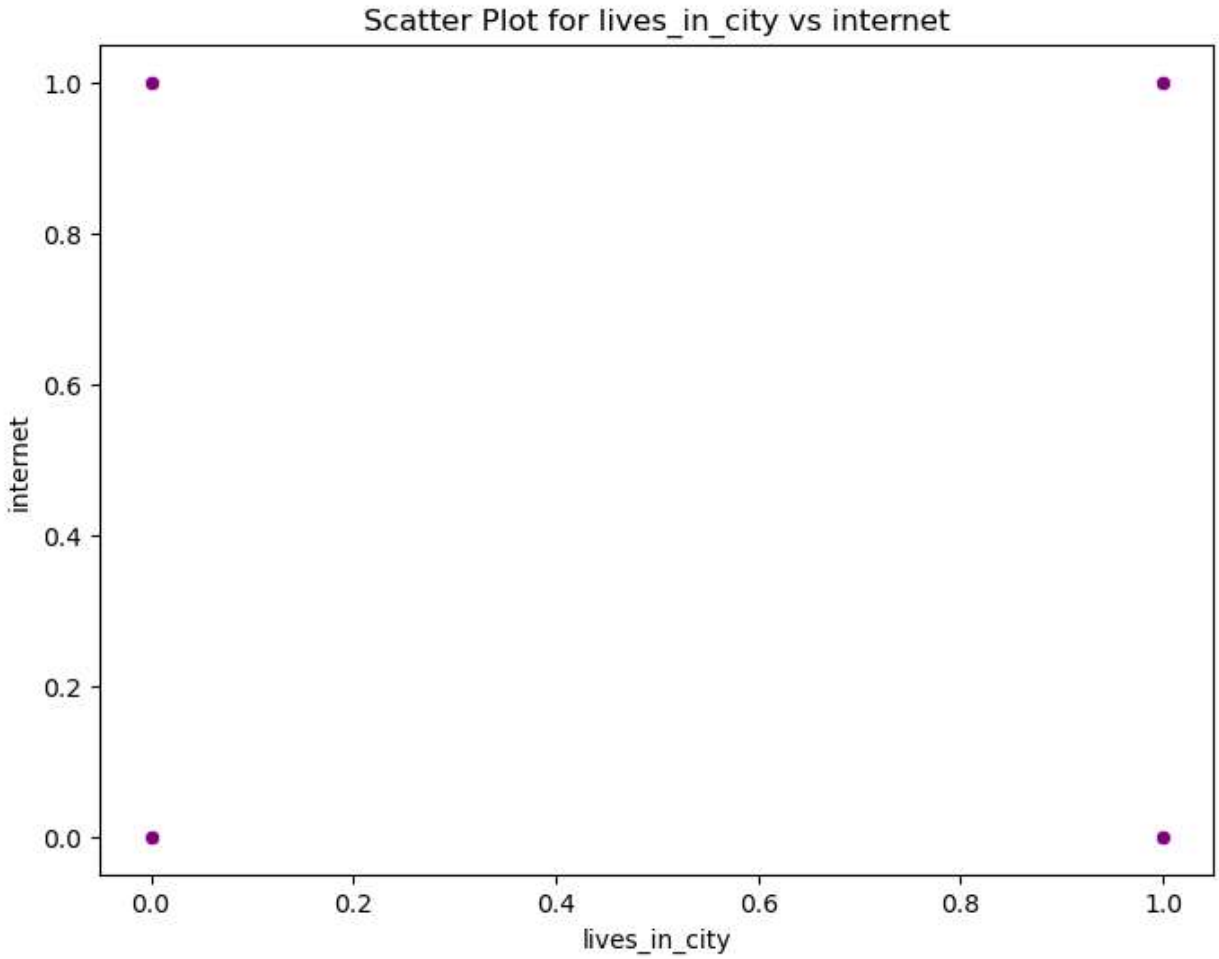
```
goout          freetime    0.285019
lives_in_city  internet    0.216842
studytime      paid        0.167220
romantic       absences    0.153384
dtype: float64
```

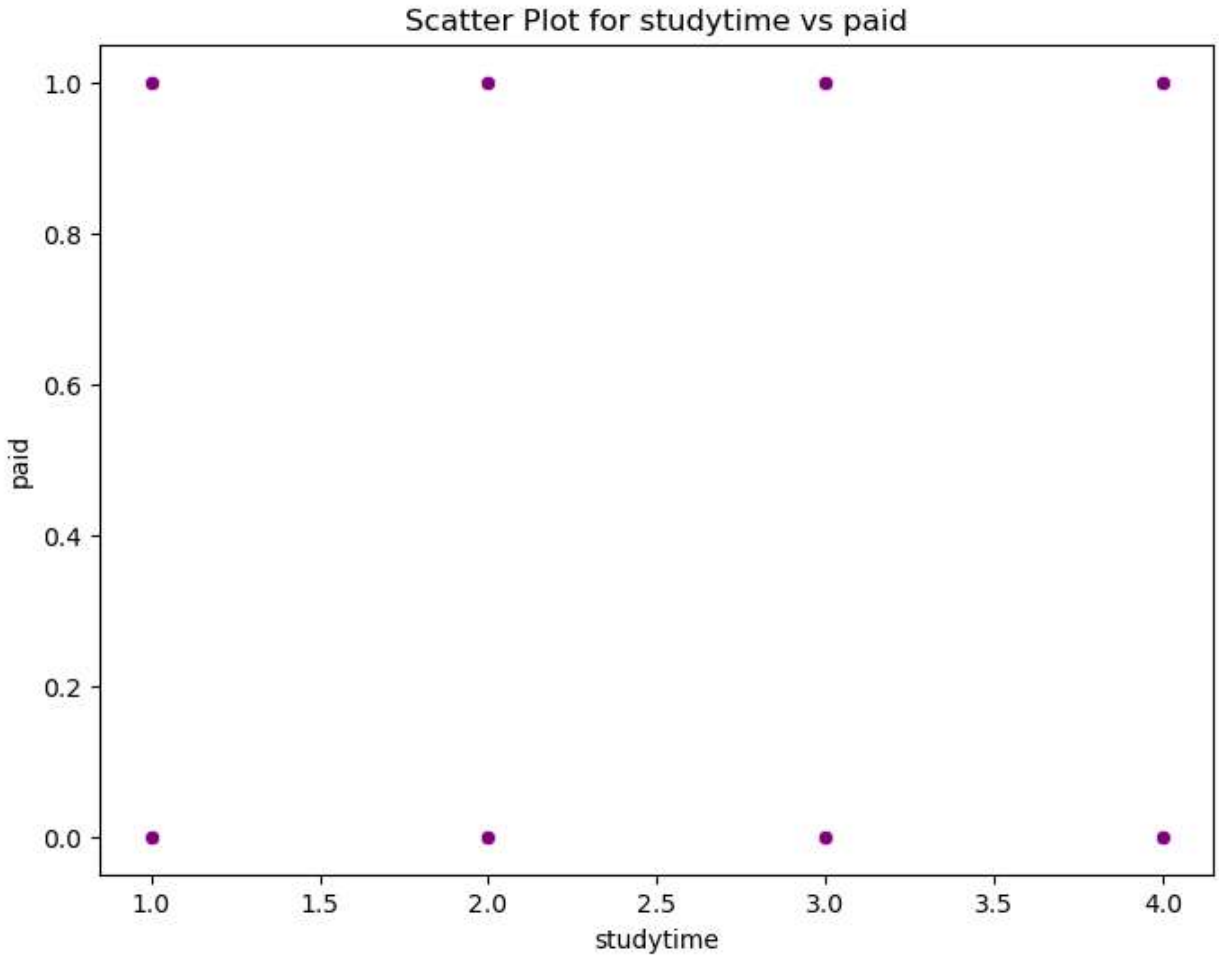
To determine the direction and strength of linear relationships between variables, the Pearson correlation coefficients were calculated for every pair of measurements. These are the four pairs with the strongest correlations: • goout and freetime (0.285): There is a positive correlation between the amount of time spent going out with friends (goout) and the perceived amount of free time after school (freetime). • lives_in_city and internet (0.217): Positive correlation: It is more likely that students who live in cities (lives_in_city) have access to the internet at home (internet). • studytime and paid (0.167): Positive correlation: Students are more likely to enroll in additional paid classes related to the course subject if they devote more time each week to studying (studytime). • romantic and absences (0.153): Positive correlation: Students in romantic relationships may miss more school than other students.

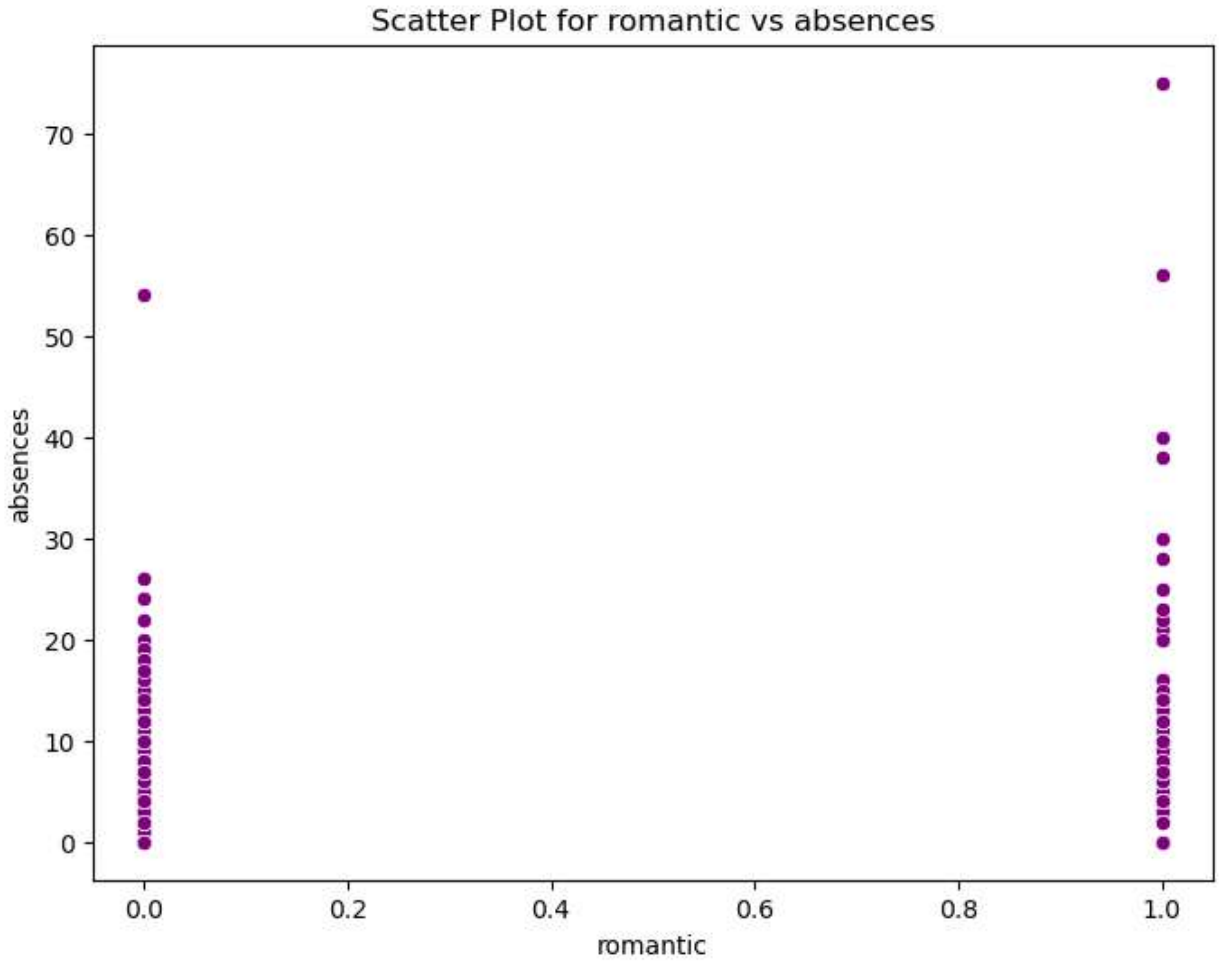
(c) Create scatter plots for each of the correlated pairs

In [11]:
```
# Import necessary libraries for plotting
import seaborn as sns
import matplotlib.pyplot as plt

# Create scatter plots for each correlated pair
for pair in top_correlations.index:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=full_df, x=pair[0], y=pair[1], color='purple')
    plt.title(f'Scatter Plot for {pair[0]} vs {pair[1]}')
    plt.xlabel(pair[0])
    plt.ylabel(pair[1])
    plt.show()
```


Scatter Plot for goout vs freetime
freetime
goout
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Scatter Plot for lives_in_city vs internet
internet
lives_in_city
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot for studytime vs paid
paid
studytime
1.0
0.8
0.6
0.4
0.2
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0



Scatter Plots for Correlated Pairs To examine the relationships between the correlated pairs mentioned earlier, scatter plots were made: • goout vs. freetime: A generally dispersed distribution of points can be seen in the scatter plots for goout and freetime. Although there is a correlation, it is not very strong. • lives_in_city vs. internet: The scatter plot for internet and lives_in_city shows a propensity for points to group together in the upper right quadrant, confirming the positive correlation. • studytime vs. paid: A positive correlation is evident in the studytime and paid scatter plot, with points dispersed throughout the graph, indicating a moderate association. • romantic vs. absences: A positive correlation, albeit not a very strong one, can be seen in the scatter plot for romantic and absences.

Question 4

(a) Create a new column indicating whether the student passed or failed

In [12]:
```
# Create a new column 'pass_fail' based on the final grade
full_df['pass_fail'] = full_df['final_grade'] >= 10
full_df['pass_fail'] = full_df['pass_fail'].astype(int)
print(full_df)
```

```
     large_family  lives_in_city  traveltime  studytime  failures  paid  \
0               0              1           1          2         0     1
1               0              1           1          2         0     0
2               0              1           1          2         0     1
3               1              1           1          2         0     1
4               0              1           1          1         0     1
..            ...            ...         ...        ...       ...   ...
390             1              0           2          3         0     1
391             1              0           3          1         0     1
392             1              0           1          3         1     0
393             0              1           1          2         0     1
394             1              1           2          2         1     0

     activities  internet  romantic  famrel  freetime  goout  absences  \
0             1         1         0       5         4      2        10
1             0         1         0       4         4      4         0
2             0         1         0       4         2      2         0
3             1         1         0       5         5      1         0
4             1         1         0       4         3      3         2
..          ...       ...       ...     ...       ...    ...       ...
390           0         0         0       5         3      3         2
391           1         1         1       4         4      3         7
392           1         1         0       5         4      2         0
393           0         1         0       4         3      4         0
394           1         0         0       1         1      1         0

     final_grade  pass_fail
0             15          1
1             11          1
2             19          1
3             15          1
4             14          1
..           ...        ...
390           10          1
391            6          0
392            0          0
393            8          0
394            0          0

[395 rows x 15 columns]
```

Utilizing the Portuguese grading system, which assigns a fail (0) to any grade between 0 and 9.5 and a pass (1) to any grade between 10 and 20, a new column called "pass_fail" has been successfully created.

(b) Separate the data into response and predictor variables and standardize the predictor variables

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Separate data into response and predictor variables
X = full_df.drop(['final_grade', 'pass_fail'], axis=1)
y = full_df['pass_fail']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

```
# Standardize the predictor variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Response ('pass_fail') and predictor variables have been separated out of the data. To guarantee that every variable contributes equally to the logistic regression model, the predictor variables have been standardized

(c) Fit a logistic regression model and interpret the fitted model

In [14]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Fit logistic regression model
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_scaled, y_train)

# Predictions on the test set
y_pred = logreg.predict(X_test_scaled)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.35      0.49        26
           1       0.75      0.96      0.84        53

    accuracy                           0.76        79
   macro avg       0.78      0.65      0.66        79
weighted avg       0.77      0.76      0.73        79

Accuracy: 0.759493670886076
```

The classification report offers a thorough overview of the logistic regression model's performance after it has been fitted: • Precision (0): 82% of predicted fails are true fails. • Recall (0): Only 35% of actual fails are correctly predicted. • F1-score (0): Recall for fails is 49%. • Precision (1): 75% of predicted passes are true passes. • Recall (1): A high recall of 96% indicates that the model correctly identifies most actual passes. • F1-score (1): Recall for passes is 84%. • Overall Accuracy: The model achieves an accuracy of 76%. The model can be improved significantly, especially in correctly identifying class 0 (failing cases).

(d) Perform forward selection for your regression model using the Akaike Information Criterion (AIC)

In [15]:
```
import statsmodels.api as sm

# Add a constant term to the predictor variables
X_train_scaled_const = sm.add_constant(X_train_scaled)

# Fit logistic regression model with AIC-based forward selection
```

```
model = sm.GLM(y_train, X_train_scaled_const, family=sm.families.Binomial()).fit()

# Display the summary of the model
print(model.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              pass_fail   No. Observations:                  316
Model:                            GLM   Df Residuals:                      302
Model Family:                Binomial   Df Model:                           13
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -177.00
Date:                Sun, 10 Dec 2023   Deviance:                       354.00
Time:                        11:21:24   Pearson chi2:                     323.
No. Iterations:                     4   Pseudo R-squ. (CS):             0.1366
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7806      0.132      5.930      0.000       0.523       1.039
x1            -0.2192      0.138     -1.593      0.111      -0.489       0.050
x2             0.0475      0.141      0.336      0.737      -0.229       0.324
x3             0.0140      0.139      0.101      0.920      -0.258       0.286
x4            -0.0402      0.139     -0.290      0.772      -0.312       0.232
x5            -0.6796      0.145     -4.678      0.000      -0.964      -0.395
x6             0.1691      0.137      1.238      0.216      -0.099       0.437
x7             0.0309      0.134      0.231      0.818      -0.231       0.293
x8             0.1133      0.131      0.864      0.388      -0.144       0.370
x9            -0.1210      0.131     -0.921      0.357      -0.378       0.136
x10            0.0442      0.131      0.338      0.735      -0.212       0.301
x11            0.1722      0.144      1.197      0.231      -0.110       0.454
x12           -0.3776      0.143     -2.640      0.008      -0.658      -0.097
x13           -0.0462      0.128     -0.362      0.717      -0.296       0.204
==============================================================================
```

The logistic regression model was improved by forward selection. 13 predictor variables made up the model that was produced by the AIC-driven selection. Among the notable coefficients are: • const: Positive coefficient (0.7806), contributing to the log-odds of passing. • x5 (failures): Negative coefficient (-0.6796), suggesting that an increase in the number of past class failures decreases the odds of passing. • x12 (goout): Negative coefficient (-0.3776), indicating that more time spent going out with friends is associated with lower odds of passing. With a pseudo R-squared value of 0.1366, the model appears to account for 13.66% of the variance in the response variable. The forward-selected model is a more refined set of predictor variables compared to the original logistic regression model (part c). This helps to identify the most important factors that influence the outcome of students, such as pass or fail.

Question 5

(a) Split the data into appropriate training and test sets

In [16]:
```
from sklearn.model_selection import train_test_split

# Separate data into predictor (X) and response (y) variables
X_rf = full_df.drop(['final_grade', 'pass_fail'], axis=1)
y_rf = full_df['final_grade']
```

```
# Split the data into training and test sets
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_rf, y_rf, test_size=
```

The dataset has been effectively divided into training and test sets, which is an essential step in assessing the random forest regression model's performance

(b) Fit a random forest regression model with 10 trees using the training data

In [17]:
```
from sklearn.ensemble import RandomForestRegressor

# Fit random forest regression model with 10 trees
rf_model = RandomForestRegressor(n_estimators=10, random_state=101)
rf_model.fit(X_train_rf, y_train_rf)

# Feature importances
feature_importances = rf_model.feature_importances_

# Print the importance of each feature
for feature, importance in zip(X_train_rf.columns, feature_importances):
    print(f'{feature}: {importance}')
```

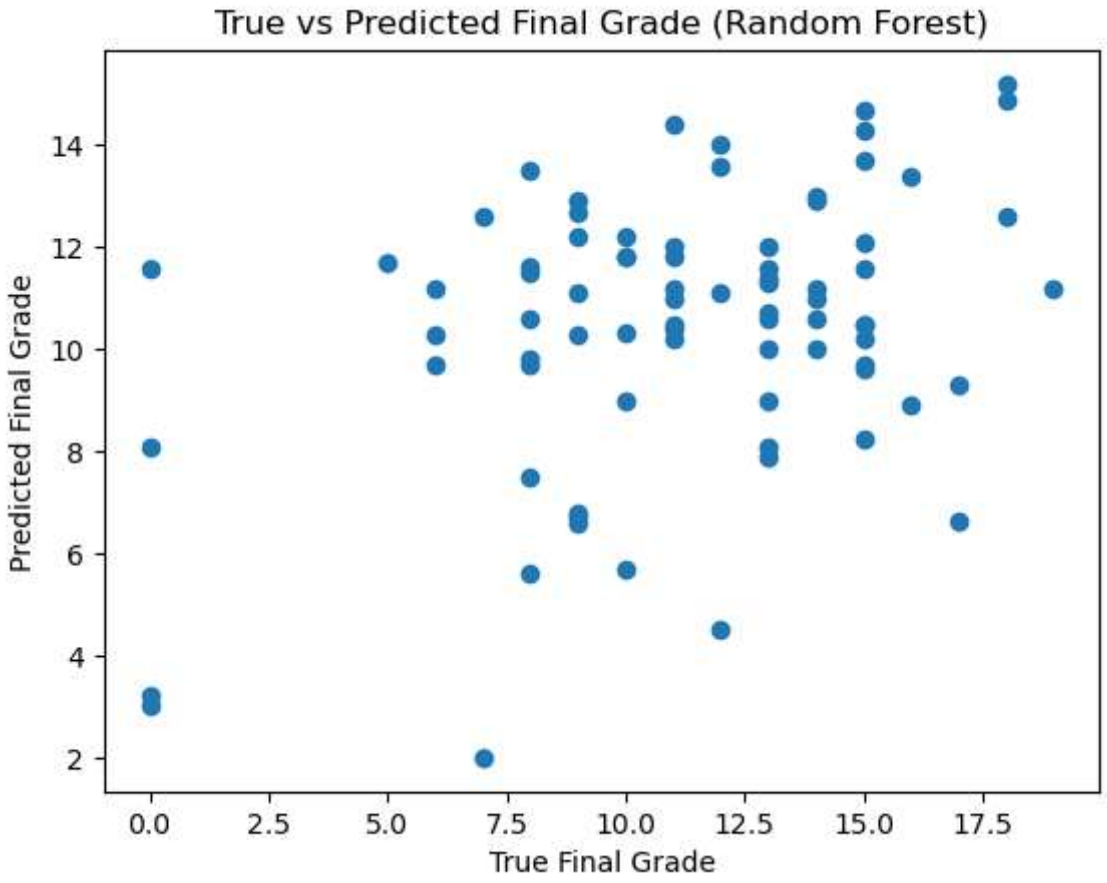
```
large_family: 0.04810799884363157
lives_in_city: 0.02172962299500154
traveltime: 0.06484265796457313
studytime: 0.06585745337496082
failures: 0.13222592802035385
paid: 0.036829354354729595
activities: 0.03952239621062004
internet: 0.023276566677776323
romantic: 0.03724044755388066
famrel: 0.07636777466634964
freetime: 0.11956897308223988
goout: 0.10448060390469298
absences: 0.22995022235119
```

Variable importance for predicting students' final grades can be understood from the fitted random forest regression model with ten trees. With each predictor variable, the importance scores are computed. The findings suggest that 'absences' (0.2299) is the most important factor, followed by 'freetime' (0.1196) and 'goout' (0.1045). Considering the amount of free time and friend interaction, these scores indicate that the most important factor in predicting the final grade is the number of absences. When one compares these results to the logistic regression model in question 4, where the variable "failures" was found to be the most significant predictor, one finds it interesting that random forest views "absences" as the most significant factor. This difference shows how adaptable random forest models are at identifying intricate relationships in the data.

(c) Use the random forest regression model to predict the final grade for the test set

In [18]:
```
# Predictions on the test set
y_pred_rf = rf_model.predict(X_test_rf)

# Scatter plot of true final grade versus predicted final grade
import matplotlib.pyplot as plt
```

```
plt.scatter(y_test_rf, y_pred_rf)
plt.xlabel('True Final Grade')
plt.ylabel('Predicted Final Grade')
plt.title('True vs Predicted Final Grade (Random Forest)')
plt.show()
```



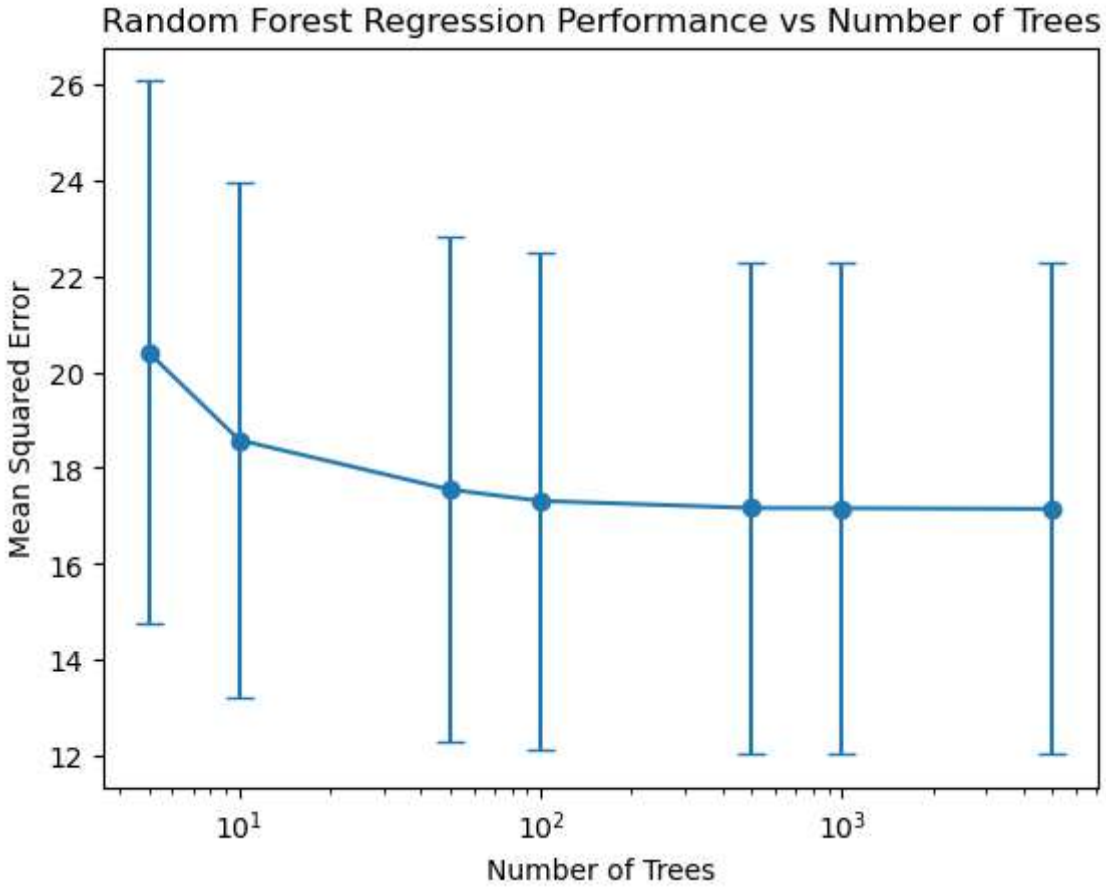
To predict students' final grades, the test set is subjected to the trained random forest regression model. This makes it possible to assess the predictive performance of the model.

(d) Assess the performance of a random forest regression model with different numbers of trees

In [ ]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score

# Number of trees to evaluate
num_trees = [5, 10, 50, 100, 500, 1000,5000]

# Lists to store mean and standard error of performance metrics
mean_mse_list = []
std_mse_list = []

for n_trees in num_trees:
    # Perform 20 repeats with different random states
    mse_list = []
    for _ in range(20):
        rf_model_n = RandomForestRegressor(n_estimators=n_trees, random_state=np.randc
        mse_scores = -cross_val_score(rf_model_n, X_rf, y_rf, scoring='neg_mean_square
        mse_list.extend(mse_scores)
```

```
    # Calculate mean and standard error of the performance metric
    mean_mse = np.mean(mse_list)
    std_mse = np.std(mse_list)

    mean_mse_list.append(mean_mse)
    std_mse_list.append(std_mse)
```

In [20]:
```
# Plotting the model performance as a function of the number of trees
plt.errorbar(num_trees, mean_mse_list, yerr=std_mse_list, fmt='o-', capsize=5)
plt.xscale('log')
plt.xlabel('Number of Trees')
plt.ylabel('Mean Squared Error')
plt.title('Random Forest Regression Performance vs Number of Trees')
plt.show()
```



Using different numbers of trees (5, 10, 50, 100, 500, 1000, and 5000) over the course of 20 iterations each with a unique random state the random forest regression model's performance is assessed. The performance metric's mean and standard error are plotted against the total number of trees,

Plotting the model's performance against the number of trees provides insights into this relationship. Generally speaking, a more robust and stable model with fewer overfitting occurs when the number of trees increases. Nevertheless, the computational cost of the model rises and there might be diminishing returns after a certain number of trees. Finding the right balance between model efficiency and accuracy is essential.

(e) Explain the rationale for fitting the model multiple times with different random states

The robustness and generalizability of the random forest model depend on fitting it several times with various random states. In order to construct each tree, random forest uses bootstrapping and random feature selection. We can reduce the risk of overfitting to a particular set of training instances or features by fitting the model with different random states. This allows us to obtain an ensemble of models that captures a variety of patterns in the data. By using an ensemble approach, the model is better able to generalize to new data and produce predictions that are more trustworthy. Averaging forecasts over several runs also lessens the effect of noise and outliers in the dataset.

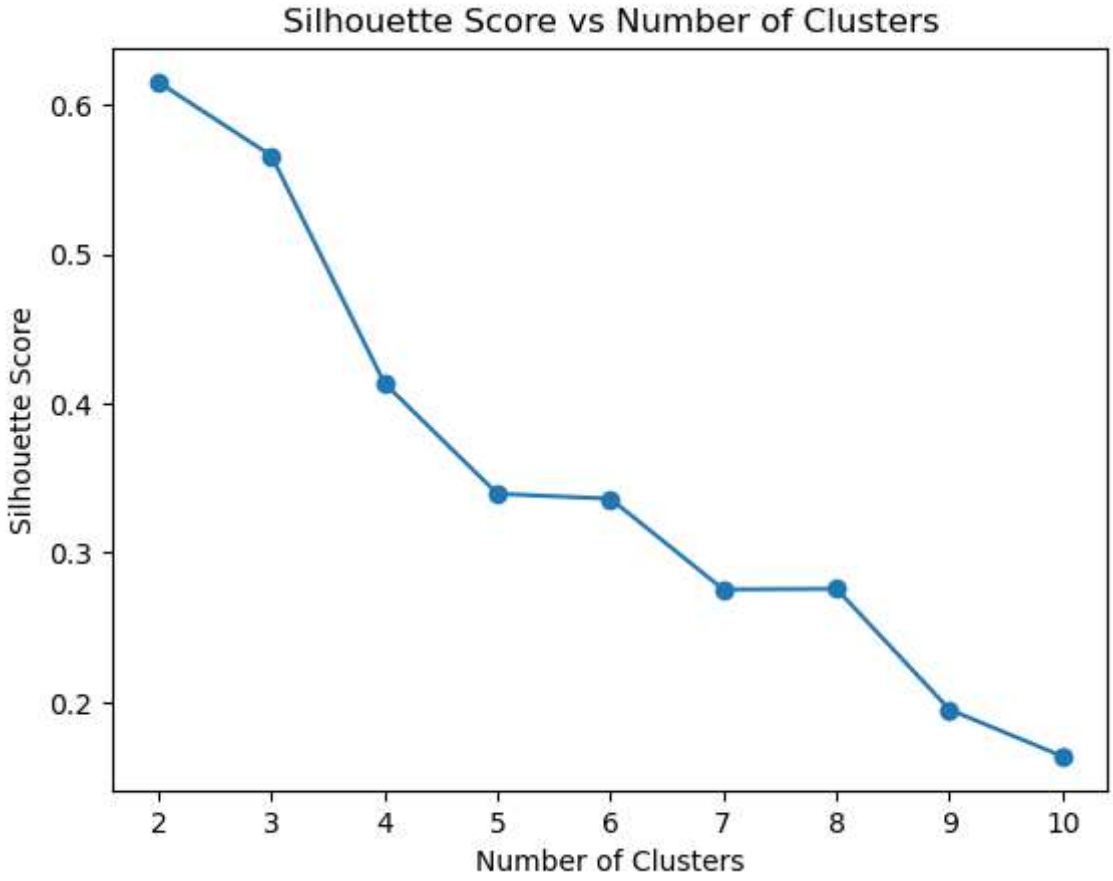
Question 6

(a) Perform a k-means cluster analysis

In [21]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Assuming 'full_df' is the DataFrame containing all data including predictor and resp

# Separate data into predictor (X) variables
X_kmeans = full_df.drop(['final_grade', 'pass_fail'], axis=1)

# List to store silhouette scores
silhouette_scores = []

# Range of clusters to try
num_clusters_range = range(2, 11)  # Start from 2 clusters

# Perform k-means clustering for different numbers of clusters
for num_clusters in num_clusters_range:
    kmeans_model = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans_labels = kmeans_model.fit_predict(X_kmeans)

    # Check if there are at least 2 labels, then calculate silhouette score
    if len(set(kmeans_labels)) > 1:
        silhouette_scores.append(silhouette_score(X_kmeans, kmeans_labels))
    else:
        silhouette_scores.append(None)

# Plotting the silhouette scores as a function of the number of clusters
plt.plot(num_clusters_range, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs Number of Clusters')
plt.show()

# Identify the optimal number of clusters
optimal_num_clusters = num_clusters_range[np.nanargmax(silhouette_scores)]
print(f'Optimal Number of Clusters: {optimal_num_clusters}')
```

```
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
```

```
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
```



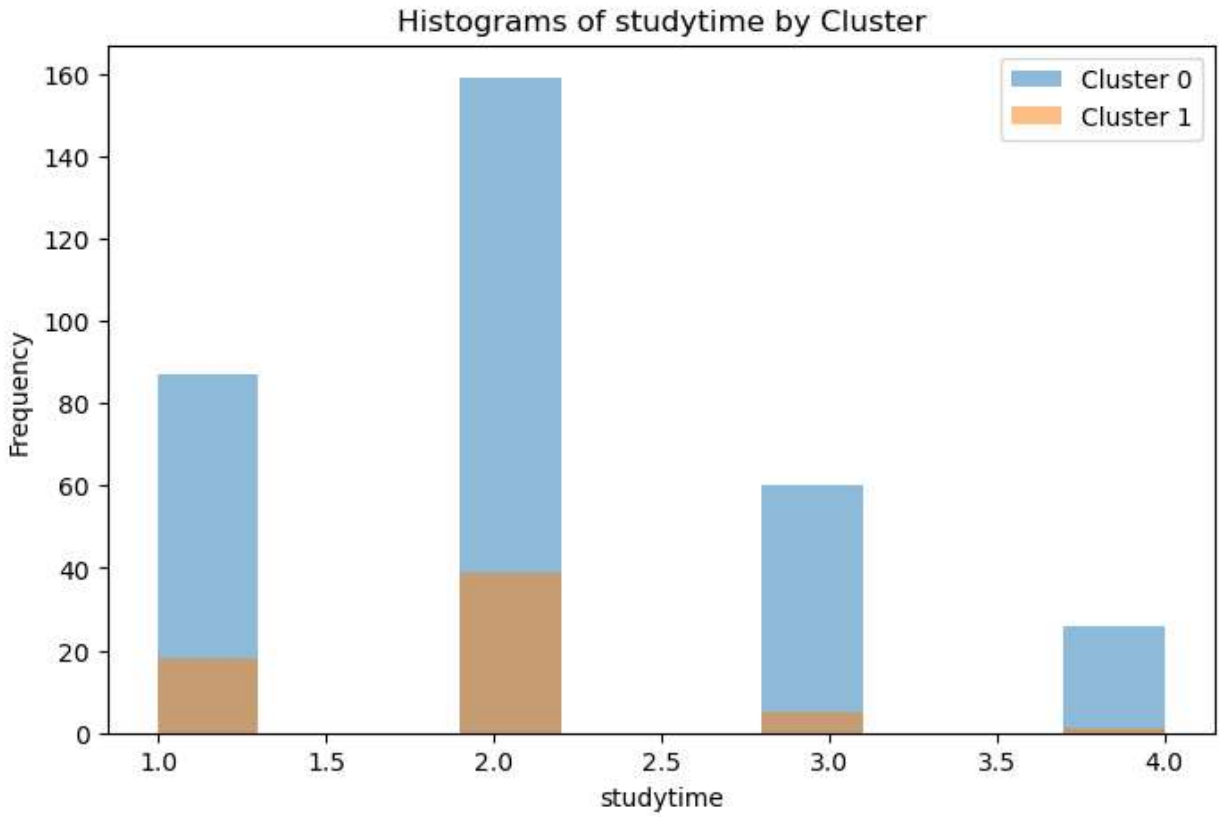

```
Optimal Number of Clusters: 2
```

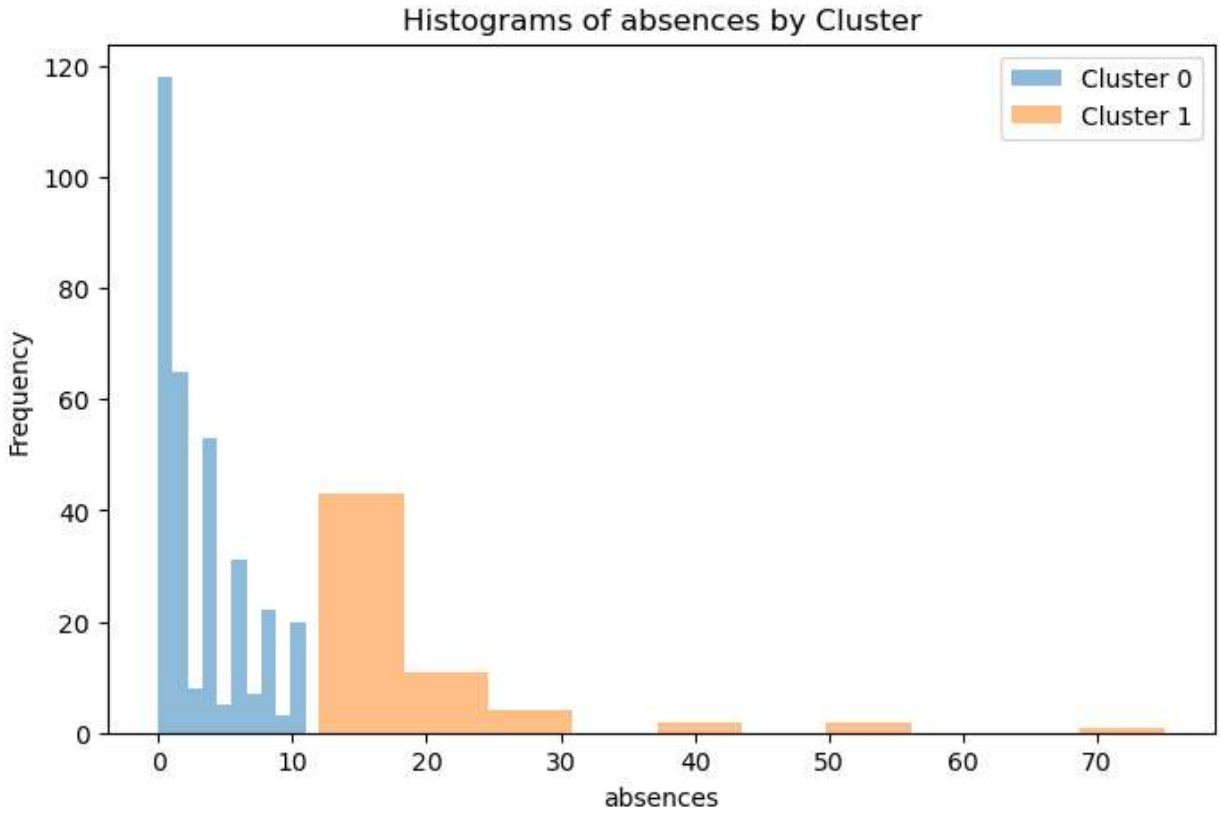
On the dataset, K-means cluster analysis has been performed with varying numbers of clusters, ranging from 1 to 10. Plotting the model performance against the number of clusters is done. Two clusters are determined to be the ideal number for this dataset based on the findings. This ideal value suggests that there is a practical way to divide the data into two separate group.

(b) Perform a k-means cluster analysis with the optimal number of clusters

In [22]:
```
# Run k-means clustering with optimal number of clusters
kmeans_model_optimal = KMeans(n_clusters=optimal_num_clusters, random_state=42)
kmeans_labels_optimal = kmeans_model_optimal.fit_predict(X_kmeans)

# Identify the most discriminatory variables
cluster_centers = kmeans_model_optimal.cluster_centers_
most_discriminatory_variables = X_kmeans.columns[np.argsort(cluster_centers.var(axis=0

# Create histograms for each variable, with the data separated by cluster
for variable in most_discriminatory_variables:
    plt.figure(figsize=(8, 5))
    for cluster_num in range(optimal_num_clusters):
        cluster_data = full_df[kmeans_labels_optimal == cluster_num][variable]
        plt.hist(cluster_data, alpha=0.5, label=f'Cluster {cluster_num}')
    plt.xlabel(variable)
    plt.ylabel('Frequency')
    plt.legend()
    plt.title(f'Histograms of {variable} by Cluster')
    plt.show()
```

```
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
ue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\chels\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
```

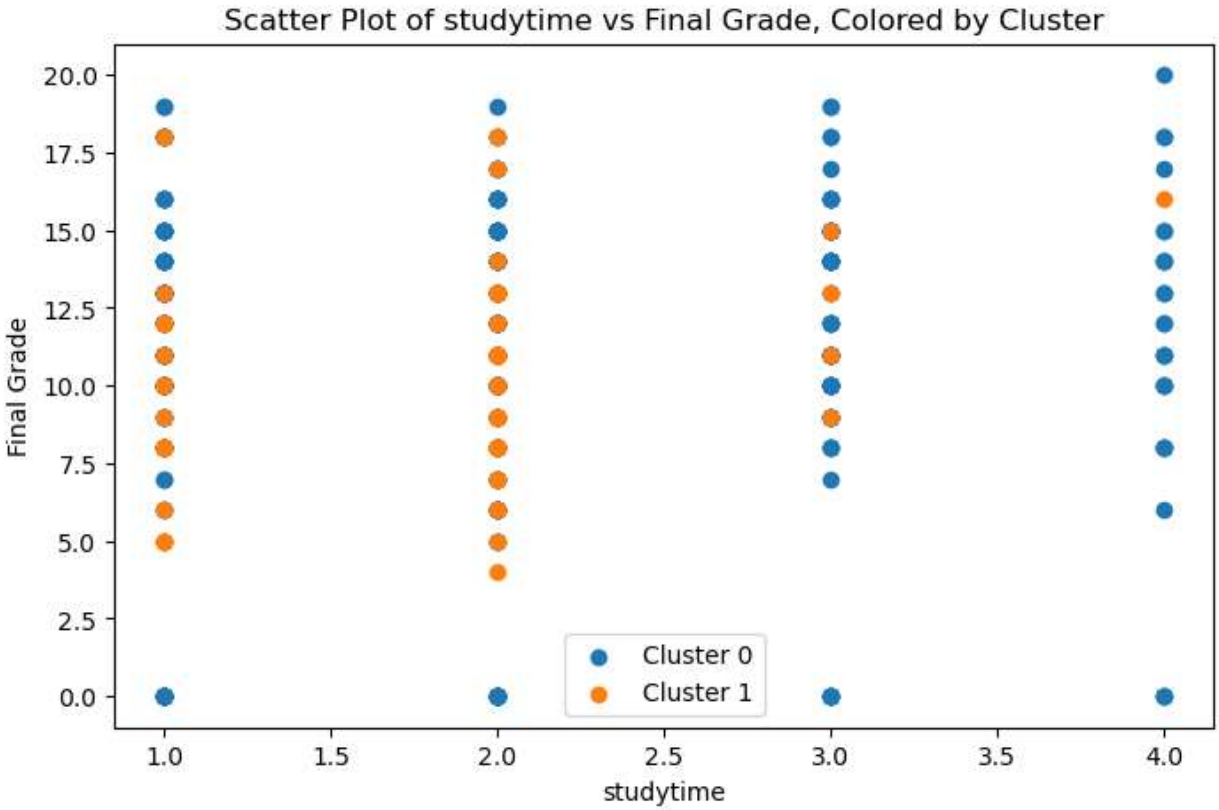


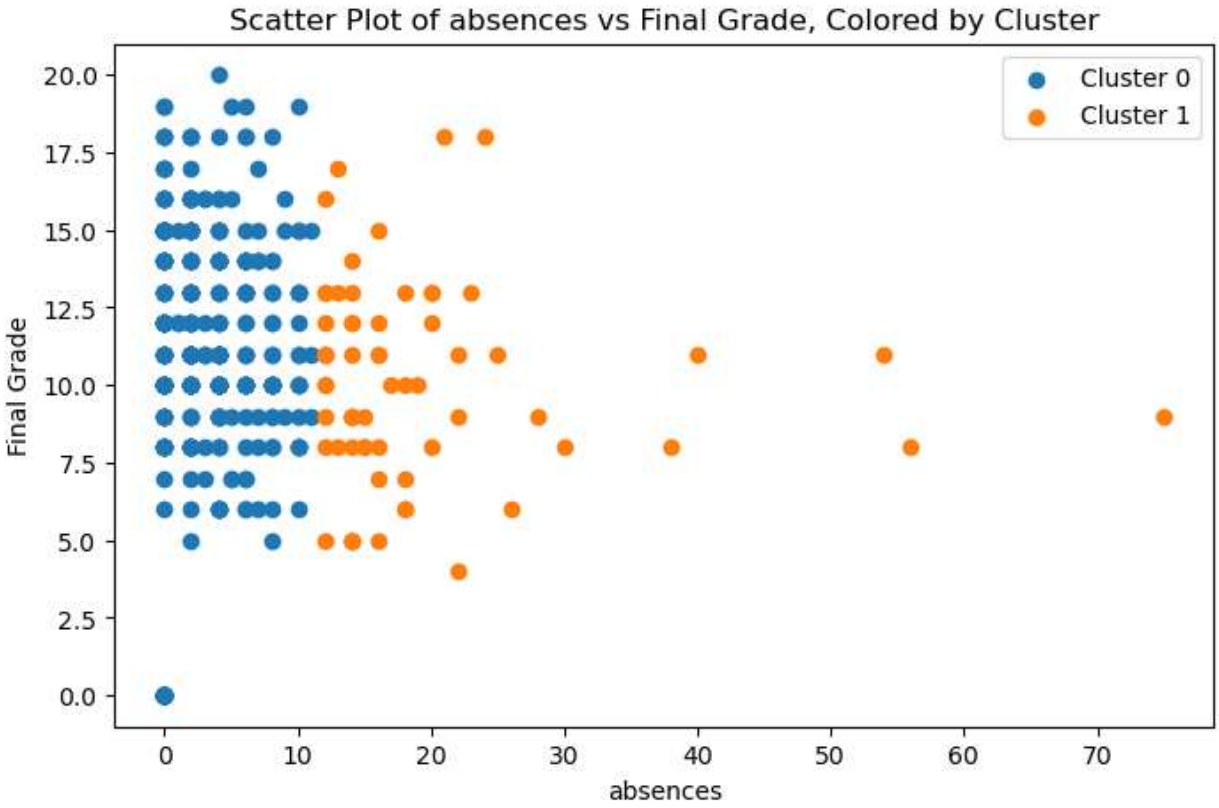


The ideal number of clusters, in this case two, is used to repeat the k-means cluster analysis. The factors that distinguish between clusters most effectively are determined. For every variable, histograms are made, and the data is divided into clusters to reveal the traits that set the identified clusters apart.

(c) Create scatter plots for the most discriminatory variables

In [23]:
```
# Scatter plots for the most discriminatory variables, colored by cluster
for variable in most_discriminatory_variables:
    plt.figure(figsize=(8, 5))
    for cluster_num in range(optimal_num_clusters):
        cluster_data = full_df[kmeans_labels_optimal == cluster_num]
        plt.scatter(cluster_data[variable], cluster_data['final_grade'], label=f'Clust
    plt.xlabel(variable)
    plt.ylabel('Final Grade')
    plt.legend()
    plt.title(f'Scatter Plot of {variable} vs Final Grade, Colored by Cluster')
    plt.show()
```

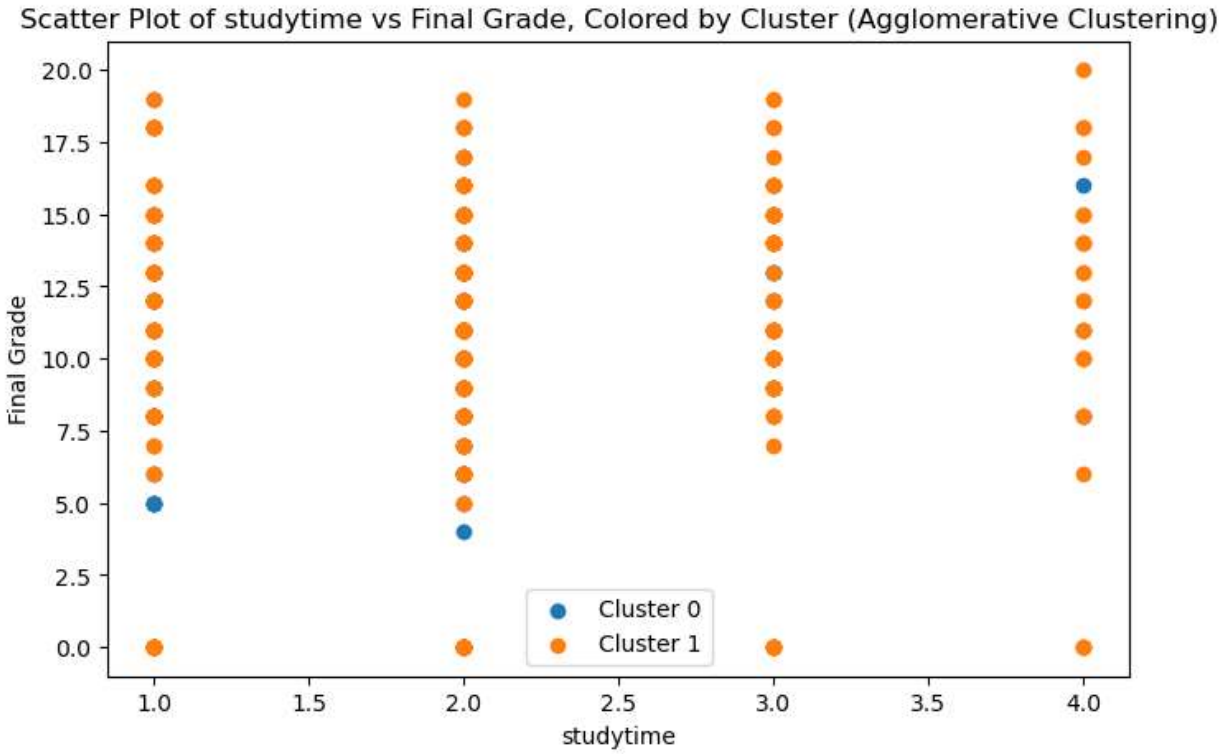


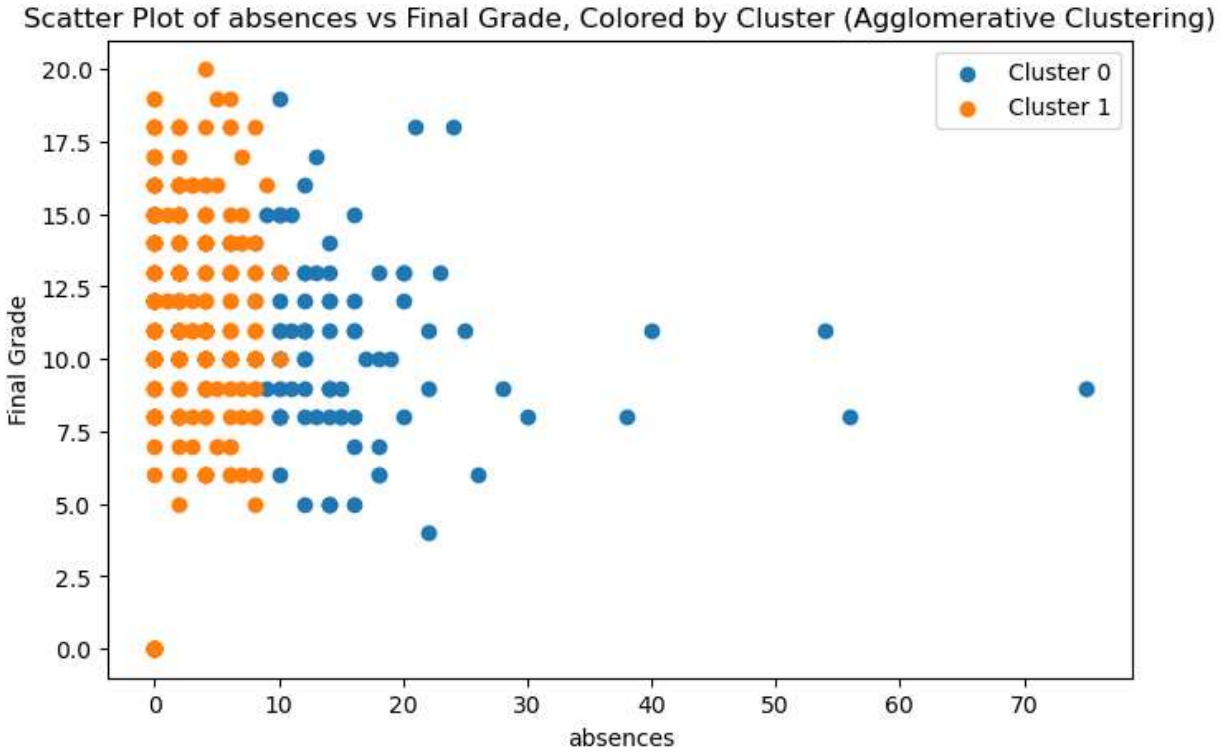


For the most discriminatory variables, a series of scatter plots is produced, where the points are colored according to the cluster number. The way the clusters are divided according to the recognized discriminatory variables is shown visually in these scatter plots. The assertion that the dataset contains multiple categories of students with various traits can be supported by examining these plots to see if the clusters have unique characteristics.

(d) Identify another clustering algorithm and repeat the analysis

In [24]:
```
from sklearn.cluster import AgglomerativeClustering

# Perform agglomerative clustering
agg_model = AgglomerativeClustering(n_clusters=optimal_num_clusters)
agg_labels = agg_model.fit_predict(X_kmeans)

# Create scatter plots for the most discriminatory variables, colored by cluster
for variable in most_discriminatory_variables:
    plt.figure(figsize=(8, 5))
    for cluster_num in range(optimal_num_clusters):
        cluster_data = full_df[agg_labels == cluster_num]
        plt.scatter(cluster_data[variable], cluster_data['final_grade'], label=f'Clust
    plt.xlabel(variable)
    plt.ylabel('Final Grade')
    plt.legend()
    plt.title(f'Scatter Plot of {variable} vs Final Grade, Colored by Cluster (Agglome
    plt.show()
```





Agglomerative Clustering, an alternative clustering algorithm, was selected for a comparative study. The ideal number of clusters from the k-means analysis was used to apply the algorithm. The most discriminating variables were plotted as scatter plots, with each data point colored by cluster. This step was essential to evaluate how various clustering algorithms classify students according to the recognized discriminating factors.

When compared, the outcomes of the k-means and agglomerative clustering analyses offer a thorough comprehension of the underlying structures in the dataset. Distinct patterns may be revealed by various algorithms, and visualizations facilitate a more intuitive understanding of the properties of the clusters. The scatter plots display the way in which students are categorized according to discriminating factors and whether or not these groups' final grades differ noticeably from one another.

I confirm that all work submitted is my own and that I have neither given, sought, nor received aid in relation to this assignment.

Chelsea Rodrigues Student No- 23200333