STAT40800 - Final Project

Suraj Bodhanandan Nhattuvetty - 23200338

In [139]:

```
#Import the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import numpy as np
```

Question 1

a) Load the dataset

In [140]:
```
male = pd.read_csv("male_stud.csv")
male
```

Out[140]:

	large_family	lives_in_city	traveltime	studytime	failures	pai
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
...	...	...	...	...	...	...
182	0	1	1	2	2	1
183	0	1	2	1	0	0
184	1	0	1	1	3	0
185	0	0	3	1	0	0
186	0	1	1	1	0	0

187 rows × 14 columns

b) Inspect the data

In [141]:
```
#Get the dimensions of the dataframe
male.shape
```

Out[141]: (187, 14)

The dataset includes the data of 187 students. There are 14 different indicators that are used to describe the data.

In [142]:
```
list(male.columns)
```

Out[142]:
```
['large_family',
 'lives_in_city',
 'traveltime',
 'studytime',
 'failures',
 'paid',
 'activities',
 'internet',
 'romantic',
 'famrel',
 'freetime',
 'goout',
 'absences',
 'final_grade']
```

The 14 different indicators can be seen as above

In [143]:
```
#Check if there are any missing values
male.isnull().values.any()
male.isna().sum()
```

Out[143]:
```
large_family     0
lives_in_city    0
traveltime       0
studytime        0
failures         0
paid             0
activities       0
internet         0
romantic         0
famrel           0
freetime         0
goout            0
absences         0
final_grade      0
dtype: int64
```

There are no missing values in the given dataset.

c) Exploratory data analysis

The indicators in the dataset are in the form numeric categories. A summary of them can be found in the following way

In [144]:
```
male.describe()
```

Out[144]:

	large_family	lives_in_city	traveltime	studytime	failure
count	187.000000	187.000000	187.000000	187.000000	187.00000
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

It is a good idea to describe the data in the below manner as most of the values act like a categorical data.

In [145]:
```
male.astype('object').describe().transpose()
```

Out[145]:

	count	unique	top	freq
large_family	187	2	1	125
lives_in_city	187	2	1	143
traveltime	187	4	1	118
studytime	187	4	2	85
failures	187	4	0	144
paid	187	2	0	114
activities	187	2	1	105
internet	187	2	1	159
romantic	187	2	0	134
famrel	187	5	4	88
freetime	187	5	3	64
goout	187	5	3	60
absences	187	23	0	52
final_grade	187	17	10	26

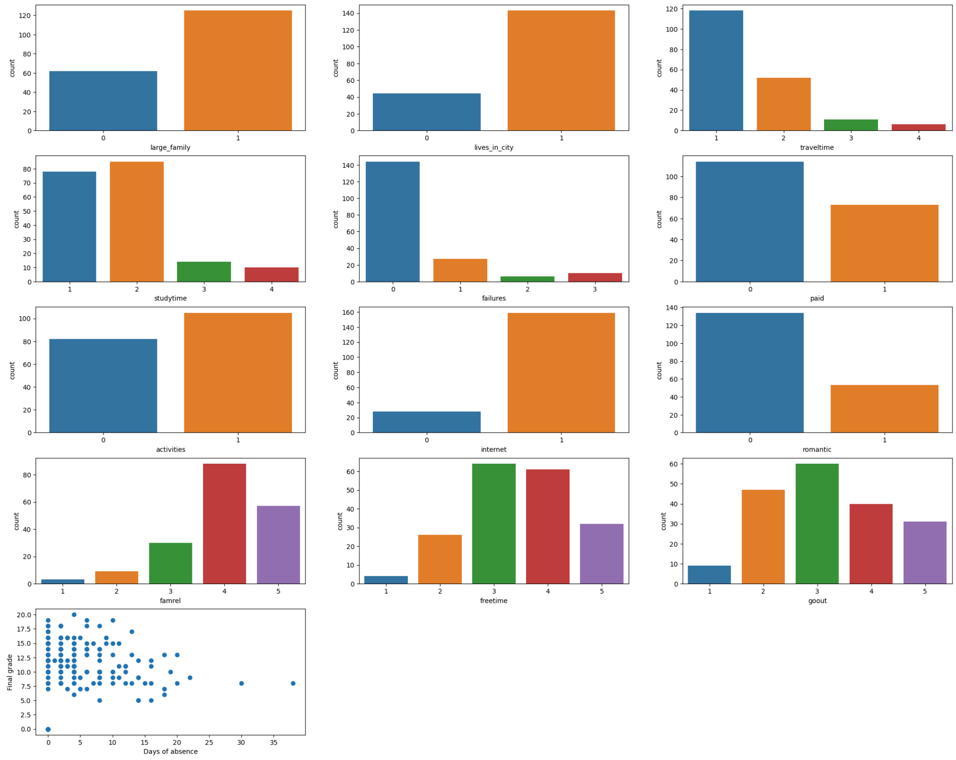
Looking at the dataset, most male students have a family with more than 3 members. Most also choose to live in urban areas. Most also opt to stay in areas which takes less than 15 minutes to travel to school.On an average they spend 2-5 hours weekly on their studies. Most students do not have any past failures and did not opt for extra paid classes. Most also did take part in extra curricular activities. Almost everyone had internet access at home. Most students were not in a romantic relationship. Approximately 50 % of the students had a very good relation with their family. On an average, students had a good amount of freetime. Students do go out with their friends for a decent amount of time.

The average number of absences is approximately 5 days with a standard deviation of approximately 6 days. The maximum number of absences is 38 and the minimum is 0. The average grade of student approximately 11 with a standard deviation of approximately 4.5. The minium grade is 0 and maximum is 20.

In [146]:

```
#Graphical summaries
fig = plt.figure(figsize=(25,20))
plt.subplot(5,3,1)
sns.countplot(x="large_family", data=male)

plt.subplot(5,3,2)
sns.countplot(x="lives_in_city", data=male)

plt.subplot(5,3,3)
sns.countplot(x="traveltime", data=male)

plt.subplot(5,3,4)
sns.countplot(x="studytime", data=male)

plt.subplot(5,3,5)
sns.countplot(x="failures", data=male)

plt.subplot(5,3,6)
sns.countplot(x="paid", data=male)

plt.subplot(5,3,7)
sns.countplot(x="activities", data=male)

plt.subplot(5,3,8)
sns.countplot(x="internet", data=male)

plt.subplot(5,3,9)
sns.countplot(x="romantic", data=male)

plt.subplot(5,3,10)
sns.countplot(x="famrel", data=male)

plt.subplot(5,3,11)
sns.countplot(x="freetime", data=male)

plt.subplot(5,3,12)
sns.countplot(x="goout", data=male)

plt.subplot(5,3,13)
plt.scatter(male['absences'], male['final_grade'])
plt.xlabel('Days of absence')
plt.ylabel('Final grade')
```

Out[146]: Text(0, 0.5, 'Final grade')



- Most students have a large family meaning there are more than 3 members in their family.
- Almost all the students live in the city with only very few in rural areas.
- Most students have a travel time to school less than 15 mins. Travel time and the number of students under that category are inversely proportional.
- Most students study between 0 to 5 hours. Students studying more than this are in very few numbers.
- Majority of the students do not have past class failures. Approximately 10% of students have past class failures.
- A lot of students have not paid for any extra classes but there is also significant number of students who have paid for it.
- Students do take part extra curricular activities but an almost 40% of students do not take part in any such activities.
- Almost all students have internet access at their homes with only a few students without such access.
- Most students were not in a romantic relationship, approximately 25% students were in one.
- Students have a really good relation with their family with only less than 10% students have a poor relation with the family
- The students also had a good amount of freetime.
- Most students did go out with their friends with only a small percentage.
- There is negative relation between final grade and number of absences. As the days of absence increases, the final grade decreases.

Question 2

a) Load the dataset

In [147]:
```
#Load the dataset
female = pd.read_csv("female_stud.csv")
female
```

Out[147]:

	large_family	lives_in_city	traveltime	studytime	failures	pai
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
...	...	...	...	...	...	...
203	1	0	2	3	0	1
204	1	0	3	1	0	1
205	1	0	1	3	1	0
206	0	1	1	2	0	1
207	1	1	2	2	1	0

208 rows × 14 columns

b) Inspect the data

In [148]:
```
#Dimension of the dataset
female.shape
```

Out[148]: (208, 14)

There are 208 students and 14 indicators included in the dataset.

In [149]:
```
#Comparing indicators of male and female dataset
print(list(male.columns)==list(female.columns))
list(female.columns)
```

```
True
```

Out[149]:
```
['large_family',
 'lives_in_city',
 'traveltime',
 'studytime',
 'failures',
 'paid',
 'activities',
 'internet',
 'romantic',
 'famrel',
 'freetime',
 'goout',
 'absences',
 'final_grade']
```

c) Perform t-test

To check whether any indicators differ in the male and female group, we perform the t-test. We can consider 2 hypothesis:

- H0 - The indicators are equal in the male and female group
- H1 - The indicators differ in the male and female group

The significance level is taken as 0.01

In [150]:
```
#Perform t-test on the indicators
alpha = 0.01
t_values, p_values = stats.ttest_ind(male, female, equal_var=True)
ttest_df = pd.DataFrame({"Indicators": male.columns,
                        "t-statistic": t_values,
                        "p-value": p_values})

#Comparing p-value with significance level
print(ttest_df)
ttest_df.loc[ttest_df['p-value'] < alpha, 'Indicators']
```

```
      Indicators  t-statistic       p-value
0   large_family    -1.788677  7.443690e-02
1   lives_in_city   -0.565304  5.721898e-01
2     traveltime     1.186055  2.363171e-01
3      studytime    -6.378011  5.045044e-10
4       failures     0.881778  3.784358e-01
5           paid    -2.581427  1.020069e-02
6     activities     1.989053  4.738832e-02
7       internet     0.875356  3.819147e-01
8       romantic    -2.033136  4.271032e-02
9         famrel     1.171091  2.422716e-01
10      freetime     4.873860  1.590705e-06
11         goout     1.508960  1.321126e-01
12      absences    -1.330451  1.841411e-01
13   final_grade     2.061993  3.986533e-02
```

Out[150]:
```
3      studytime
10      freetime
Name: Indicators, dtype: object
```

From the t-test we see that, p-values of **studytime** and **freetime** are lesser than 0.01. Therefore we reject the null hypothesis for these two and accept it for the rest of the indicators. This means that there is a significant difference in the indicators **studytime** and **freetime** for male and female groups. When the p-value is lesser than the significance level it means that the result has not happened due to a random chance.

The indicators used in the female dataset are the same as that of the male dataset.

Question 3

a) Combine the dataframes

In [151]:
```
#Combine the dataset vertically
student = pd.concat([male, female], axis=0)
student.reset_index(inplace=True, drop=True)
student
```

Out[151]:

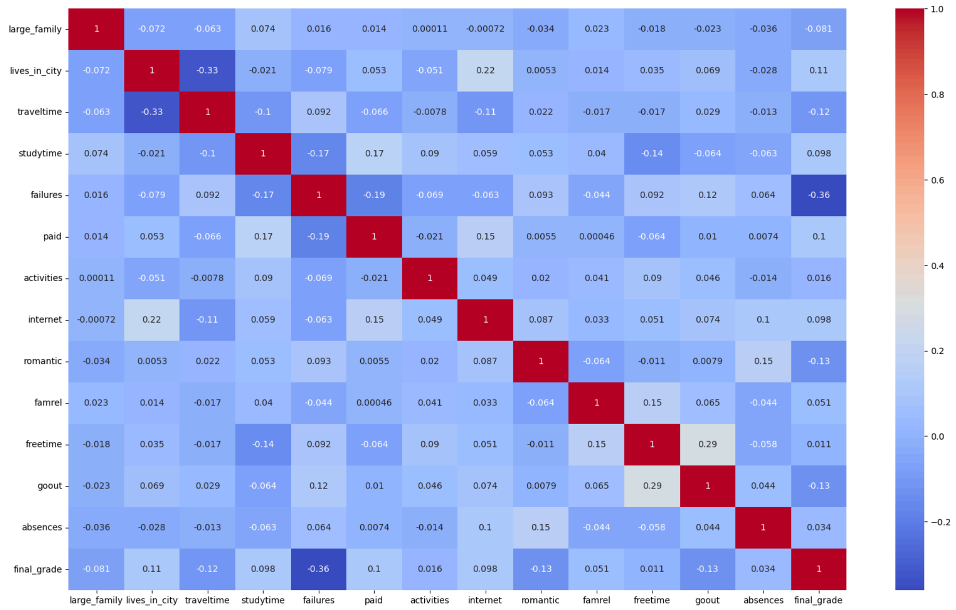
	large_family	lives_in_city	traveltime	studytime	failures	pai
0	0	1	1	2	0	1
1	0	1	1	2	0	0
2	0	1	1	2	0	1
3	1	1	1	2	0	1
4	0	1	1	1	0	1
...	...	...	...	...	...	...
390	1	0	2	3	0	1
391	1	0	3	1	0	1
392	1	0	1	3	1	0
393	0	1	1	2	0	1
394	1	1	2	2	1	0

395 rows × 14 columns

b) Compute Pearson correlation coefficient

In [152]:

```
#Compute Pearson correlation coefficient for every indicator
plt.figure(figsize=(20,12))
corr_mat = student.corr(method='pearson')
sns.heatmap(corr_mat, annot=True, cmap='coolwarm')
plt.show()
```



Find the most correlated pairs

In [153]:

```
#Find top 4 correlated pairs
corr_mat_abs = corr_mat.abs()

#Convert to upper triangular form
upper_mat = corr_mat_abs.where(np.triu(np.ones(corr_mat_abs.shape),
k=1).astype(bool))

#Drop all the null values
unique_pairs = upper_mat.unstack().dropna()

#Sort the values
sorted_mat = unique_pairs.sort_values()
print(sorted_mat)
```

```
activities   large_family     0.000113
famrel       paid             0.000460
internet     large_family     0.000720
romantic     lives_in_city    0.005257
             paid             0.005536
                                ...
paid         failures         0.188039
internet     lives_in_city    0.216842
goout        freetime         0.285019
traveltime   lives_in_city    0.328096
final_grade  failures         0.360415
Length: 91, dtype: float64
```

The four most strongly correlated pairs are:

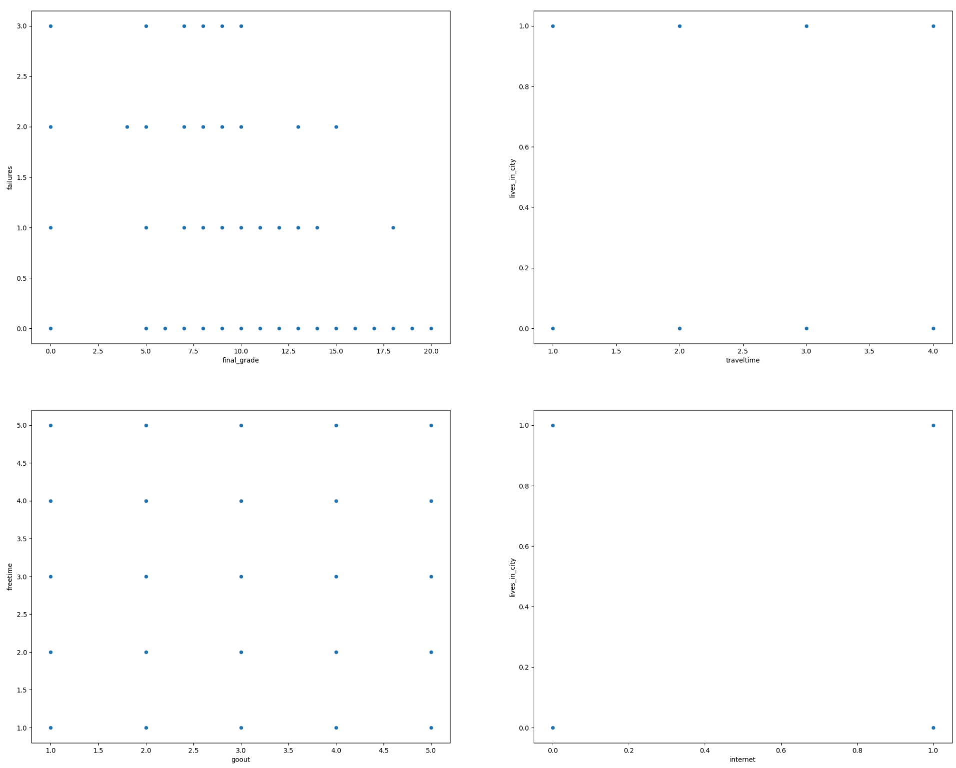
- final_grade and failures
- traveltime and lives_in_city
- goout and freetime
- internet and lives_in_city

c) Scatterplots

In [154]:

```
#Scatter plots for correlated pairs
fig = plt.figure(figsize=(25,20))
plt.subplot(2,2,1)
sns.scatterplot(x='final_grade', y='failures', data=student)

plt.subplot(2,2,2)
sns.scatterplot(x='traveltime', y='lives_in_city', data=student)

plt.subplot(2,2,3)
sns.scatterplot(x='goout', y='freetime', data=student)

plt.subplot(2,2,4)
sns.scatterplot(x='internet', y='lives_in_city', data=student)
```

Out[154]: <Axes: xlabel='internet', ylabel='lives_in_city'>



From the plot between final grade and failures, it can be easily seen that there is negative correlation. For the other plots, the correlation is harder to understand as most of the variables are like categorical values.

Question 4

a) Create new column

In [155]:
```
#Create new column Result
student['Result'] = (student['final_grade']>9.5).astype(int)
student.head()
```

Out[155]:

	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

A new column **Result** is created which indicates the pass or fail status of student. The column has binary values

- 0 - The student has failed
- 1 - The student has passed

b) Split data and standardise the variables

In [156]:
```
#Response variable
student_response = student['Result']
#Predictor variables
student_predictor = student.drop(['Result','final_grade'], axis=1)
#Standardise the variables
student_predictor_std = (student_predictor-
student_predictor.mean())/student_predictor.std()
```

c) Fit logistic regression model

In [157]:
```
#Import model
from sklearn.linear_model import LogisticRegression
type(student_response)
import statsmodels.api as sm
```

In [158]:
```
student_predictor_std.insert(0,'intercept',1)
student_predictor_std
```

Out[158]:

	intercept	large_family	lives_in_city	traveltime	studytime	
0	1	-1.568015	0.534714	-0.642435	-0.042232	-(
1	1	-1.568015	0.534714	-0.642435	-0.042232	-(
2	1	-1.568015	0.534714	-0.642435	-0.042232	-(
3	1	0.636134	0.534714	-0.642435	-0.042232	-(
4	1	-1.568015	0.534714	-0.642435	-1.233786	-(
...	...	...	...	...	...	...
390	1	0.636134	-1.865423	0.791247	1.149321	-(
391	1	0.636134	-1.865423	2.224929	-1.233786	-(
392	1	0.636134	-1.865423	-0.642435	1.149321	0
393	1	-1.568015	0.534714	-0.642435	-0.042232	-(
394	1	0.636134	0.534714	0.791247	-0.042232	0

395 rows × 14 columns

In [159]:
```
#Fit the model
logit1 = sm.Logit(student_response, student_predictor_std).fit()
print(logit1.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.556201
         Iterations 5
                           Logit Regression Results
==================================================================
============
Dep. Variable:                 Result   No. Observations:
395
Model:                          Logit   Df Residuals:
381
Method:                           MLE   Df Model:
13
Date:                Sat, 09 Dec 2023   Pseudo R-squ.:
0.1221
Time:                        11:55:04   Log-Likelihood:
-219.70
converged:                       True   LL-Null:
-250.25
Covariance Type:            nonrobust   LLR p-value:
3.330e-08
==================================================================
===============
                    coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------
---------------
intercept         0.7801      0.118      6.602      0.000
0.548       1.012
large_family     -0.1204      0.121     -0.998      0.318
-0.357       0.116
lives_in_city     0.0507      0.125      0.404      0.686
-0.195       0.297
traveltime        0.0063      0.126      0.050      0.960
-0.241       0.254
studytime         0.0400      0.125      0.320      0.749
-0.205       0.285
failures         -0.6692      0.130     -5.132      0.000
-0.925      -0.414
paid              0.0768      0.121      0.633      0.527
-0.161       0.315
activities       -0.0269      0.119     -0.226      0.821
-0.260       0.206
internet          0.1361      0.119      1.147      0.251
-0.096       0.369
romantic         -0.1623      0.117     -1.383      0.167
-0.392       0.068
famrel            0.0644      0.118      0.547      0.584
-0.166       0.295
freetime          0.1361      0.126      1.079      0.280
-0.111       0.383
goout            -0.4254      0.127     -3.344      0.001
-0.675      -0.176
```

```
absences         -0.1279      0.111     -1.149      0.250
-0.346      0.090
==================================================================
===============
```

In [160]:

```
logit1.aic
```

Out[160]: 467.39887281774656

d) Forward selection using AIC

We can implement a forward selection for regression in the following way:

In [161]:

```
#Function for forward AIC
def forwardAIC_logistic(X_std, Y, colnames):
    variables = ['intercept']
    mod = sm.Logit(Y,X_std[variables])
    res = mod.fit()
    mod_aic = res.aic
    aic_vals = dict()
    while(len(colnames)!=0):
        for p in colnames:
            variables.append(p)
            mod_new = sm.Logit(Y,X_std[variables])
            res_new = mod_new.fit()
            mod_aic_new = res_new.aic
            aic_vals[p] = mod_aic_new
            variables.remove(p)
        min_aic = min(aic_vals, key=aic_vals.get)
        min_aic_num = min(aic_vals)
        if min_aic in variables:
            break
        variables.append(min_aic)
        colnames.remove(min_aic)
    print(variables)
    return res_new
```

Use this function for the given data

In [162]:
```
columns = list(student_predictor_std.columns)
columns.remove('intercept')
logistic_new = forwardAIC_logistic(student_predictor_std,
student_response, columns)
```

```
Optimization terminated successfully.
         Current function value: 0.633549
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.632665
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.632203
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.632577
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.630725
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578633
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.629230
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.633472
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.631684
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.628837
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.632470
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.633381
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.616589
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.629482
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.577872
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578297
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578530
         Iterations 5
```

```
Optimization terminated successfully.
         Current function value: 0.578472
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578130
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578574
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.577727
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.576334
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578124
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.578563
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566845
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.576188
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.565885
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566075
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566759
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566802
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566159
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.566834
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.565287
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.564350
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.565976
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.564931
         Iterations 5
```

```
Optimization terminated successfully.
         Current function value: 0.564698
         Iterations 5
['intercept', 'failures', 'goout']
```

In [163]:
```
print(logistic_new.summary())
print(logistic_new.aic)
```

```
                           Logit Regression Results
==================================================================
============
Dep. Variable:                 Result   No. Observations:
395
Model:                          Logit   Df Residuals:
391
Method:                           MLE   Df Model:
3
Date:                Sat, 09 Dec 2023   Pseudo R-squ.:
0.1087
Time:                        11:55:04   Log-Likelihood:
-223.06
converged:                       True   LL-Null:
-250.25
Covariance Type:            nonrobust   LLR p-value:
9.253e-12
==================================================================
============
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------
------------
intercept      0.7592      0.116      6.531      0.000       0.531
0.987
failures      -0.7061      0.128     -5.535      0.000      -0.956
-0.456
goout         -0.3501      0.117     -2.980      0.003      -0.580
-0.120
absences      -0.1435      0.108     -1.330      0.183      -0.355
0.068
==================================================================
============
454.11132408212734
```

The new model obtained using forward stepwise regression has the variables **failures** , **goout** and **absences** .

The AIC value for model obtained from stepwise regeression is approximately 454.11 and for the previous model is 467.40. So the model obtained using the forward stepwise regeression is a better fit as the AIC value is lower.

Question 5

a) Split data into training and test sets

In [164]:
```
#Splitting data into response and predictor variables
X = student.drop(['Result','final_grade'], axis=1)
Y = student['final_grade']
```

The data is split into approximately 70% training data and 30% test data.

In [165]:
```
#Splitting the data into test and train
train_size = 277
np.random.seed(123)
train_select = np.random.permutation(range(len(Y)))
X_train = X.iloc[train_select[:train_size],:].reset_index(drop=True)
X_test = X.iloc[train_select[train_size:],:].reset_index(drop=True)
y_train = Y.iloc[train_select[:train_size]].reset_index(drop=True)
y_test = Y.iloc[train_select[train_size:]].reset_index(drop=True)
```

b) Fit random forest regression model

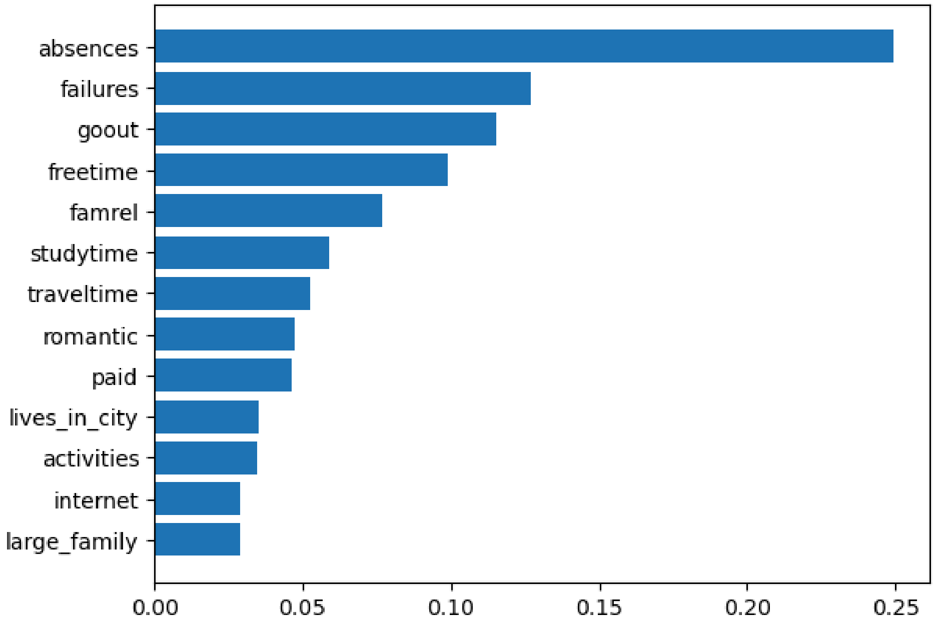
In [166]:
```
#Import random forest regressor
from sklearn.ensemble import RandomForestRegressor
#RandomForestRegressor?
```

In [167]:
```
#Fitting the model
rf = RandomForestRegressor(n_estimators=10, random_state=101)
rf_fit = rf.fit(X_train, y_train)
```

In [168]:

```
#Sorting important features
sorted_idx = rf.feature_importances_.argsort()
plt.barh(X.columns[sorted_idx], rf.feature_importances_[sorted_idx])
```

Out[168]: `<BarContainer object of 13 artists>`



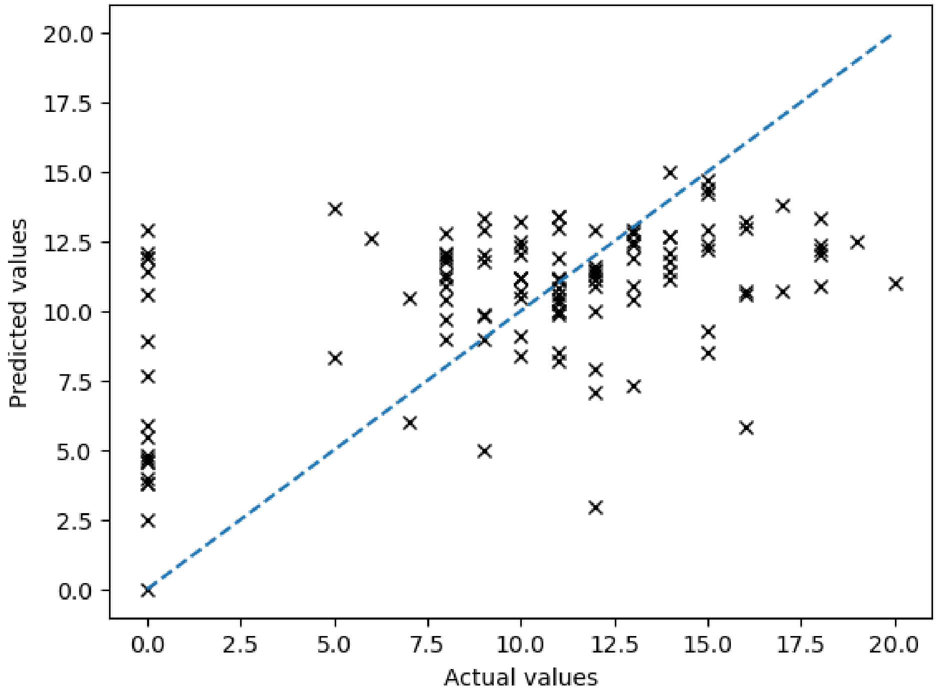
The important variables for predicting the final grade of a student are:
- absences
- failures
- goout
- freetime

The models fit in the question 4 and question 5 are very similar because the stepwise regression picked up the same features - absences, failures and goout. Stepwise regression does not give us an optimal model but it gives a better and good model. Now the random forest regressor model also suggests the same features to be important. So we can conclude that these features cannot be omitted for any other model as well.

c) Predict and create scatter plot

In [169]:
```
#Predict the values
rf_test_pred = rf.predict(X_test)
fig = plt.figure()
plt.plot(y_test,rf_test_pred,'kx')
plt.plot([0,20], [0,20], ls="--")
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
```

Out[169]: `Text(0, 0.5, 'Predicted values')`



The model is a decent fit looking at the plot. There are outliers for the lower value predictions. For values 10 and above, the prediction is better as it is close to the regression line, but there are also some points spread away from the line indicating more error in the prediction.

d) Fit and predict using different models

In [170]:
```
#Fitting the model
trees = [5, 10, 50, 100, 500, 1000, 5000]
perf_df = pd.DataFrame(columns=['Trees','Performance'])
pred_values = []
for item in trees:
    for i in range(20):
        rf = RandomForestRegressor(n_estimators=item,
random_state=101+i)
        rf_fit = rf.fit(X_train, y_train)
        rf_test_pred = rf.predict(X_test)
        pred_values.append(rf_test_pred)
        RSS_rf = np.mean(pow((rf_test_pred- y_test),2))
        perf_list = [item, RSS_rf]
        perf_df.loc[len(perf_df)] = perf_list
perf_df
```

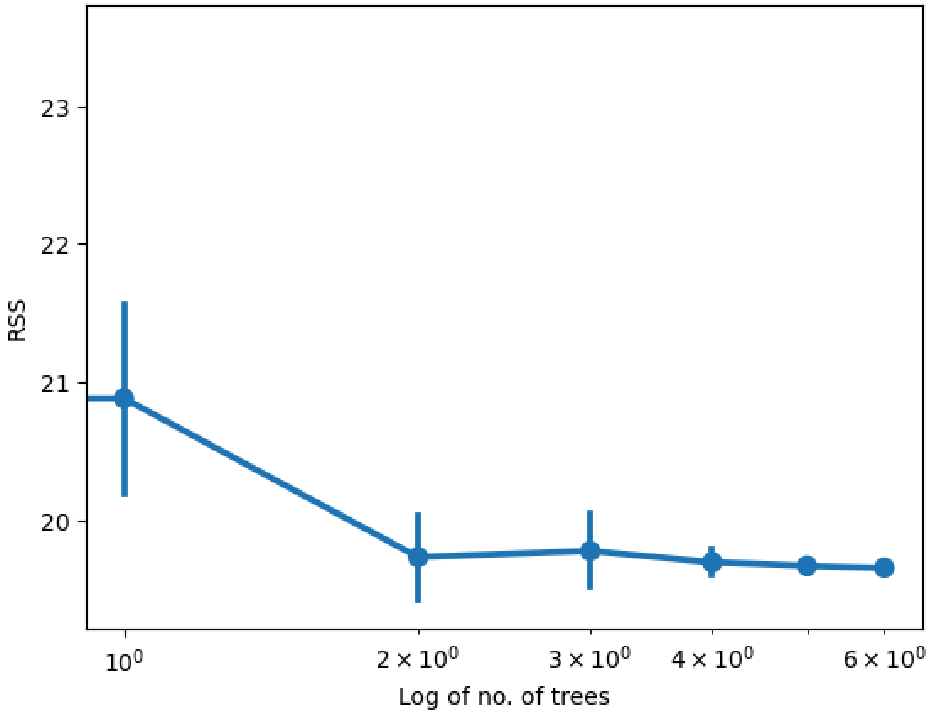
Out[170]:

	Trees	Performance
0	5.0	22.870847
1	5.0	23.177213
2	5.0	24.602524
3	5.0	26.756761
4	5.0	21.403729
...	...	...
135	5000.0	19.729556
136	5000.0	19.662193
137	5000.0	19.623193
138	5000.0	19.686962
139	5000.0	19.656232

140 rows × 2 columns

In [171]:

```
#Plot the performance metric
#plt.scatter(perf_df['Trees'], perf_df['Performance'], alpha=0.5)
sns.pointplot(x='Trees', y='Performance', data=perf_df)
plt.xscale('log')
plt.xlabel('Log of no. of trees')
plt.ylabel('RSS')
plt.show()
```



As the number of trees in the model increase, the RSS value and the standard deviation decreases. This means that the performance of the model is better but it also increases the computation time.

e) Models with different random states

Random state controls the shuffling of the data. If we just perform one fit with the a particular random state, we may not get proper RSS value. When we fit the model with different random states multiple times, the average RSS value should be close to the true RSS value for the dataset.

Question 6

a) K-means cluster analysis

In [172]:

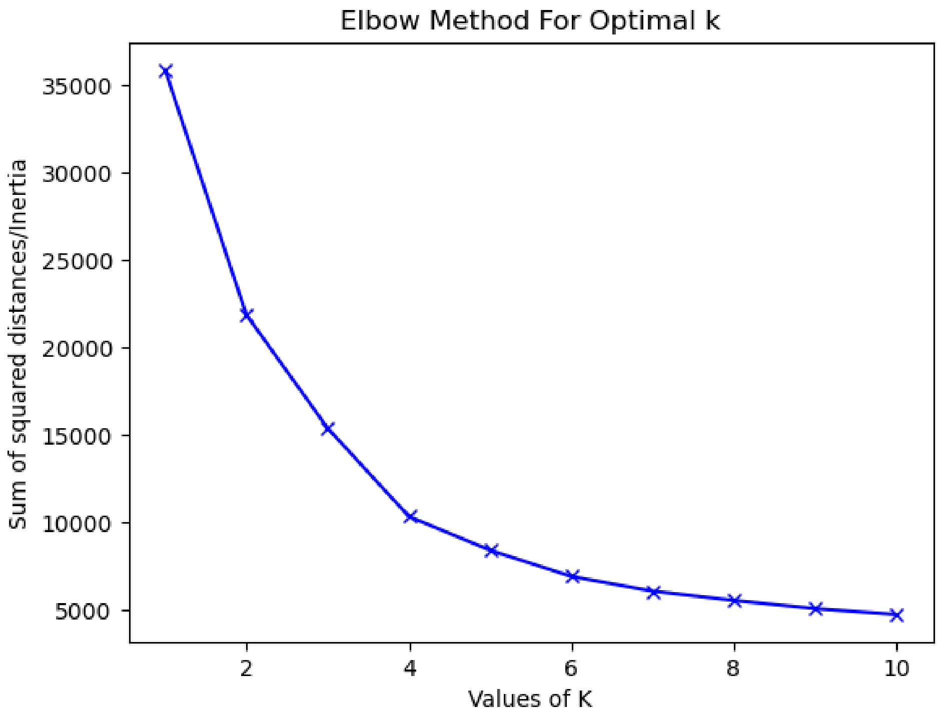
```
import warnings
warnings.filterwarnings('ignore')
```

In [173]:

```
#Import model
from sklearn.cluster import KMeans
```

In [174]:

```
#Fit k-means model
k_student = student.drop(['Result'], axis=1)
sse = []
for k in range(1, 11):
    km = KMeans(n_clusters=k, n_init=10)
    km.fit(k_student)
    sse.append(km.inertia_)
plt.plot(range(1,11), sse,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```



The optimal value for k is 4.

b) K-means cluster analysis with optimal k

In [175]:
```
km = KMeans(n_clusters=4, n_init=10)
km.fit(k_student)
preds = km.predict(k_student)
```

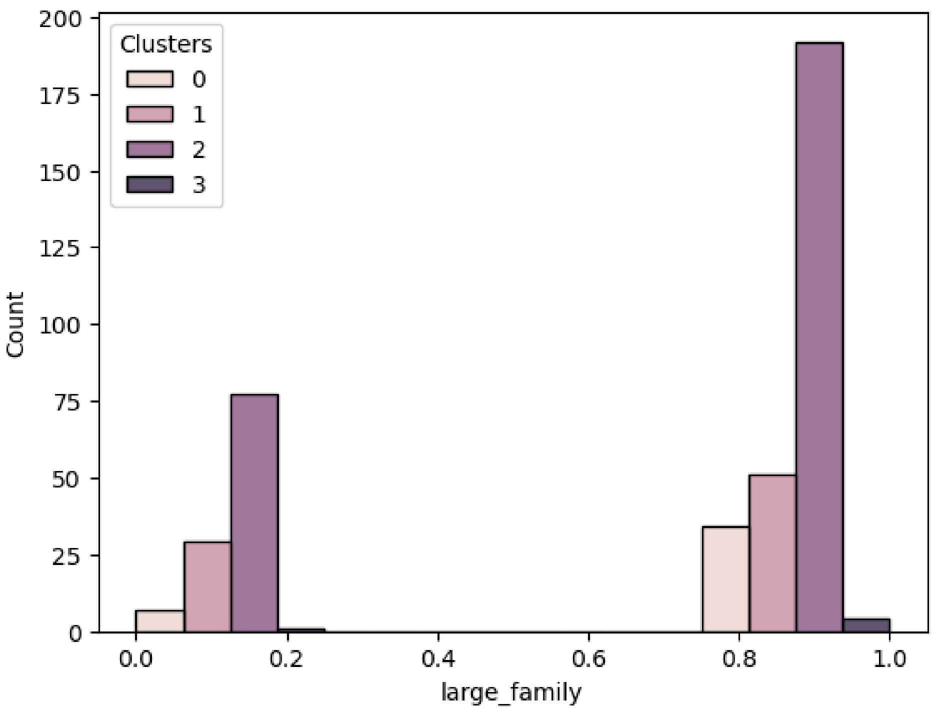
In [176]:
```
type(preds)
```

Out[176]: numpy.ndarray

In [177]:
```
#sns.countplot(x=preds, hue=k_student['large_family'], stat='percent')
pred_df = pd.DataFrame()
pred_df['Clusters'] = preds
temp_student = pd.concat([k_student, pred_df], axis='columns')
filter_df0 = temp_student[temp_student['Clusters']==0]
filter_df1 = temp_student[temp_student['Clusters']==1]
filter_df2 = temp_student[temp_student['Clusters']==2]
filter_df3 = temp_student[temp_student['Clusters']==3]
```

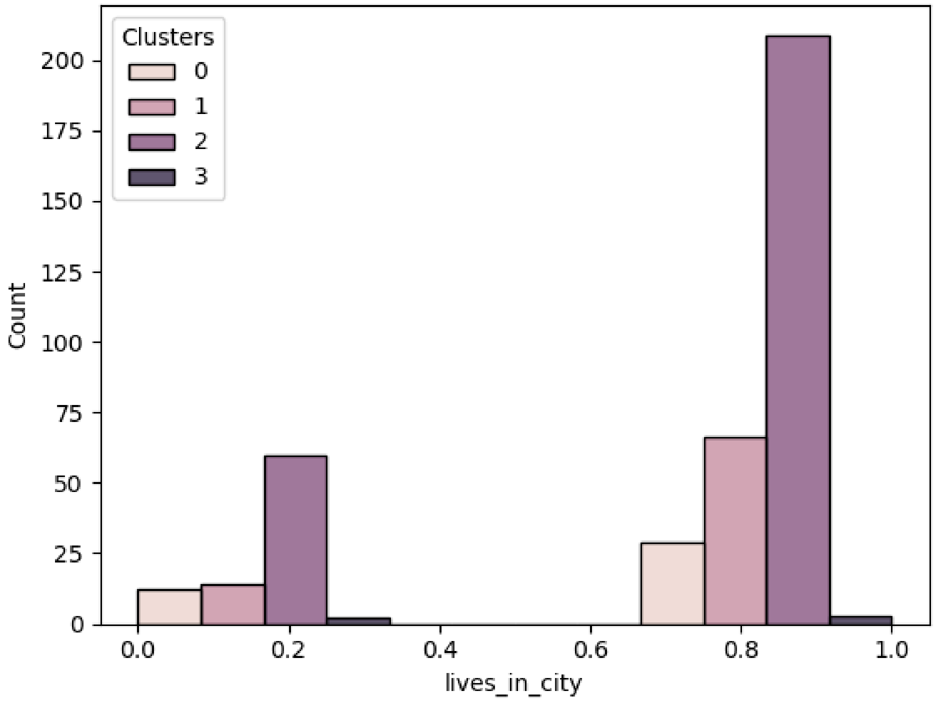
In [178]:
```
sns.histplot(x = temp_student['large_family'],
hue=temp_student['Clusters'], multiple="dodge", bins=4)
```

Out[178]: <Axes: xlabel='large_family', ylabel='Count'>



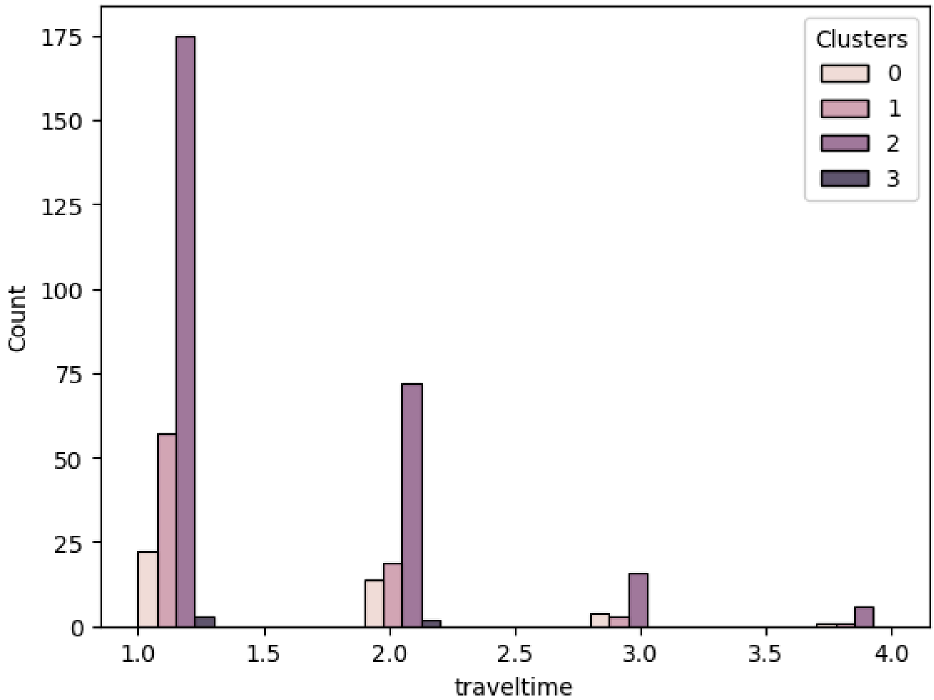
In [179]:
```
sns.histplot(x = temp_student['lives_in_city'],
hue=temp_student['Clusters'], multiple="dodge", bins=3)
```

Out[179]: <Axes: xlabel='lives_in_city', ylabel='Count'>



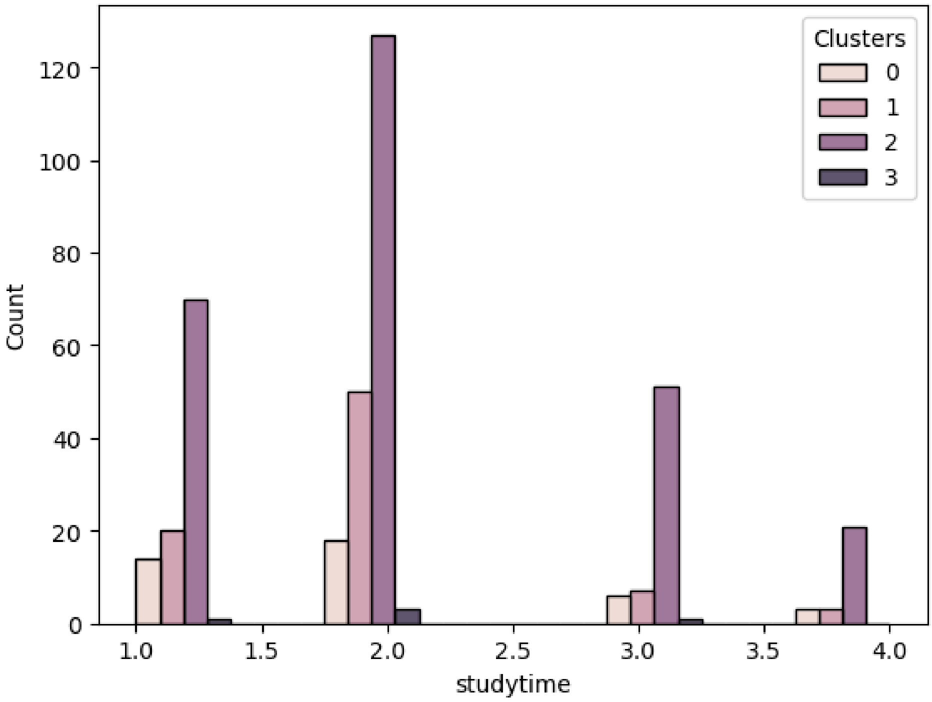
In [180]:
```
sns.histplot(x = temp_student['traveltime'],
hue=temp_student['Clusters'], multiple="dodge", bins=10)
```

Out[180]: <Axes: xlabel='traveltime', ylabel='Count'>



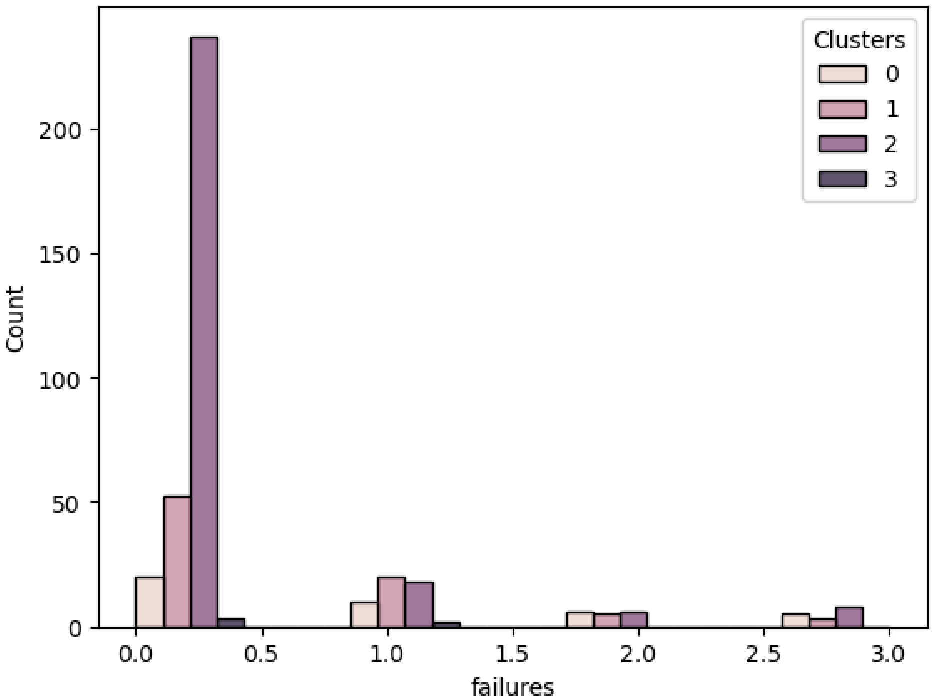
In [181]:
```
sns.histplot(x = temp_student['studytime'],
hue=temp_student['Clusters'], multiple="dodge", bins=8)
```

Out[181]: <Axes: xlabel='studytime', ylabel='Count'>



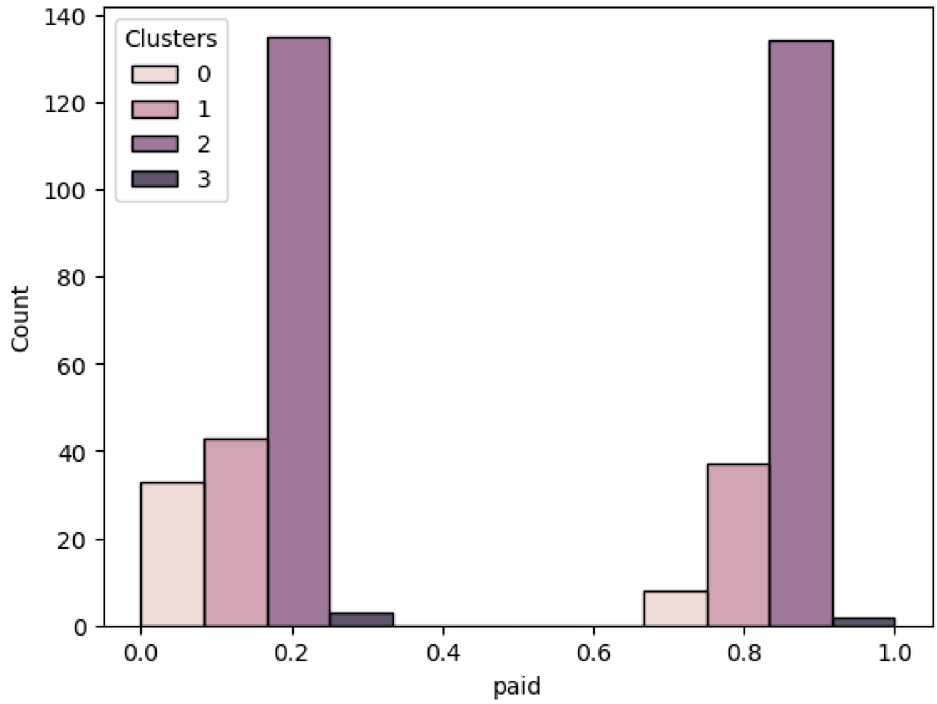
In [182]:
```
sns.histplot(x = temp_student['failures'], hue=temp_student['Clusters'],
multiple="dodge", bins=7)
```

Out[182]: <Axes: xlabel='failures', ylabel='Count'>



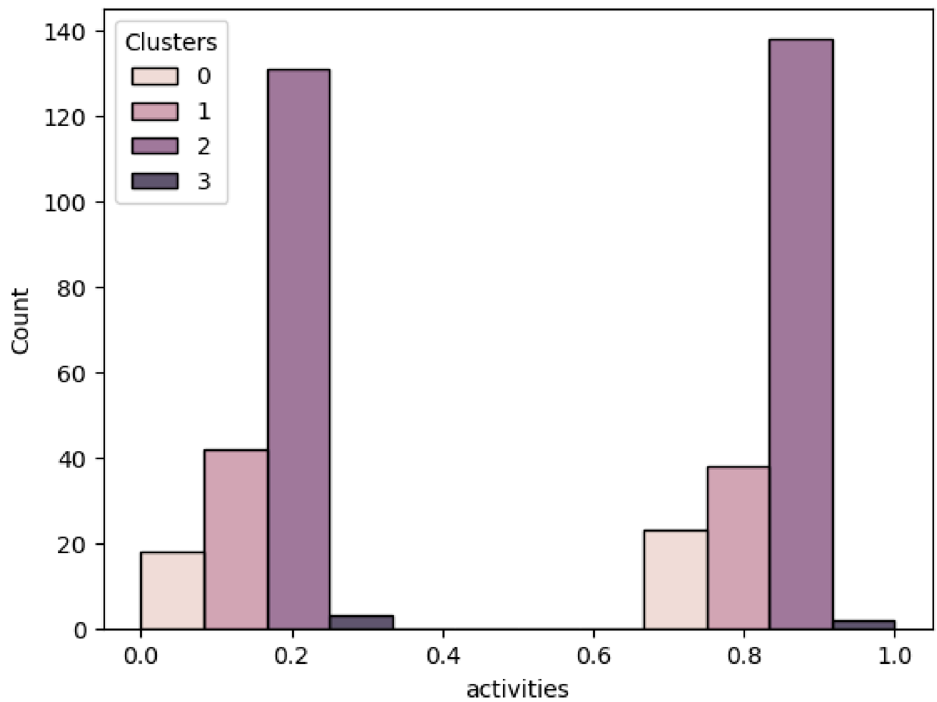
In [183]:
```
sns.histplot(x = temp_student['paid'], hue=temp_student['Clusters'],
multiple="dodge", bins=3)
```

Out[183]: <Axes: xlabel='paid', ylabel='Count'>



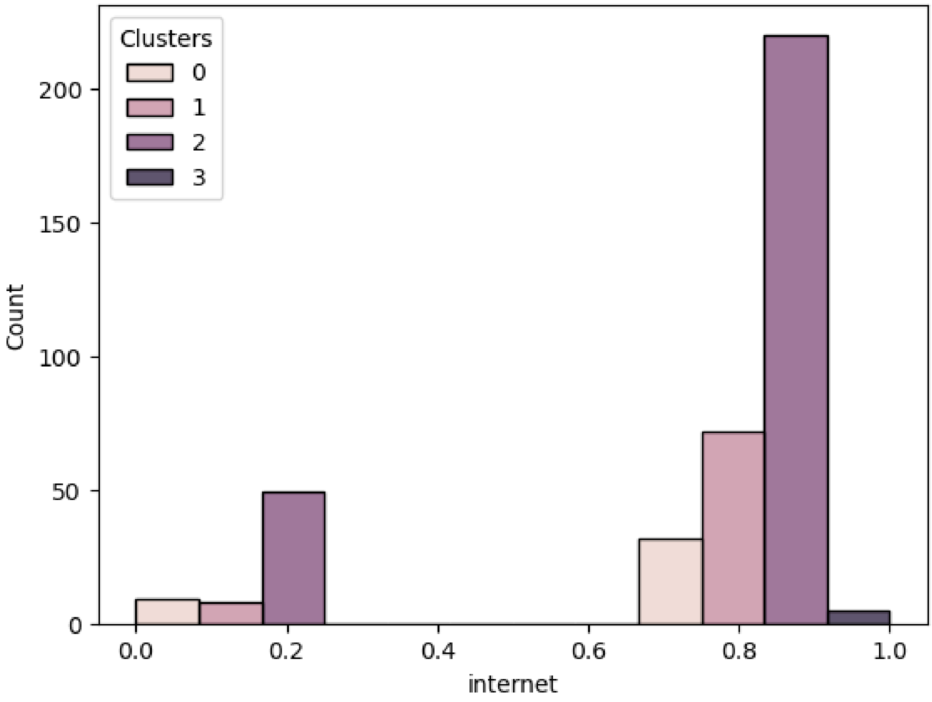
In [184]:
```
sns.histplot(x = temp_student['activities'],
hue=temp_student['Clusters'], multiple="dodge", bins=3)
```

Out[184]: <Axes: xlabel='activities', ylabel='Count'>



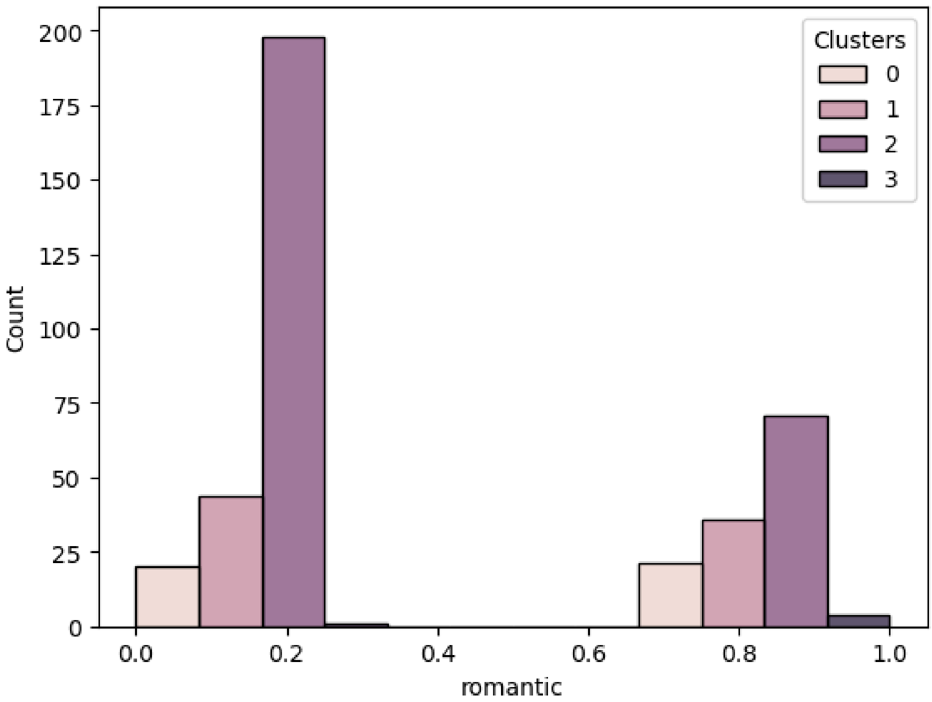
In [185]:
```
sns.histplot(x = temp_student['internet'], hue=temp_student['Clusters'],
multiple="dodge", bins=3)
```

Out[185]: <Axes: xlabel='internet', ylabel='Count'>



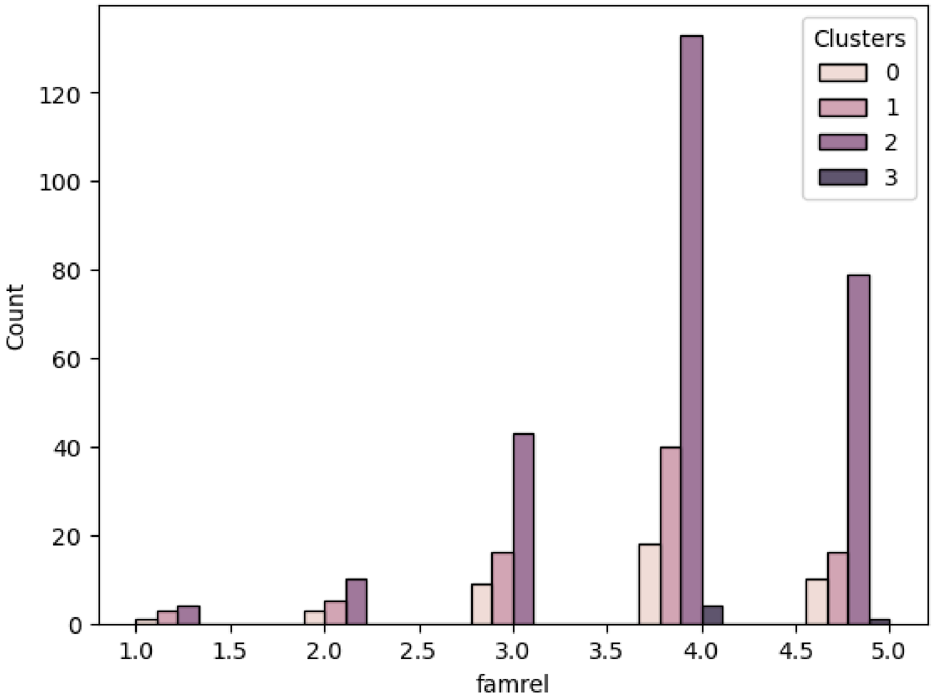
In [186]:
```
sns.histplot(x = temp_student['romantic'], hue=temp_student['Clusters'],
multiple="dodge", bins=3)
```

Out[186]: <Axes: xlabel='romantic', ylabel='Count'>



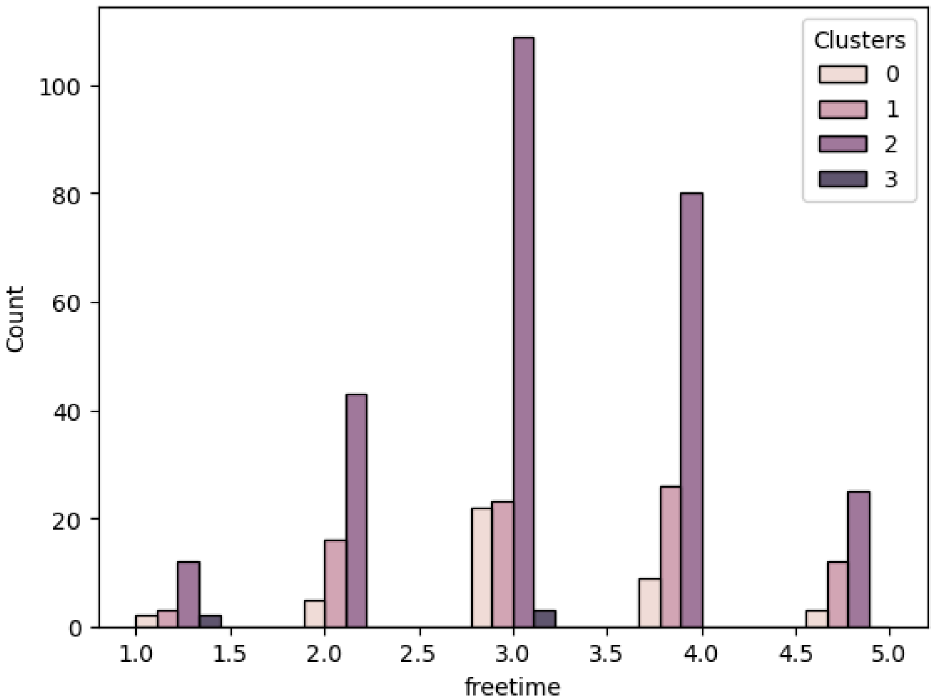
In [187]:
```
sns.histplot(x = temp_student['famrel'], hue=temp_student['Clusters'],
multiple="dodge", bins=9)
```

Out[187]: <Axes: xlabel='famrel', ylabel='Count'>



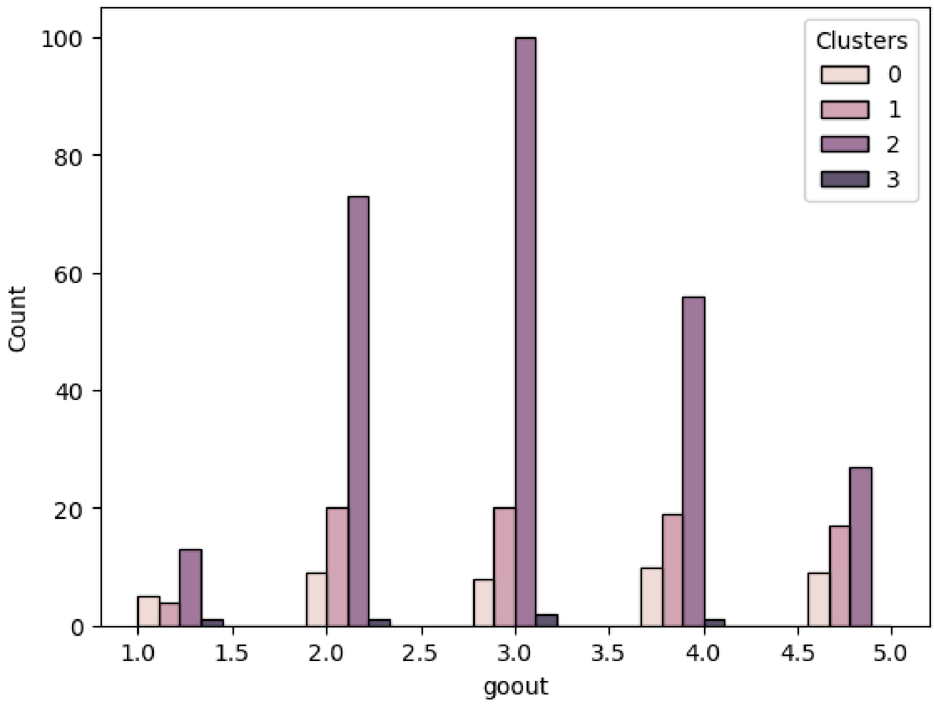
In [188]:
```
sns.histplot(x = temp_student['freetime'], hue=temp_student['Clusters'],
multiple="dodge", bins=9)
```

Out[188]: <Axes: xlabel='freetime', ylabel='Count'>



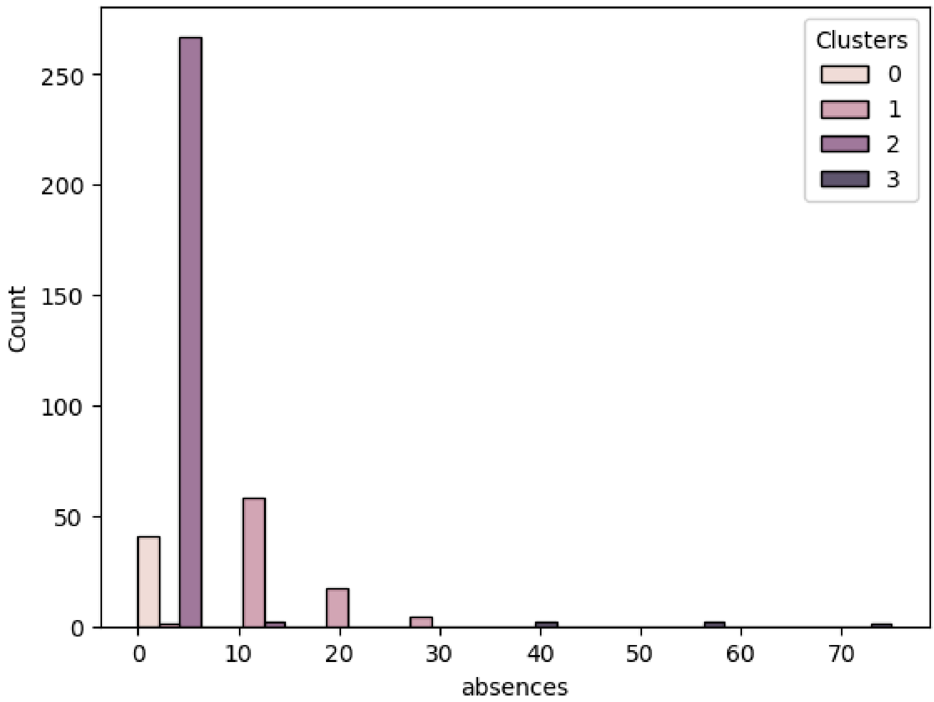
In [189]:
```
sns.histplot(x = temp_student['goout'], hue=temp_student['Clusters'],
multiple="dodge", bins=9)
```

Out[189]: <Axes: xlabel='goout', ylabel='Count'>



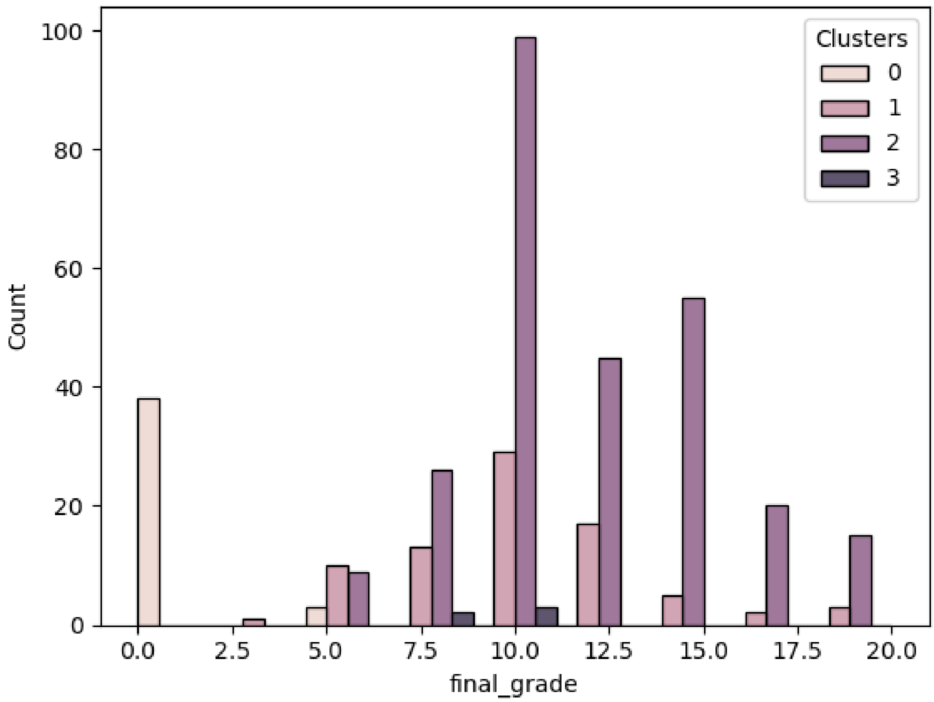
In [190]:
```
sns.histplot(x = temp_student['absences'], hue=temp_student['Clusters'],
multiple="dodge", bins=9)
```

Out[190]: <Axes: xlabel='absences', ylabel='Count'>



In [191]:
```
sns.histplot(x = temp_student['final_grade'],
hue=temp_student['Clusters'], multiple="dodge", bins=9)
```

Out[191]: <Axes: xlabel='final_grade', ylabel='Count'>



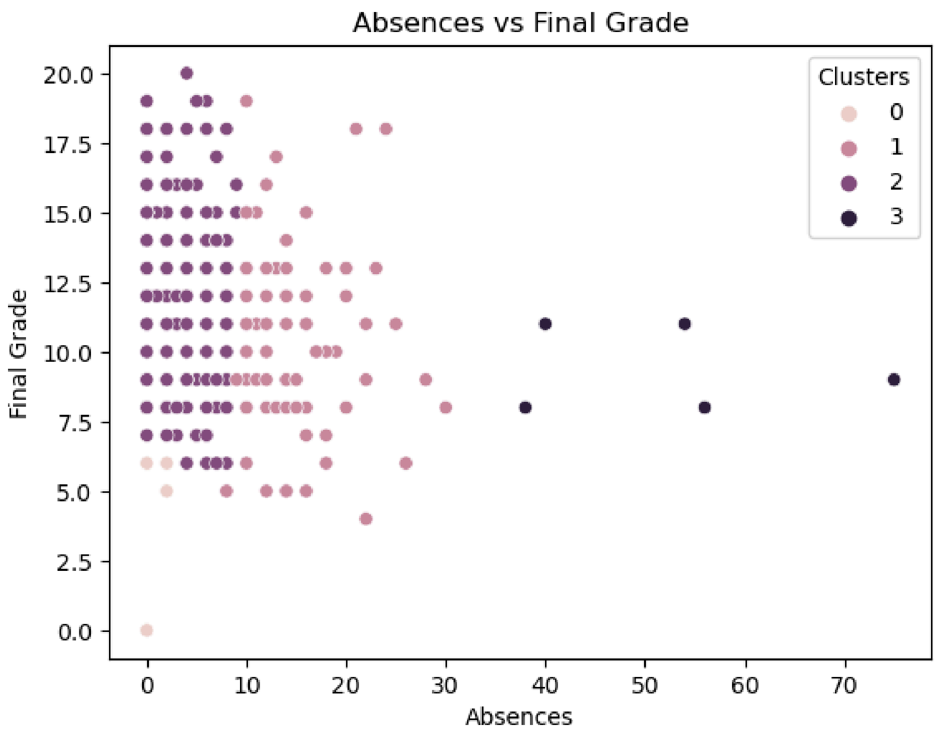
Discriminatory - final_grade, absences, traveltime, failures(maybe)

c) Scatter plots

In [192]:

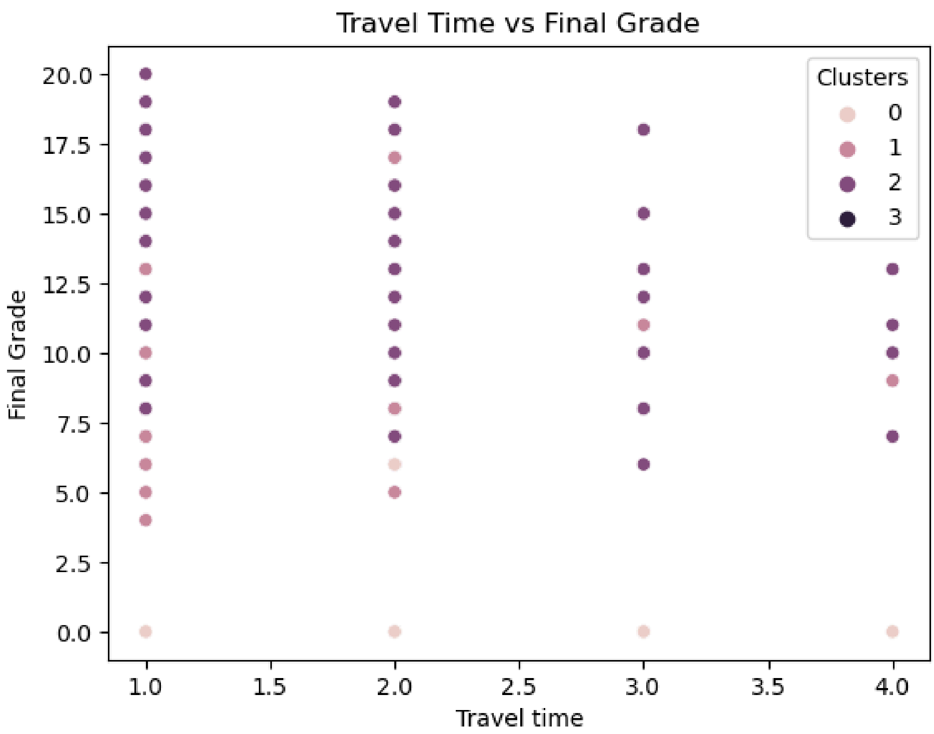
```
#Scatter plot fpr absences and final grade
sns.scatterplot(x = temp_student['absences'], y =
temp_student['final_grade'], hue = temp_student['Clusters'])
plt.xlabel('Absences')
plt.ylabel('Final Grade')
plt.title('Absences vs Final Grade')
```

Out[192]: `Text(0.5, 1.0, 'Absences vs Final Grade')`



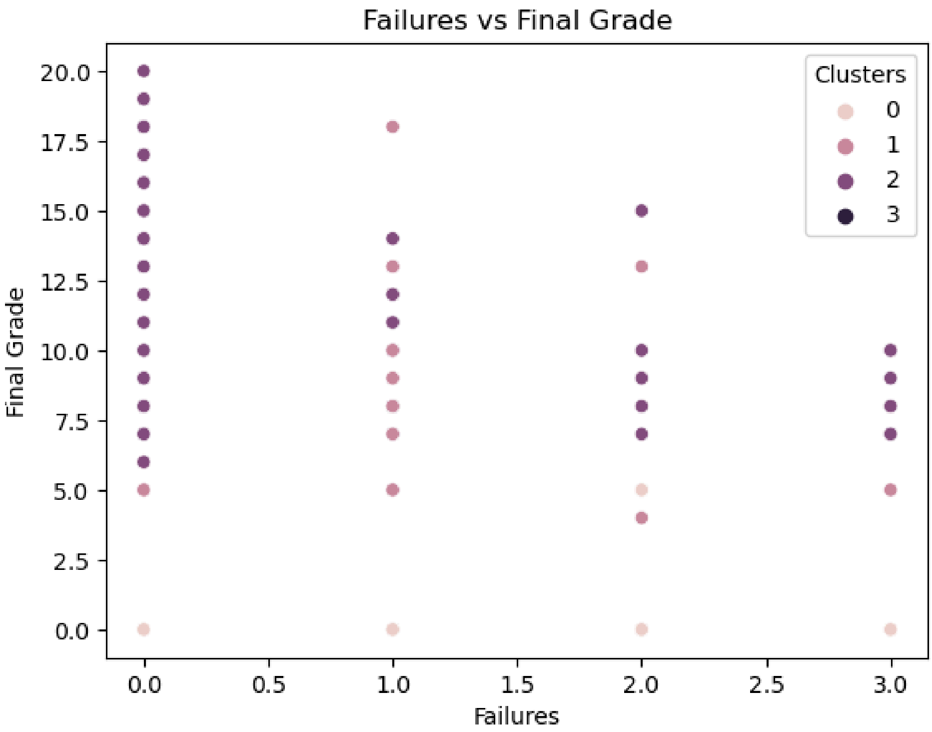
In [193]:
```
#Scatter plot for travel time and final grade
sns.scatterplot(x = temp_student['traveltime'], y =
temp_student['final_grade'], hue = temp_student['Clusters'])
plt.xlabel('Travel time')
plt.ylabel('Final Grade')
plt.title('Travel Time vs Final Grade')
```

Out[193]: `Text(0.5, 1.0, 'Travel Time vs Final Grade')`



In [194]:
```
#Scatter plot for failures and final grade
sns.scatterplot(x = temp_student['failures'], y =
temp_student['final_grade'], hue = temp_student['Clusters'])
plt.xlabel('Failures')
plt.ylabel('Final Grade')
plt.title('Failures vs Final Grade')
```

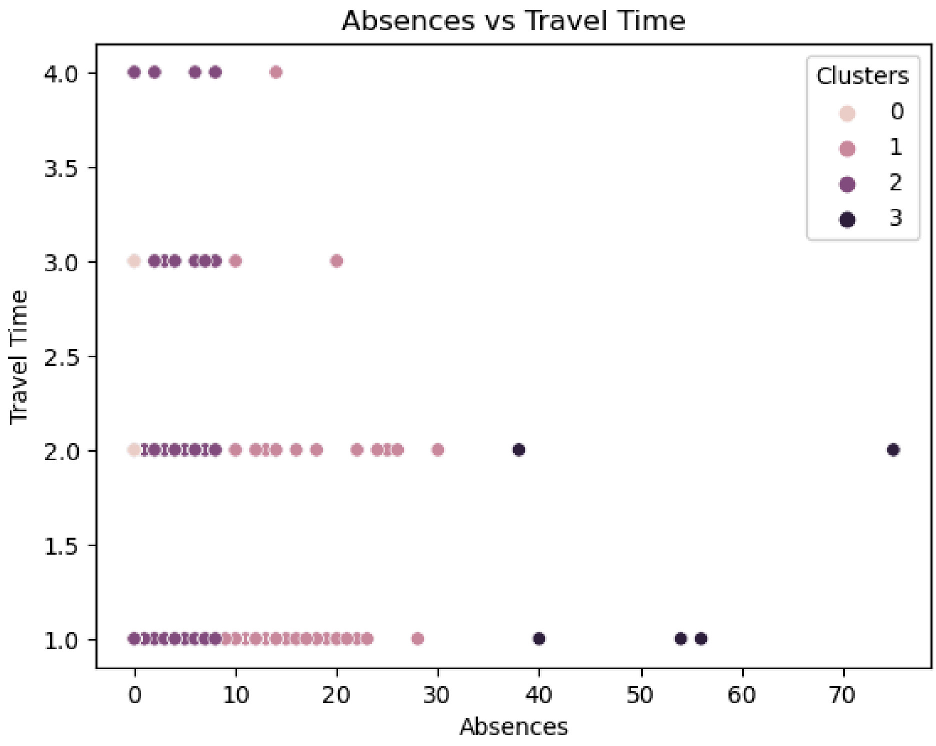
Out[194]: `Text(0.5, 1.0, 'Failures vs Final Grade')`



In [195]:

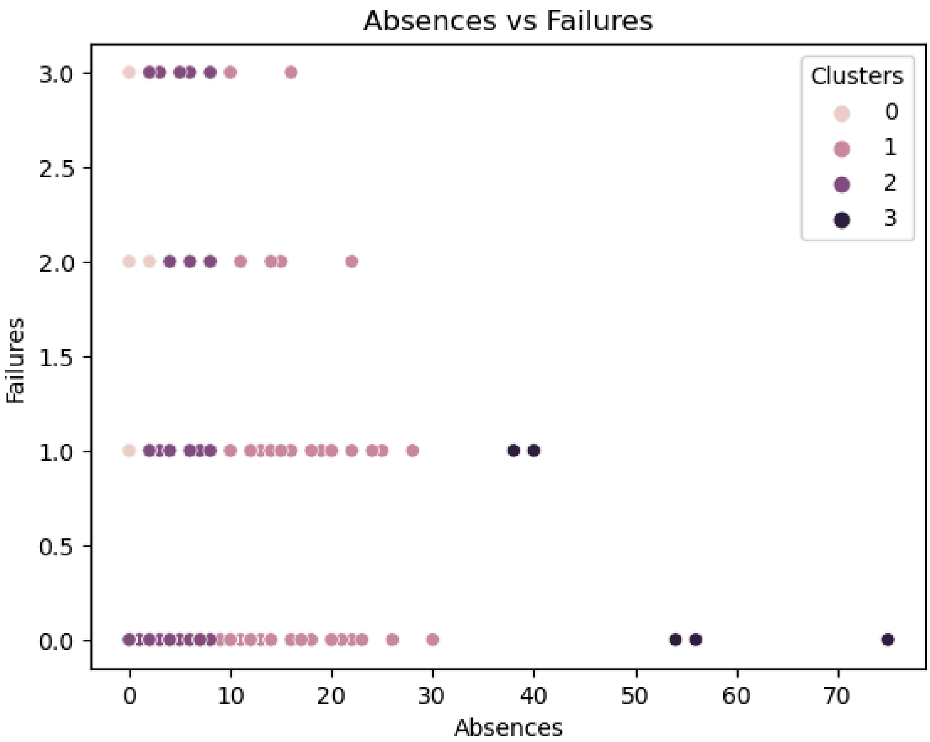
```
#Scatter plot for absences and travel time
sns.scatterplot(x = temp_student['absences'], y =
temp_student['traveltime'], hue = temp_student['Clusters'])
plt.xlabel('Absences')
plt.ylabel('Travel Time')
plt.title('Absences vs Travel Time')
```

Out[195]: Text(0.5, 1.0, 'Absences vs Travel Time')



In [196]:
```
#Scatter plot for absences and failures
sns.scatterplot(x = temp_student['absences'], y =
temp_student['failures'], hue = temp_student['Clusters'])
plt.xlabel('Absences')
plt.ylabel('Failures')
plt.title('Absences vs Failures')
```

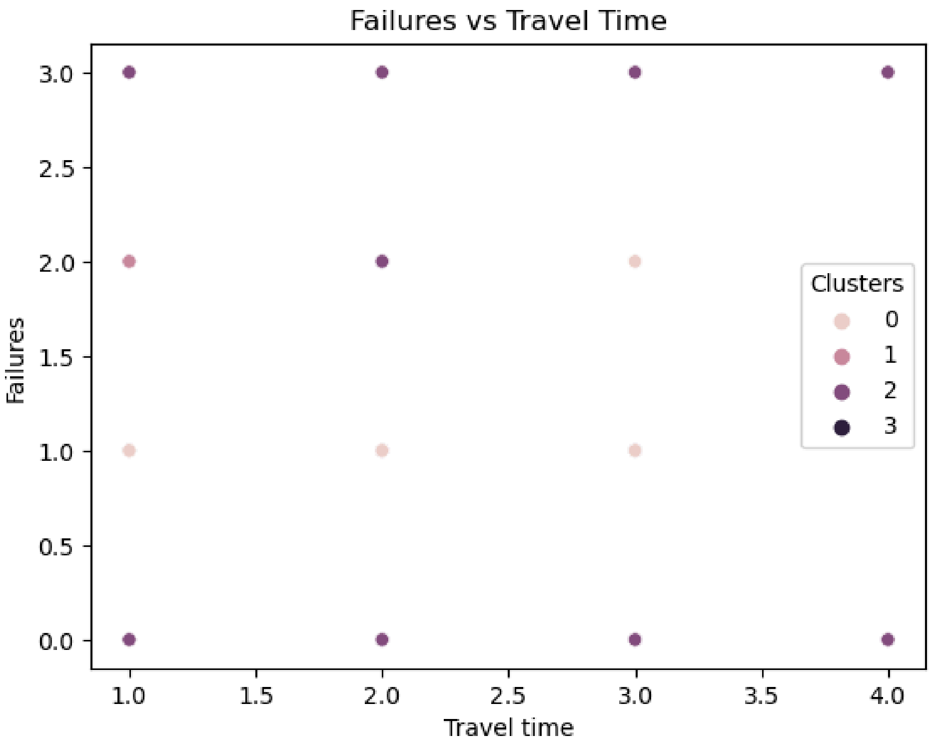
Out[196]: `Text(0.5, 1.0, 'Absences vs Failures')`



In [197]:

```
#Scatter plot for travel time and failures
sns.scatterplot(x = temp_student['traveltime'], y =
temp_student['failures'], hue = temp_student['Clusters'])
plt.xlabel('Travel time')
plt.ylabel('Failures')
plt.title('Failures vs Travel Time')
```

Out[197]: `Text(0.5, 1.0, 'Failures vs Travel Time')`



From the scatter plots it can be seen that the clusters are being created for certain ranges of values of the variables. In some cases it is more evident, like in "Absences vs Failures", "Absences vs Travel Time" and "Absences vs Final Grade". From the plots "Absences" seem to be the most discriminatory variable for the clustering. This means that there are students with some disinct characteristics which can be mapped to different classes.

d) Different clustering algorithm - DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that categorises the data into clusters based on their density. It does not require a cluster count to be given. It can also find outliers and classify it into noise instead of it being in a cluster. It can find clusters of arbitrary shapes unlike in K-means.

It has main two parameters, **Epsilon** and **Minimum points**. Epsilon is the maximum distance to be considered between two neighbors and Minimum points is the minimum number of points required to form a dense region.

This algorithm is best used when the clusters are irregularly shaped and there are outliers in the dataset

In [200]:

```
#Import the model
from sklearn.cluster import DBSCAN
```

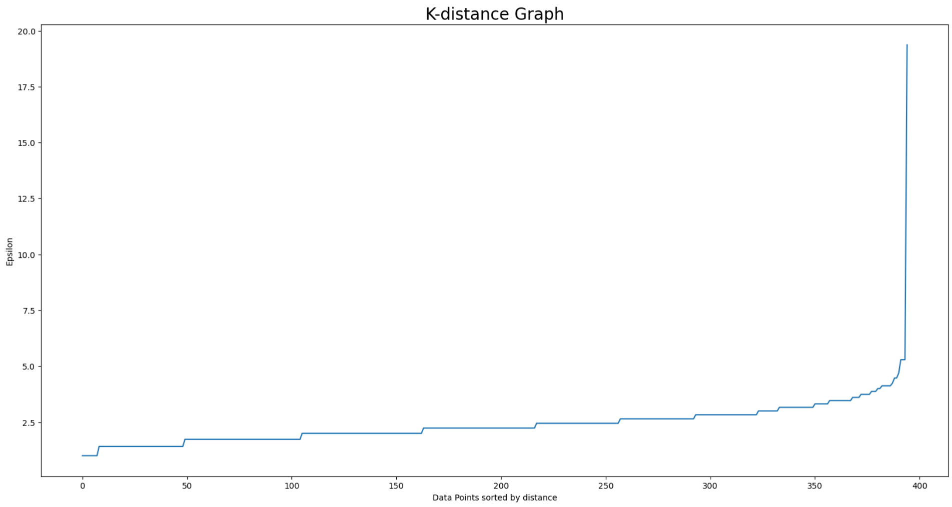
We plot a k-distance to find the optimal Epsilon value. The optimal value of Epsilon is the point where there is maximum curvature in the graph.

In [201]:

```
#Fit nearest neighbors
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(k_student)
distances, indices = nbrs.kneighbors(k_student)
```

In [202]:

```
# Plot K-distance Graph
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(20,10))
plt.plot(distances)
plt.title('K-distance Graph',fontsize=20)
plt.xlabel('Data Points sorted by distance')
plt.ylabel('Epsilon')
plt.show()
```



The optimal Epsilon value is 3

In [203]:

```
#Fit DBSCAN model
dbscan=DBSCAN(eps=3, min_samples=6)
dbscan.fit(k_student)
dbscan.labels_
labels = dbscan.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```

```
Estimated number of clusters: 3
Estimated number of noise points: 87
```

Plot histograms

In [204]:

```
pred_df1 = pd.DataFrame()
pred_df1['Clusters'] = labels
temp_student1 = pd.concat([k_student, pred_df1], axis='columns')
#Removing noise
temp_student1 = temp_student1[temp_student1['Clusters'] != -1]
temp_student1
```

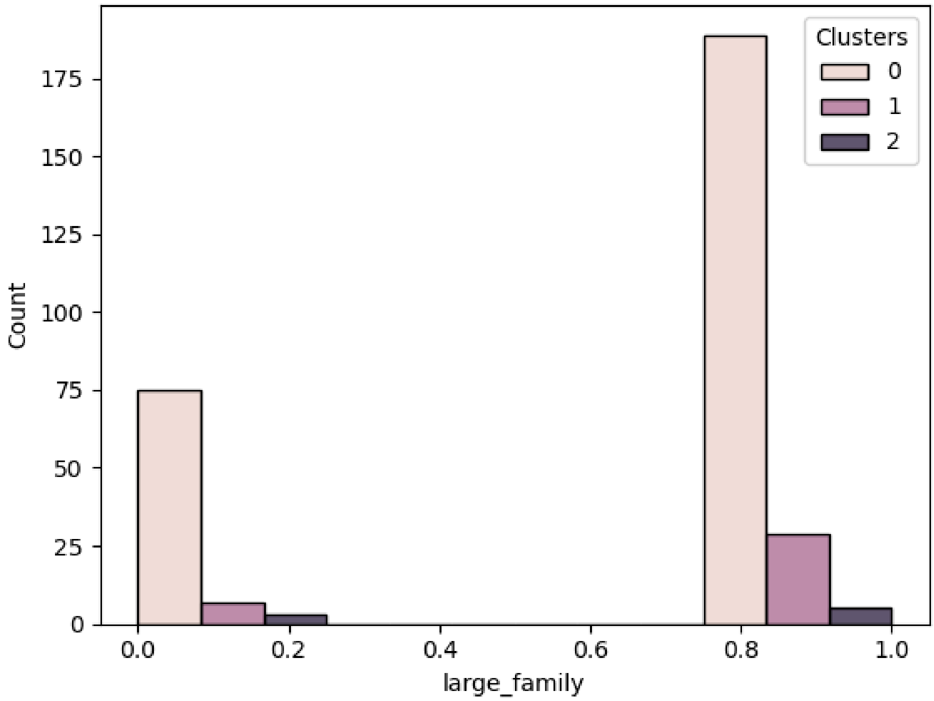
Out[204]:

	large_family	lives_in_city	traveltime	studytime	failures	pai
0	0	1	1	2	0	1
1	0	1	1	2	0	0
3	1	1	1	2	0	1
4	0	1	1	1	0	1
5	1	1	2	2	0	1
...	...	...	...	...	...	...
388	1	1	1	2	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

308 rows × 15 columns

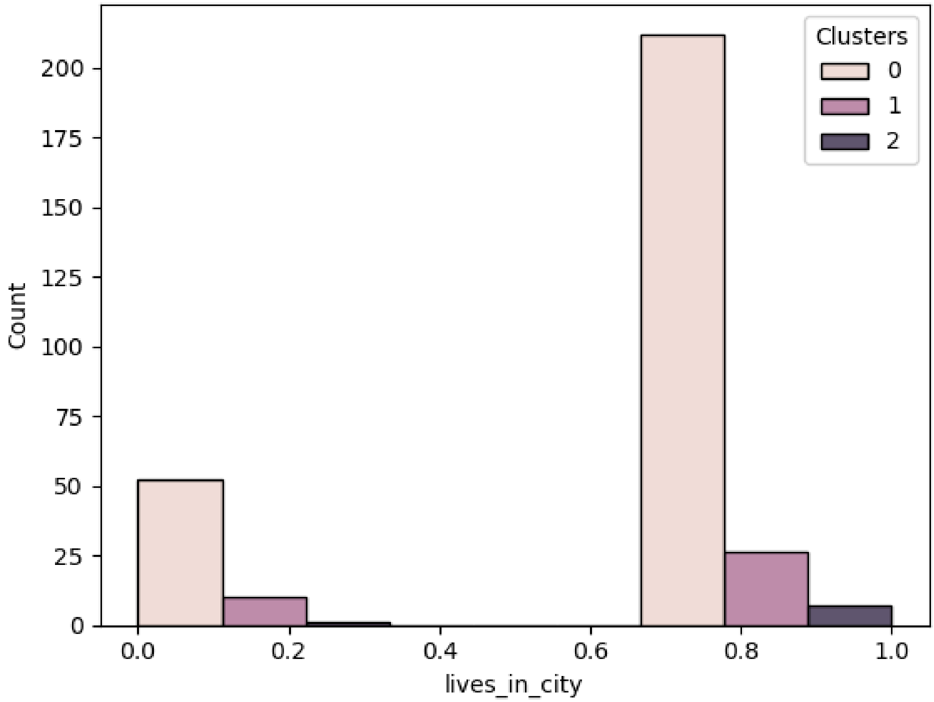
In [205]:
```
sns.histplot(x = temp_student1['large_family'],
hue=temp_student1['Clusters'], multiple="dodge", bins=4)
```

Out[205]: <Axes: xlabel='large_family', ylabel='Count'>



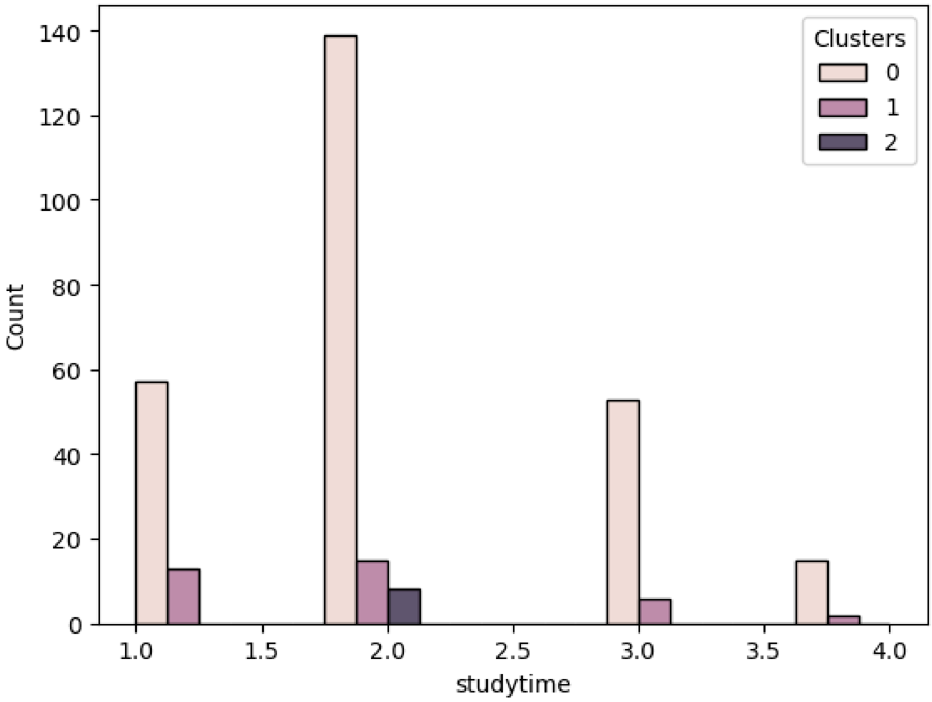
In [206]:
```
sns.histplot(x = temp_student1['lives_in_city'],
hue=temp_student1['Clusters'], multiple="dodge", bins=3)
```

Out[206]: <Axes: xlabel='lives_in_city', ylabel='Count'>



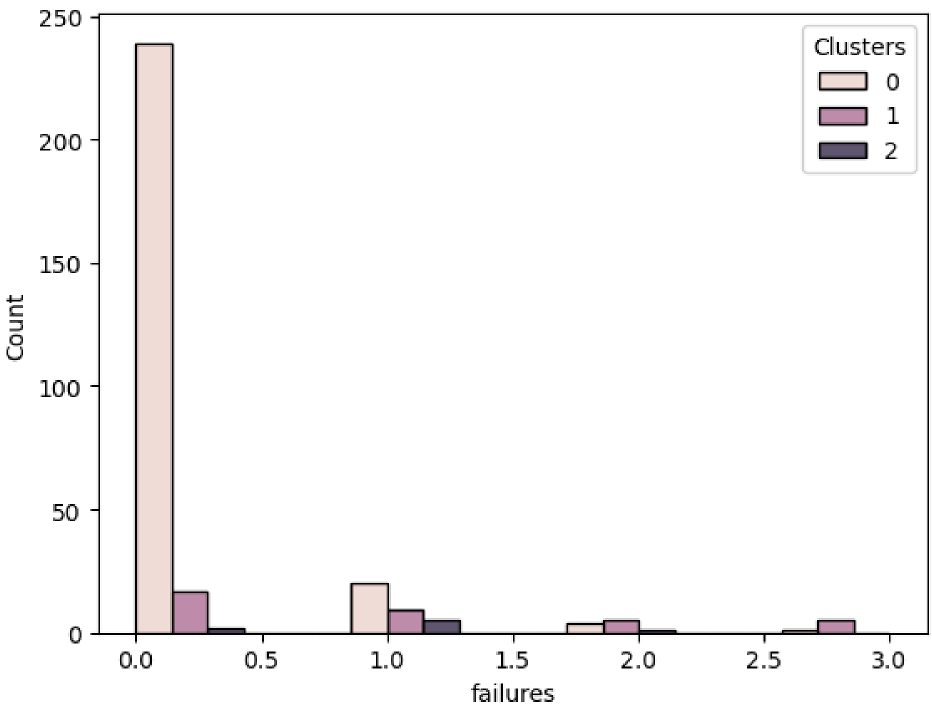
In [207]:
```
sns.histplot(x = temp_student1['studytime'],
hue=temp_student1['Clusters'], multiple="dodge", bins=8)
```

Out[207]: <Axes: xlabel='studytime', ylabel='Count'>



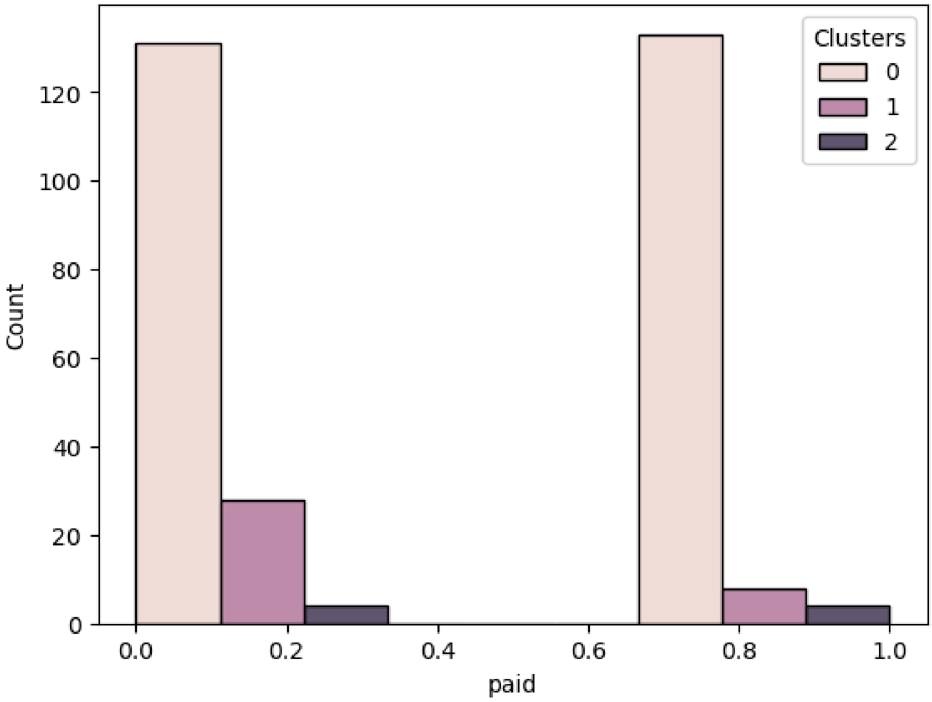
In [208]:
```
sns.histplot(x = temp_student1['failures'],
hue=temp_student1['Clusters'], multiple="dodge", bins=7)
```

Out[208]: <Axes: xlabel='failures', ylabel='Count'>



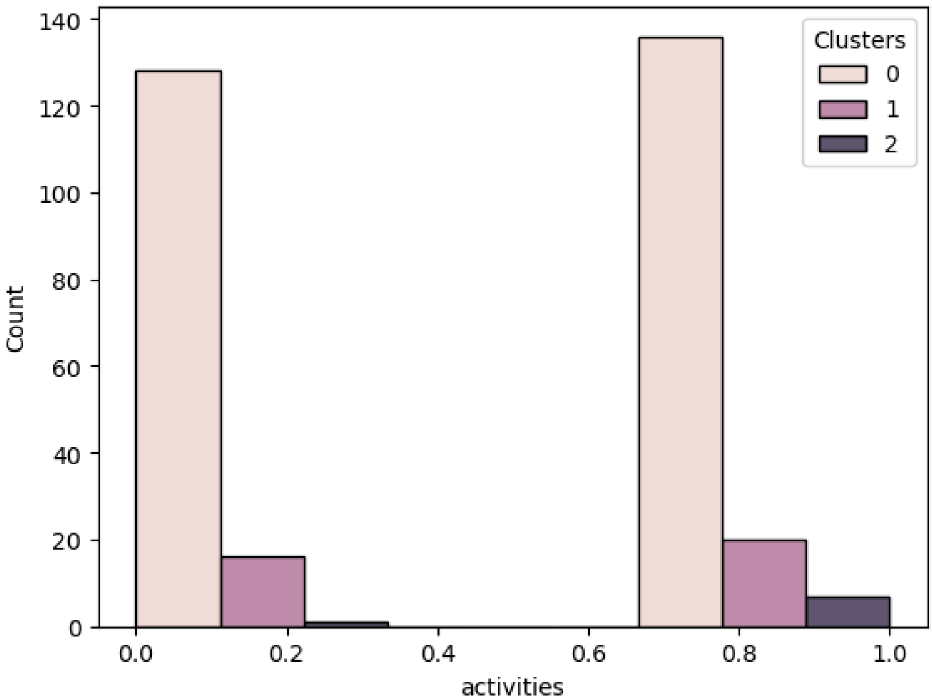
In [209]:
```
sns.histplot(x = temp_student1['paid'], hue=temp_student1['Clusters'],
multiple="dodge", bins=3)
```

Out[209]: <Axes: xlabel='paid', ylabel='Count'>



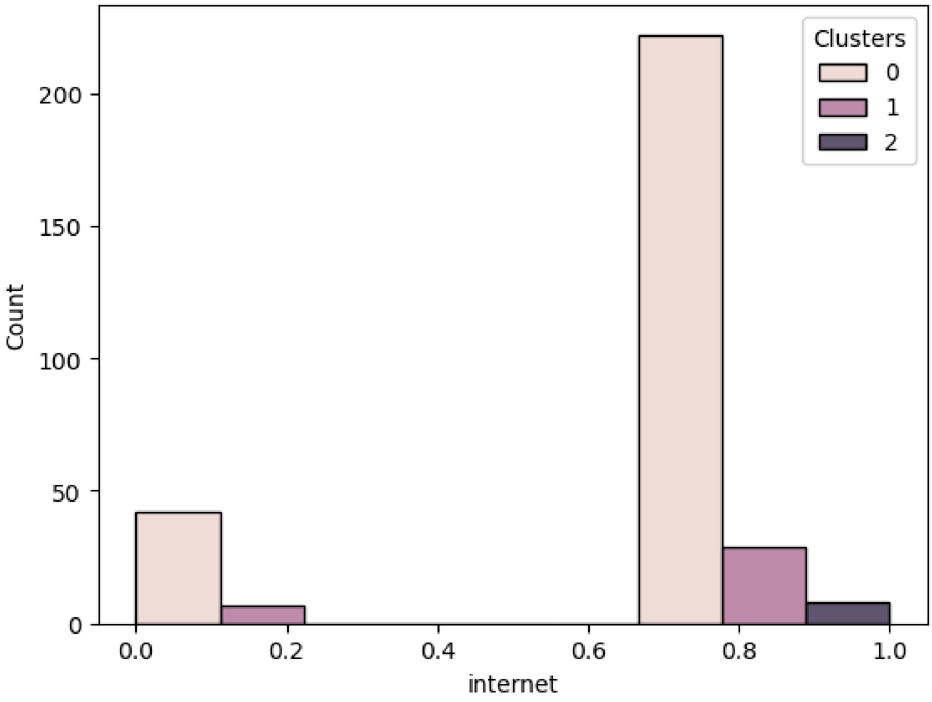
In [210]:
```
sns.histplot(x = temp_student1['activities'],
hue=temp_student1['Clusters'], multiple="dodge", bins=3)
```

Out[210]: <Axes: xlabel='activities', ylabel='Count'>



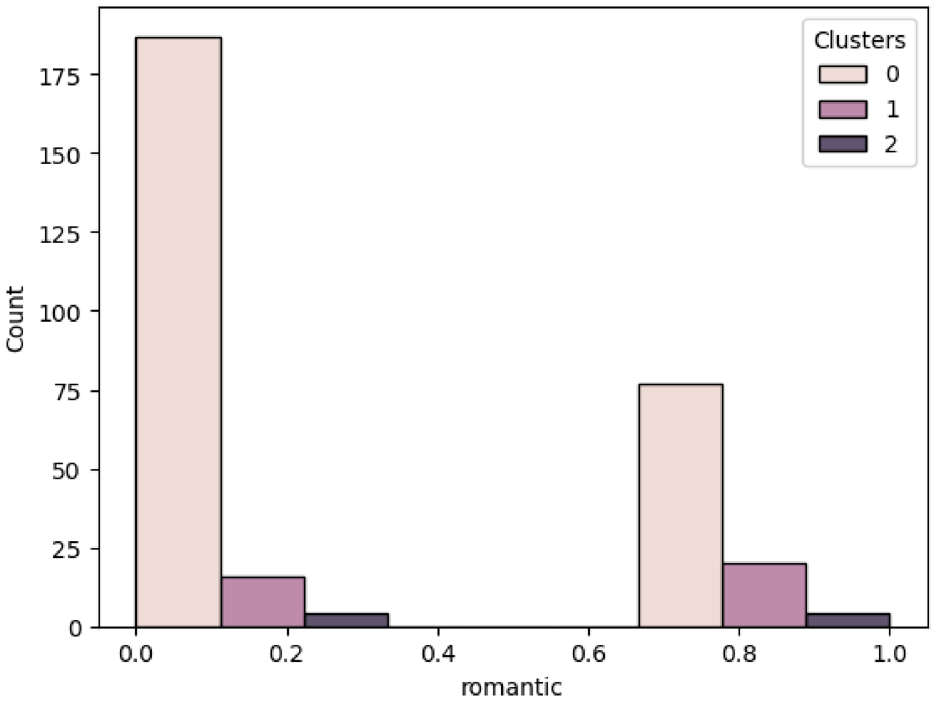
In [211]:
```
sns.histplot(x = temp_student1['internet'],
hue=temp_student1['Clusters'], multiple="dodge", bins=3)
```

Out[211]: <Axes: xlabel='internet', ylabel='Count'>



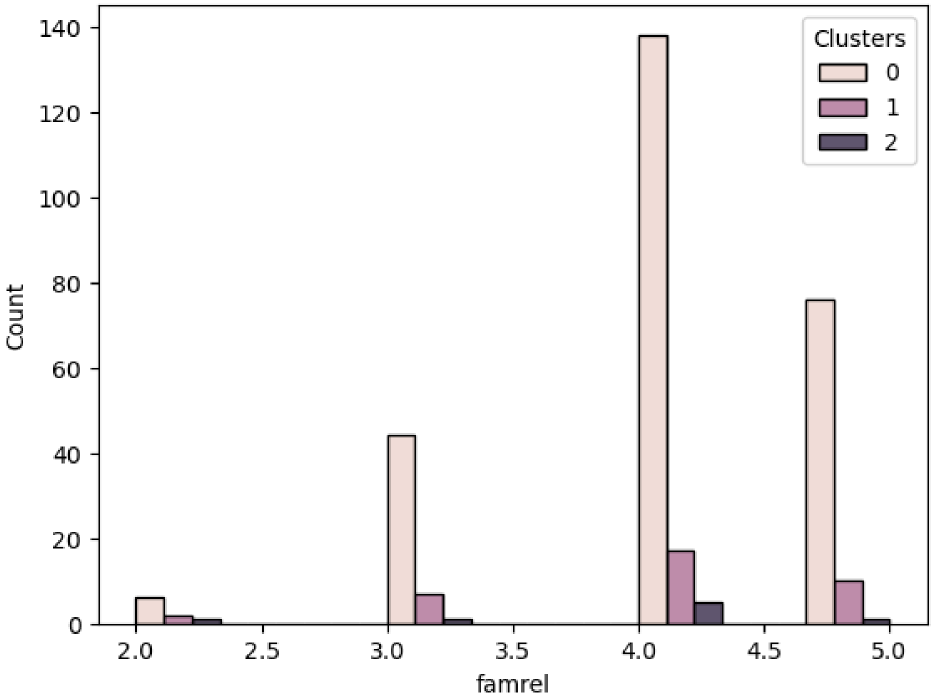
In [212]:
```
sns.histplot(x = temp_student1['romantic'],
hue=temp_student1['Clusters'], multiple="dodge", bins=3)
```

Out[212]: <Axes: xlabel='romantic', ylabel='Count'>



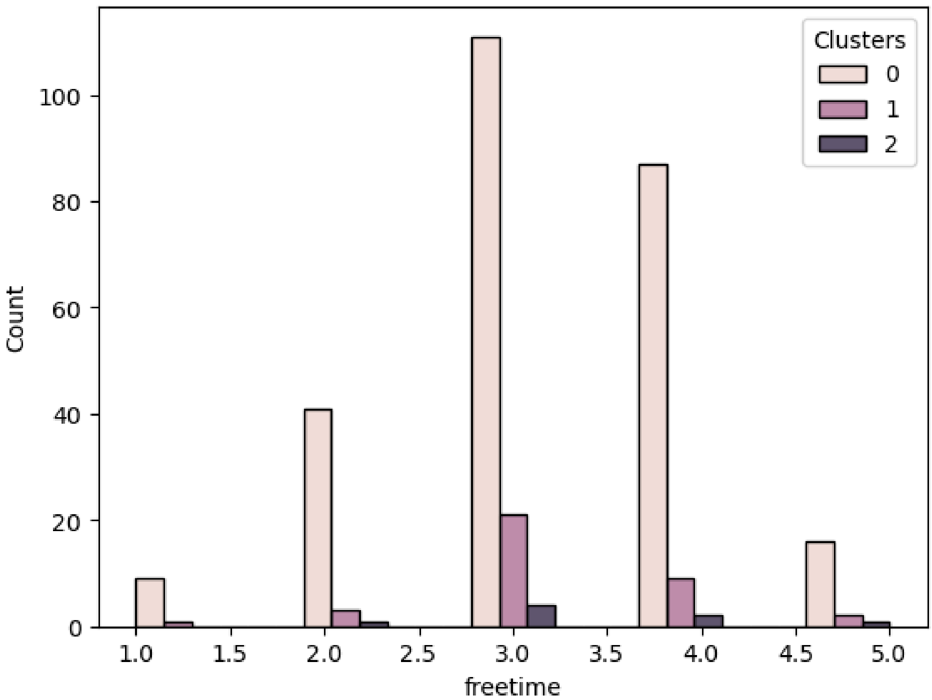
In [213]:
```
sns.histplot(x = temp_student1['famrel'], hue=temp_student1['Clusters'], multiple="dodge", bins=9)
```

Out[213]: <Axes: xlabel='famrel', ylabel='Count'>



In [214]:
```
sns.histplot(x = temp_student1['freetime'], hue=temp_student1['Clusters'], multiple="dodge", bins=9)
```

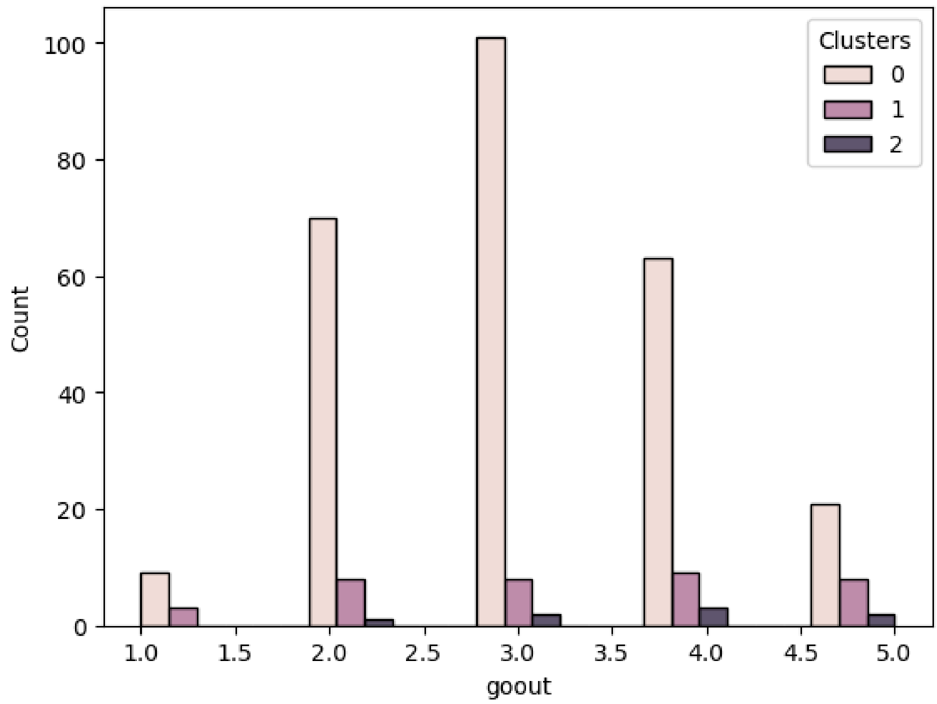
Out[214]: <Axes: xlabel='freetime', ylabel='Count'>



In [215]:

```
sns.histplot(x = temp_student1['goout'], hue=temp_student1['Clusters'],
multiple="dodge", bins=9)
```

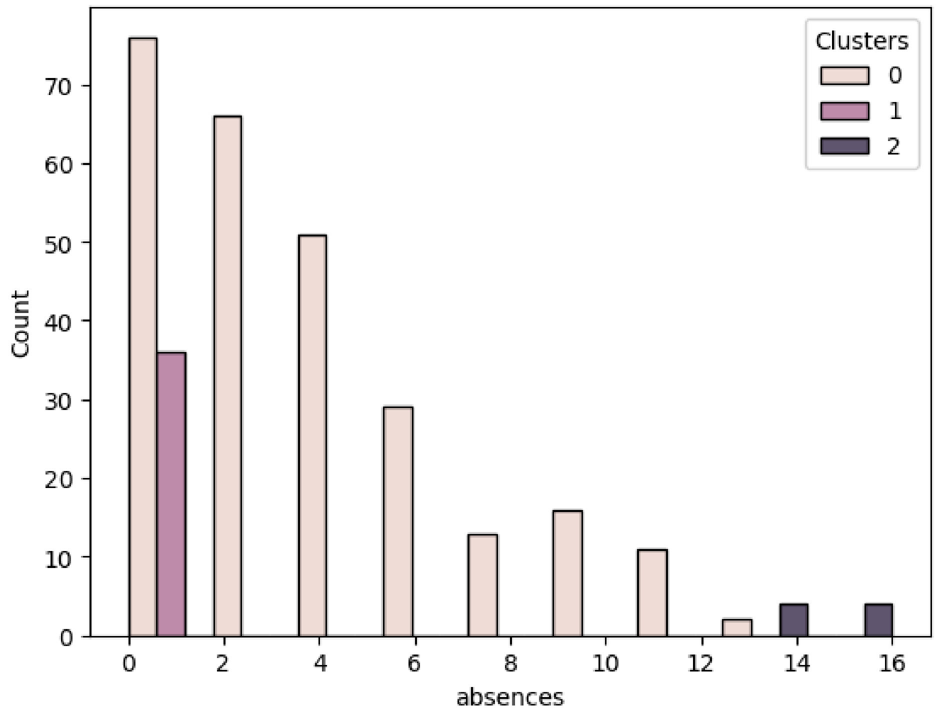
Out[215]: <Axes: xlabel='goout', ylabel='Count'>



In [216]:

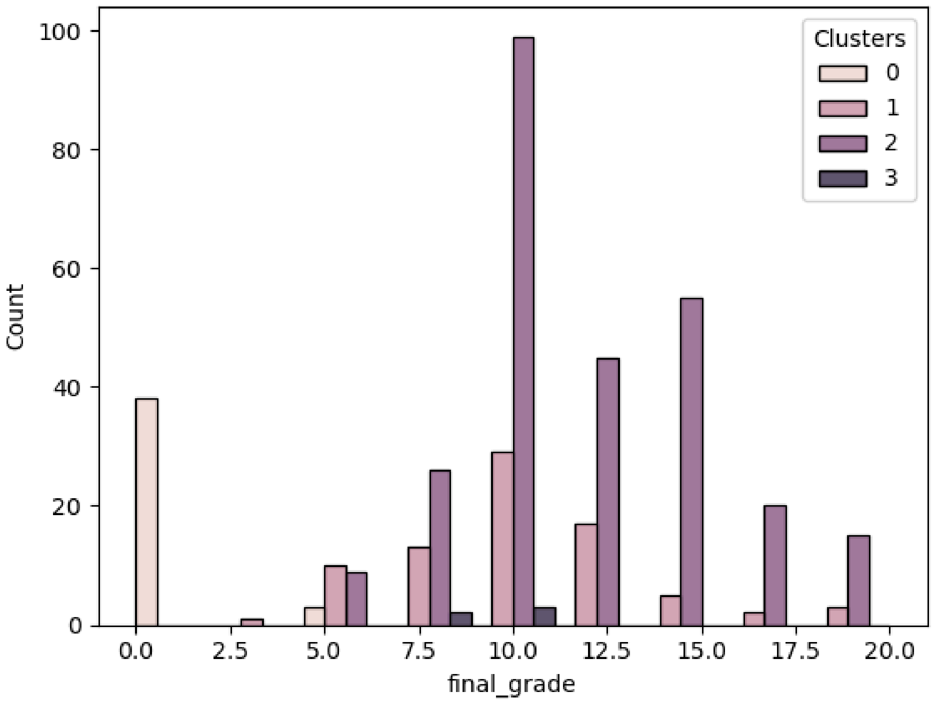
```
sns.histplot(x = temp_student1['absences'],
hue=temp_student1['Clusters'], multiple="dodge", bins=9)
```

Out[216]: <Axes: xlabel='absences', ylabel='Count'>



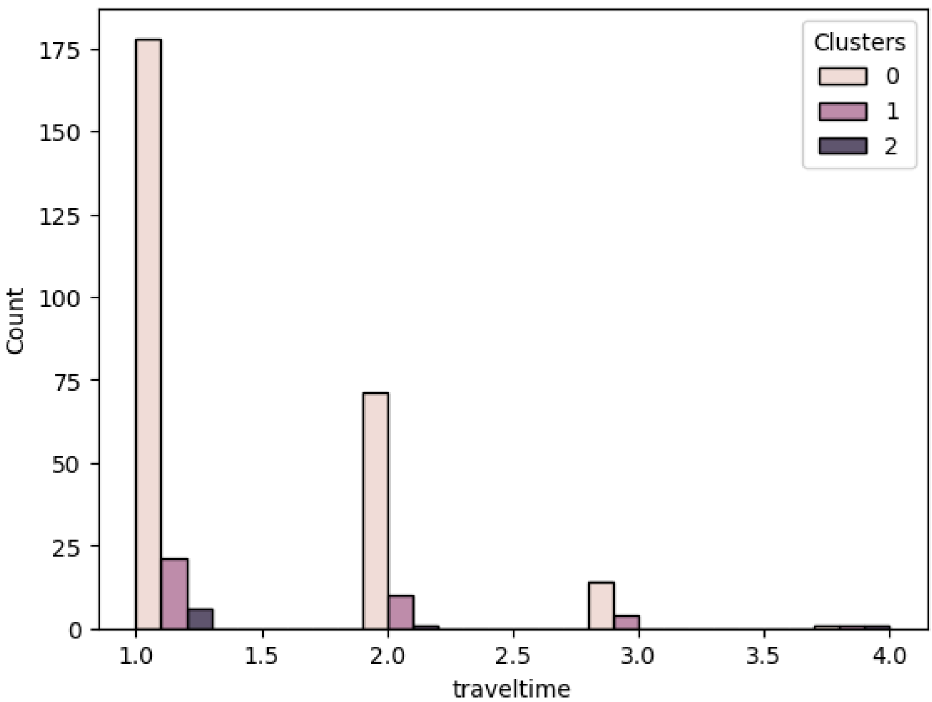
In [217]:
```
sns.histplot(x = temp_student['final_grade'],
hue=temp_student['Clusters'], multiple="dodge", bins=9)
```

Out[217]: <Axes: xlabel='final_grade', ylabel='Count'>



In [218]:
```
sns.histplot(x = temp_student1['traveltime'],
hue=temp_student1['Clusters'], multiple="dodge", bins=10)
```

Out[218]: <Axes: xlabel='traveltime', ylabel='Count'>

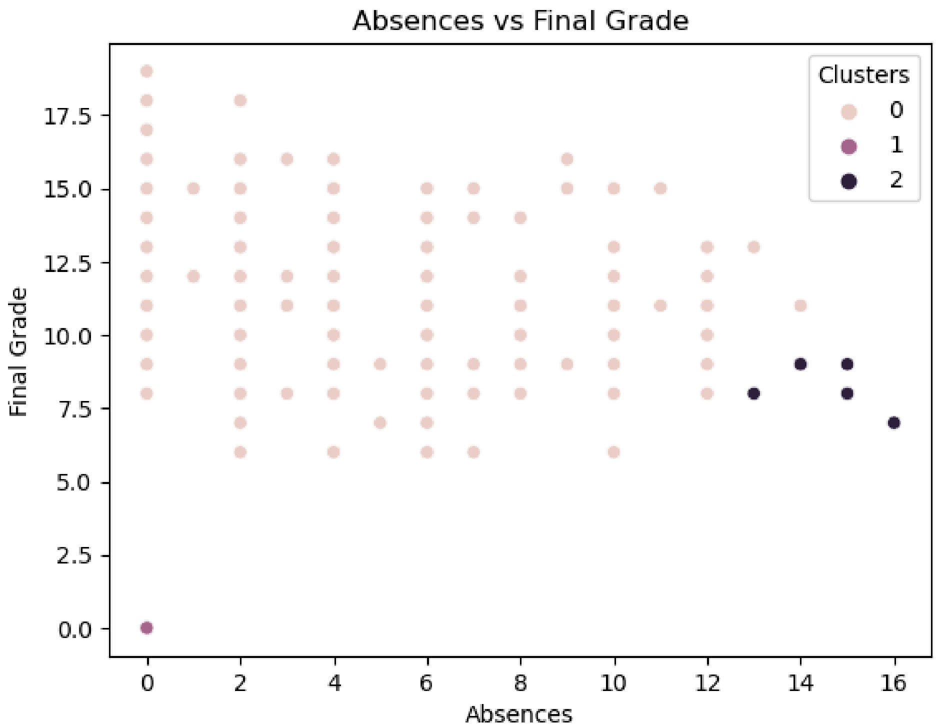


The Discriminatory variables are:
- final_grade
- absences
- traveltime

Scatter plots

In [219]:
```
#Scatter plot fpr absences and final grade
sns.scatterplot(x = temp_student1['absences'], y =
temp_student1['final_grade'], hue = temp_student1['Clusters'])
plt.xlabel('Absences')
plt.ylabel('Final Grade')
plt.title('Absences vs Final Grade')
```

Out[219]: `Text(0.5, 1.0, 'Absences vs Final Grade')`

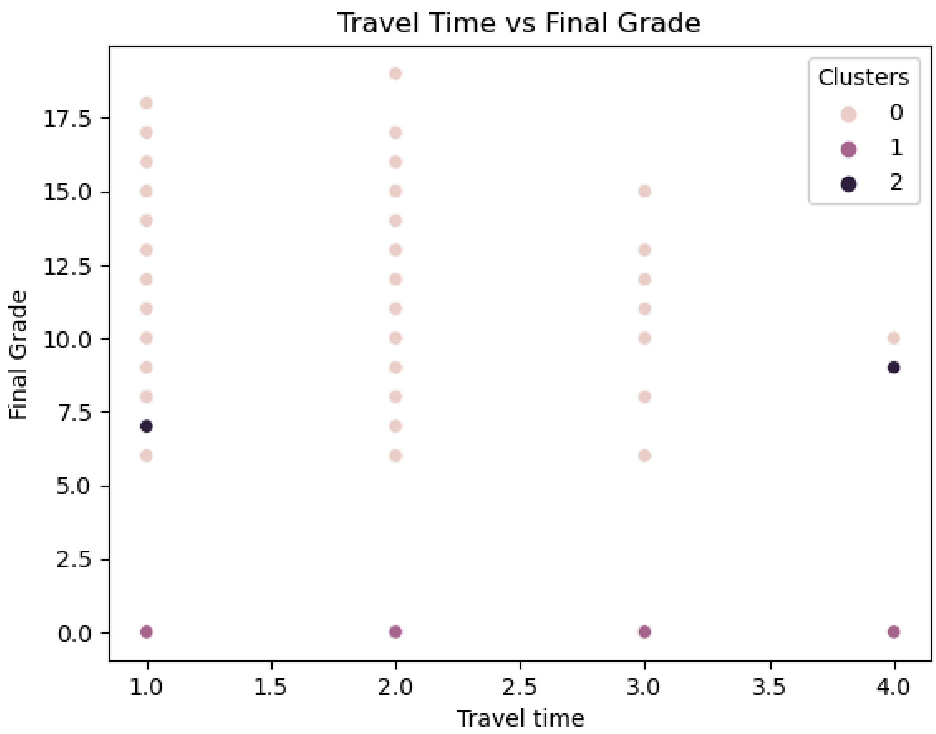


In [220]:

```
#Scatter plot for travel time and final grade
sns.scatterplot(x = temp_student1['traveltime'], y =
temp_student1['final_grade'], hue = temp_student1['Clusters'])
plt.xlabel('Travel time')
plt.ylabel('Final Grade')
plt.title('Travel Time vs Final Grade')
```

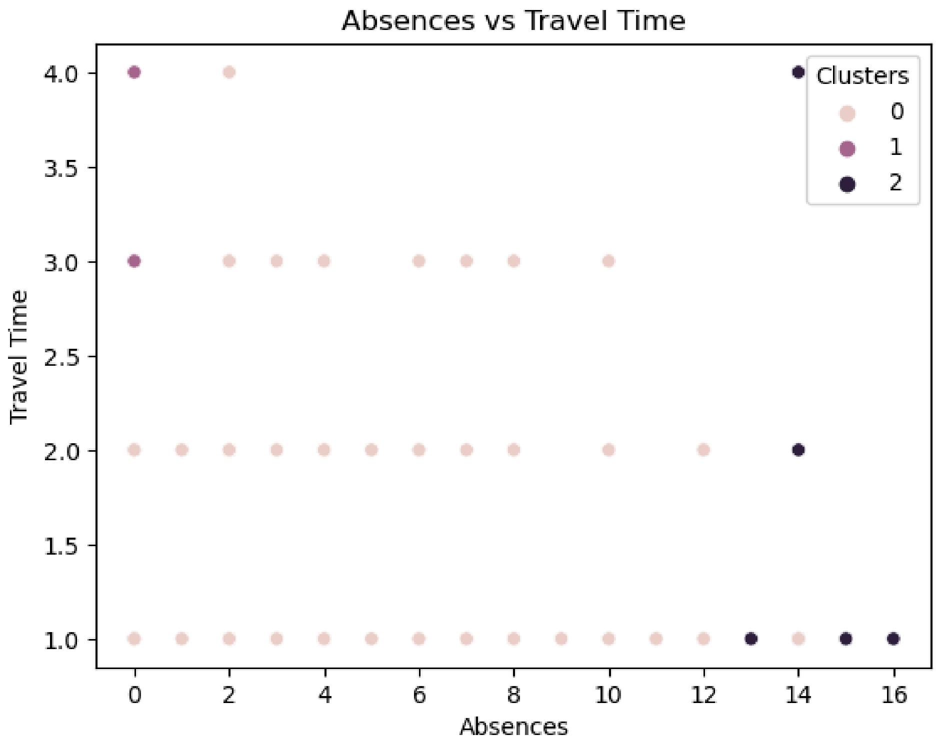
Out[220]:

```
Text(0.5, 1.0, 'Travel Time vs Final Grade')
```



In [221]:
```
#Scatter plot for absences and travel time
sns.scatterplot(x = temp_student1['absences'], y =
temp_student1['traveltime'], hue = temp_student1['Clusters'])
plt.xlabel('Absences')
plt.ylabel('Travel Time')
plt.title('Absences vs Travel Time')
```

Out[221]: `Text(0.5, 1.0, 'Absences vs Travel Time')`



For DBSCAN, the cluster with most number of points it cluster 0. It does not look like there is any particular pattern for the clusters and for this dataset, this algorithm might not have been a good choice.
Looking at the plots for DBSCAN and K-means, it looks like K-Means was able to better categorise the students.

Honour Code

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