```
#importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
```

```
#file read

df = pd.read_csv("/content/student_prediction.csv")
```

```
df.head()
```

	STUDENTID	AGE	GENDER	HS_TYPE	SCHOLARSHIP	WORK	ACTIVITY	PARTNER	SALARY	TRANSPORT	...	PREP_
0	STUDENT1	2	2	3	3	1	2	2	1	1	...	
1	STUDENT2	2	2	3	3	1	2	2	1	1	...	
2	STUDENT3	2	2	2	3	2	2	2	2	4	...	
3	STUDENT4	1	1	1	3	1	2	1	2	1	...	
4	STUDENT5	2	2	1	3	2	2	1	3	1	...	

5 rows × 33 columns

```
df.rename(columns={'KIDS':'PARENTAL_STATUS'},inplace=True)
```

```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
AGE	145.0	1.620690	0.613154	1.0	1.0	2.0	2.0	3.0
GENDER	145.0	1.600000	0.491596	1.0	1.0	2.0	2.0	2.0
HS_TYPE	145.0	1.944828	0.537216	1.0	2.0	2.0	2.0	3.0
SCHOLARSHIP	145.0	3.572414	0.805750	1.0	3.0	3.0	4.0	5.0
WORK	145.0	1.662069	0.474644	1.0	1.0	2.0	2.0	2.0
ACTIVITY	145.0	1.600000	0.491596	1.0	1.0	2.0	2.0	2.0
PARTNER	145.0	1.579310	0.495381	1.0	1.0	2.0	2.0	2.0
SALARY	145.0	1.627586	1.020245	1.0	1.0	1.0	2.0	5.0
TRANSPORT	145.0	1.620690	1.061112	1.0	1.0	1.0	2.0	4.0
MOTHER_EDU	145.0	2.282759	1.223062	1.0	1.0	2.0	3.0	6.0
FATHER_JOB	145.0	2.806897	1.329664	1.0	2.0	3.0	4.0	5.0
ATTEND	145.0	1.241379	0.429403	1.0	1.0	1.0	1.0	2.0
PREP_EXAM	145.0	1.165517	0.408483	1.0	1.0	1.0	1.0	3.0
LISTENS	145.0	2.055172	0.674736	1.0	2.0	2.0	3.0	3.0

Taking Sample for train and test

```
X= df.drop(['GRADE','STUDENTID'],axis=1)
y= df['GRADE']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=42)
```

Logistic Regression

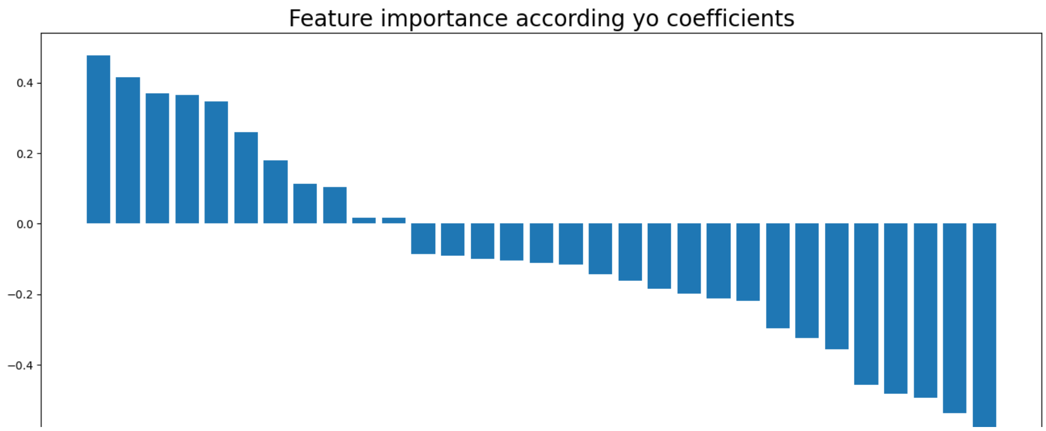
```
from sklearn.linear_model import LogisticRegression

model= LogisticRegression(solver='liblinear')
model.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression(solver='liblinear')
```

Checking The factors

```
importances= pd.DataFrame(data={'Attribute':X_train.columns,'Importance':model.coef_[0]})
importances= importances.sort_values(by='Importance',ascending=False)
```

```
plt.figure(figsize=(16,8))
plt.bar(x=importances['Attribute'],height=importances['Importance'])
plt.title('Feature importance according yo coefficients',size=20)
plt.xticks(rotation='vertical')
plt.show()
```



Making New Dataframe selecting the important columns

```
new_df=df[['MOTHER_JOB','FATHER_JOB','SALARY','PARENTAL_STATUS','COURSE ID','IMPACT','GRADE']]
```

```
new_df.head()
```

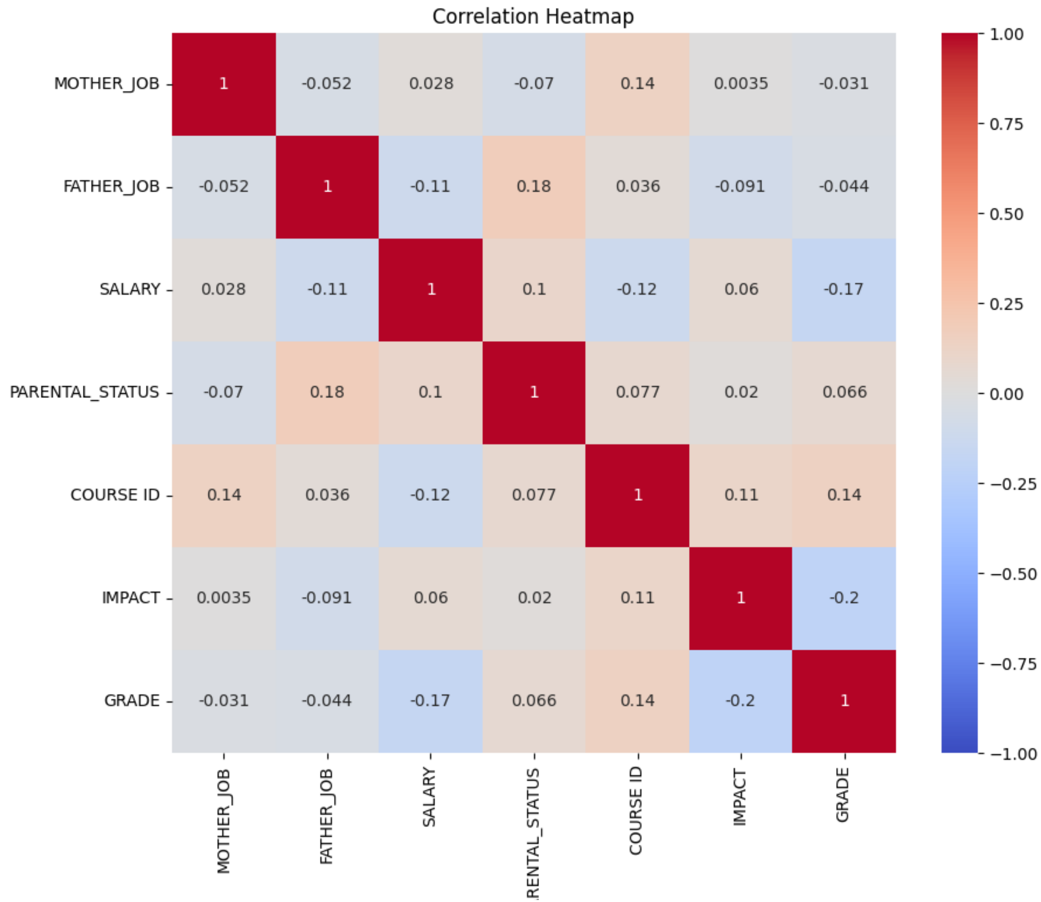
	MOTHER_JOB	FATHER_JOB	SALARY	PARENTAL_STATUS	COURSE ID	IMPACT	GRADE
0	2	5	1	1	1	1	1
1	2	1	1	1	1	1	1
2	2	1	2	1	1	1	1
3	2	1	2	1	1	1	1
4	2	4	3	1	1	1	1

```
new_df.describe()
```

	MOTHER_JOB	FATHER_JOB	SALARY	PARENTAL_STATUS	COURSE ID	IMPACT	GRADE
count	145.000000	145.000000	145.000000	145.000000	145.000000	145.000000	145.000000
mean	2.358621	2.806897	1.627586	1.172414	4.131034	1.206897	3.227586
std	0.805156	1.329664	1.020245	0.490816	3.260145	0.588035	2.197678
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	2.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000
50%	2.000000	3.000000	1.000000	1.000000	3.000000	1.000000	3.000000
75%	2.000000	4.000000	2.000000	1.000000	7.000000	1.000000	5.000000
max	5.000000	5.000000	5.000000	3.000000	9.000000	3.000000	7.000000

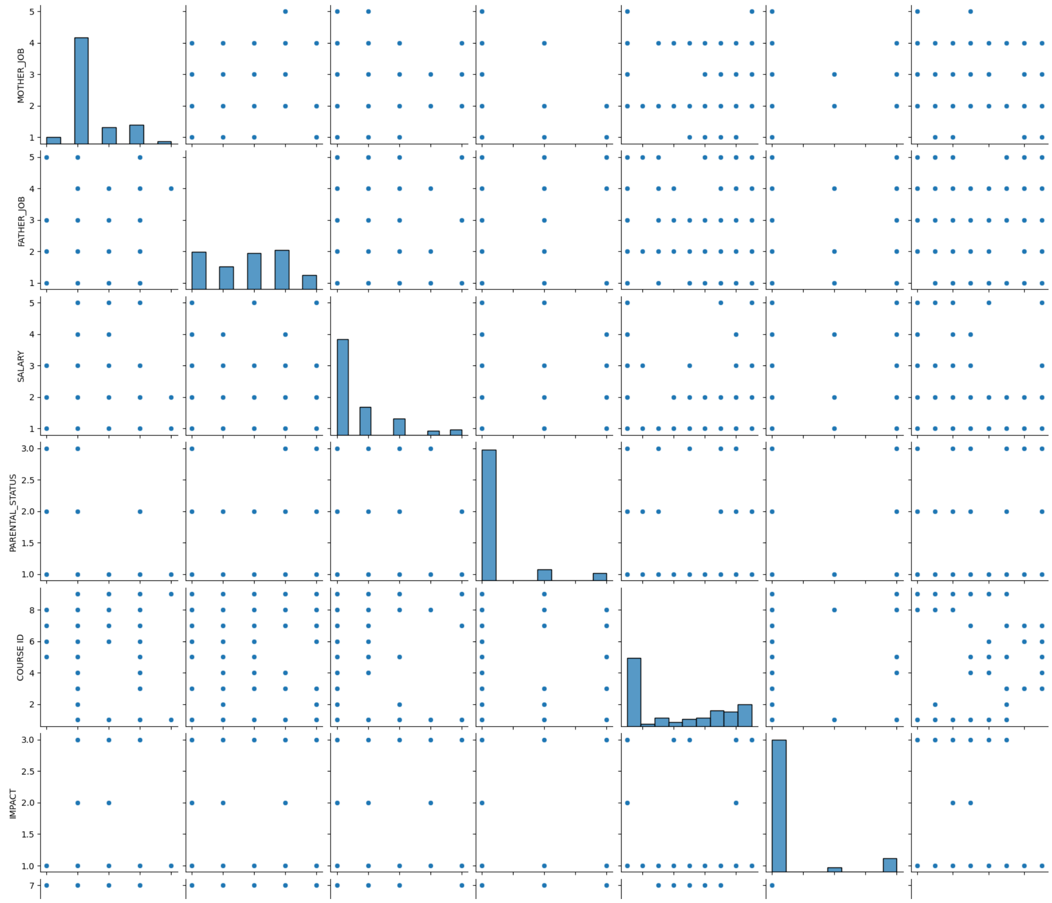
Making Visual According to The Columns

```
# Select the columns of interest
columns_of_interest = ['MOTHER_JOB', 'FATHER_JOB', 'SALARY', 'PARENTAL_STATUS', 'COURSE ID', 'IMPACT', 'GRADE']

# Create a correlation matrix
correlation_matrix = new_df[columns_of_interest].corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



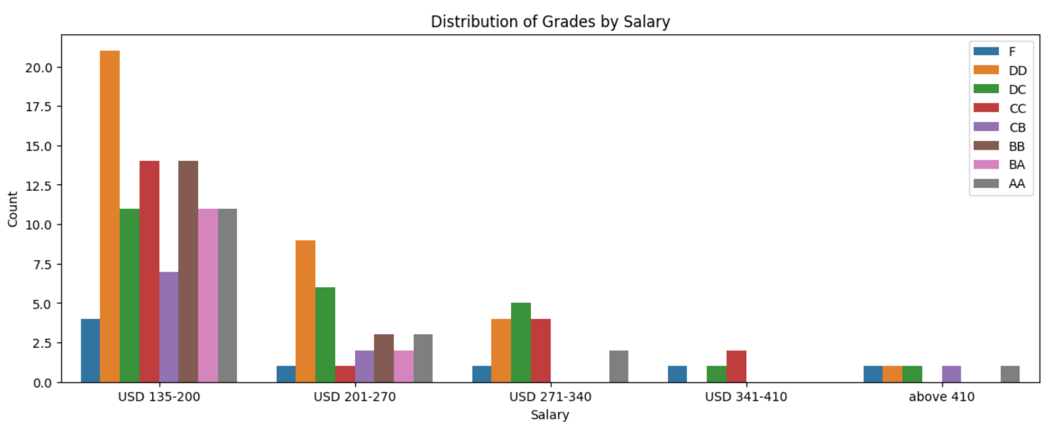

```
sns.pairplot(new_df,hue_order=['GRADE','IMPACT'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f06ae67d3c0>
```



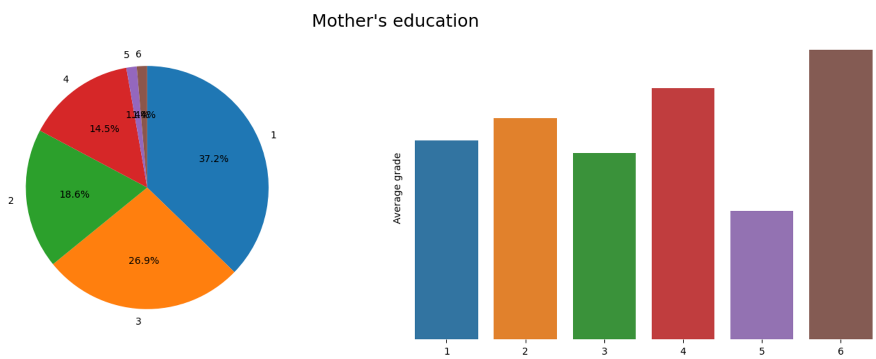
How does the financial status of a student affect their education performance?

```
plt.figure(figsize=(14, 5))
sns.countplot(data=new_df, x='SALARY', order=np.arange(1, 6, 1), hue='GRADE')
plt.xticks(np.arange(5), ['USD 135-200', 'USD 201-270', 'USD 271-340', 'USD 341-410', 'above 410'])
plt.legend(['F', 'DD', 'DC', 'CC', 'CB', 'BB', 'BA', 'AA'], loc='upper right')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.title('Distribution of Grades by Salary')
plt.show()
```

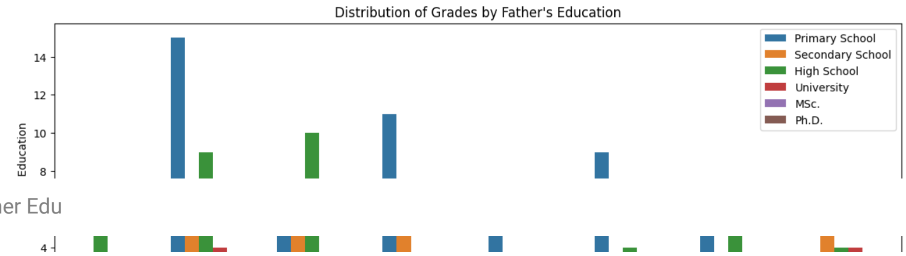


The parent's background affect the student’s performance

```
data = df['MOTHER_EDU'].value_counts(normalize=True) * 100

fig = plt.figure(figsize=(14, 5), constrained_layout=True)

plt.subplot(121)
plt.pie(data, labels=data.index, startangle=90, counterclock=False, autopct='%1.1f%%')
data = df.groupby('MOTHER_EDU')['GRADE'].mean().sort_values(ascending=False)

plt.subplot(122)
bp = sns.barplot(x=data.index, y=data)
plt.xlabel('')
plt.ylabel('Average grade')
plt.yticks([])
plt.box(False)

fig.suptitle('Mother\'s education', fontsize=18)
plt.show()
```

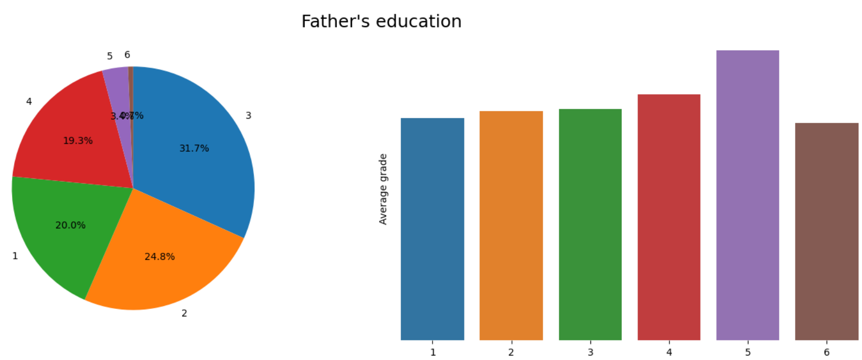



```
plt.figure(figsize=(14, 5))
sns.countplot(data=df, x='GRADE', order=np.arange(8), hue='MOTHER_EDU')
plt.xticks(np.arange(8), ['Fail', 'DD', 'DC', 'CC', 'CB', 'BB', 'BA', 'AA'])
plt.legend(['Primary School', 'Secondary School', 'High School', 'University', 'MSc.', 'Ph.D.'], loc='upper right')
plt.xlabel('Grade')
plt.ylabel("Mother's Education")
plt.title("Distribution of Grades by Father's Education")
plt.show()
```

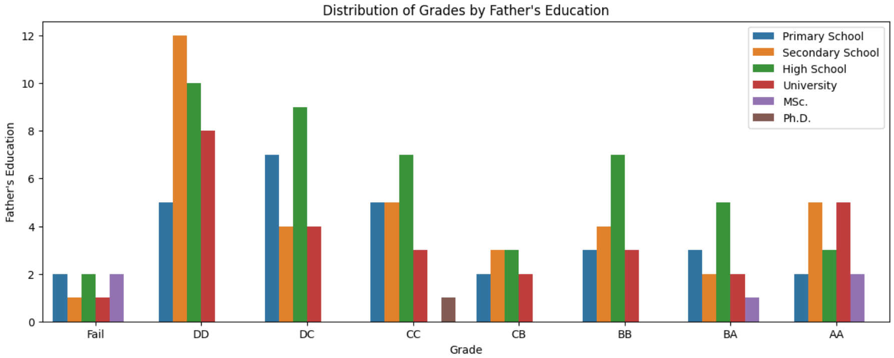


Father Edu

```
data = df['FATHER_EDU'].value_counts(normalize=True) * 100

fig = plt.figure(figsize=(14, 5), constrained_layout=True)

plt.subplot(121)
plt.pie(data, labels=data.index, startangle=90, counterclock=False, autopct='%1.1f%%')
data = df.groupby('FATHER_EDU')['GRADE'].mean().sort_values(ascending=False)

plt.subplot(122)
bp = sns.barplot(x=data.index, y=data)
plt.xlabel('')
plt.ylabel('Average grade')
plt.yticks([])
plt.box(False)

fig.suptitle('Father\'s education', fontsize=18)
plt.show()
```



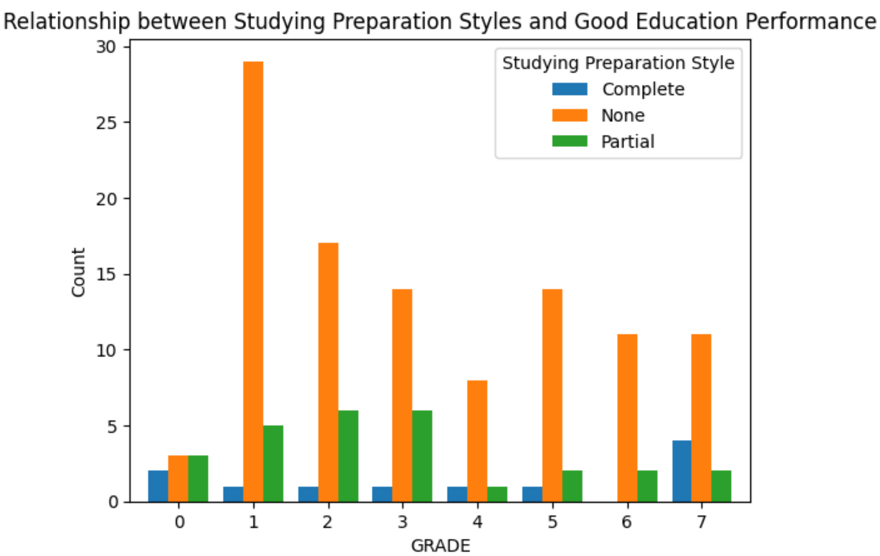

```
plt.figure(figsize=(14, 5))
sns.countplot(data=df, x='GRADE', order=np.arange(8), hue='FATHER_EDU')
plt.xticks(np.arange(8), ['Fail', 'DD', 'DC', 'CC', 'CB', 'BB', 'BA', 'AA'])
plt.legend(['Primary School', 'Secondary School', 'High School', 'University', 'MSc.', 'Ph.D.'], loc='upper right')
plt.xlabel('Grade')
plt.ylabel("Father's Education")
plt.title("Distribution of Grades by Father's Education")
plt.show()
```



The relation between studying preparation styles and good education performance

```
# Define the preparation styles and their corresponding categories
preparation_styles = {
    1: 'None',
    2: 'Partial',
    3: 'Complete'
}


# Map the preparation styles to their categories
df['PREP_STUDY_STYLE'] = df['PREP_STUDY'].map(preparation_styles)

# Group the data by grade and preparation style, and calculate the count of each preparation style
grouped_data = df.groupby(['GRADE', 'PREP_STUDY_STYLE']).size().unstack()

# Create a grouped bar chart
plt.figure(figsize=(10, 6))
grouped_data.plot(kind='bar', width=0.8)
plt.xlabel('GRADE')
plt.ylabel('Count')
plt.title('Relationship between Studying Preparation Styles and Good Education Performance')
plt.legend(title='Studying Preparation Style')
plt.xticks(rotation=0)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

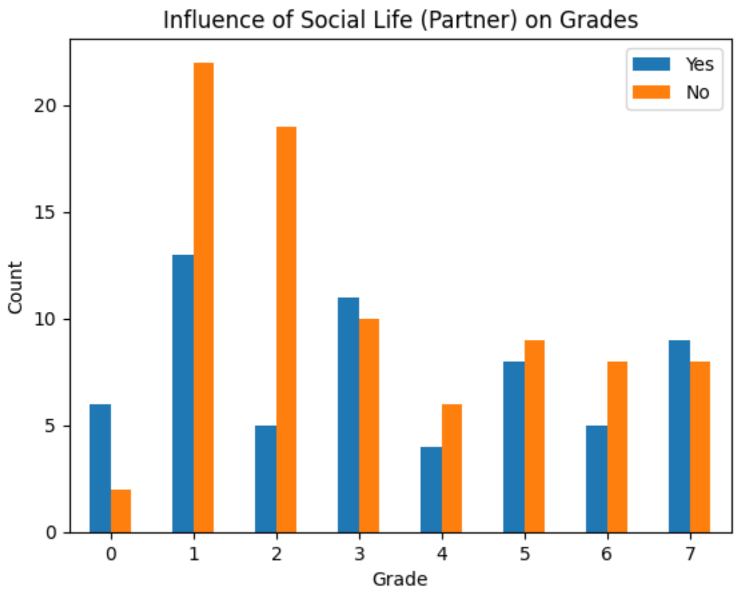


Does the performance get influenced by the social life of the students?

```
# Convert grade values to numeric
df['GRADE'] = pd.to_numeric(df['GRADE'], errors='coerce')

# Create a cross-tabulation of grade and partner
```

```
cross_tab = pd.crosstab(df['GRADE'], df['PARTNER'])

# Create a bar chart
plt.figure(figsize=(10, 6))
cross_tab.plot(kind='bar')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Influence of Social Life (Partner) on Grades')
plt.legend(['Yes','No'])
plt.xticks(rotation=0)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

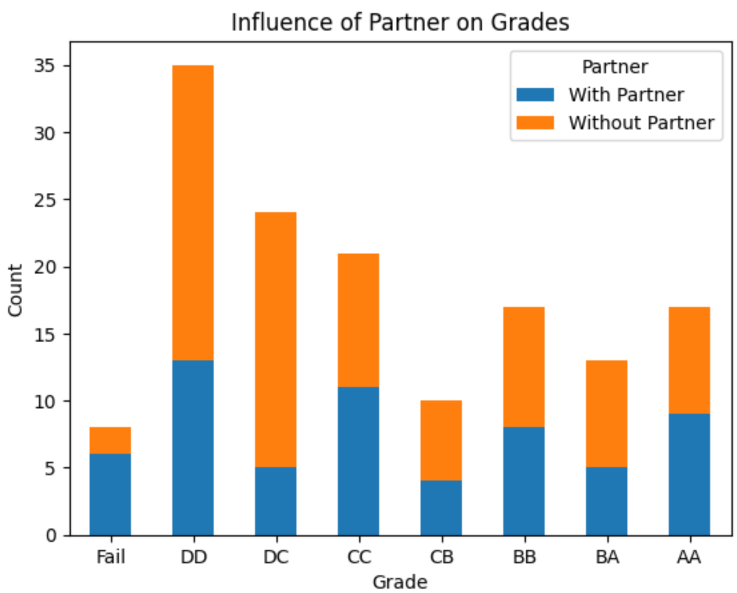



```
# Count the number of occurrences for each grade based on partner
grade_counts = df.groupby(['GRADE', 'PARTNER']).size().unstack()

# Create a bar chart
plt.figure(figsize=(10, 6))
grade_counts.plot(kind='bar', stacked=True)
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Influence of Partner on Grades')
plt.legend(title='Partner', loc='upper right')
plt.xticks(rotation=0)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

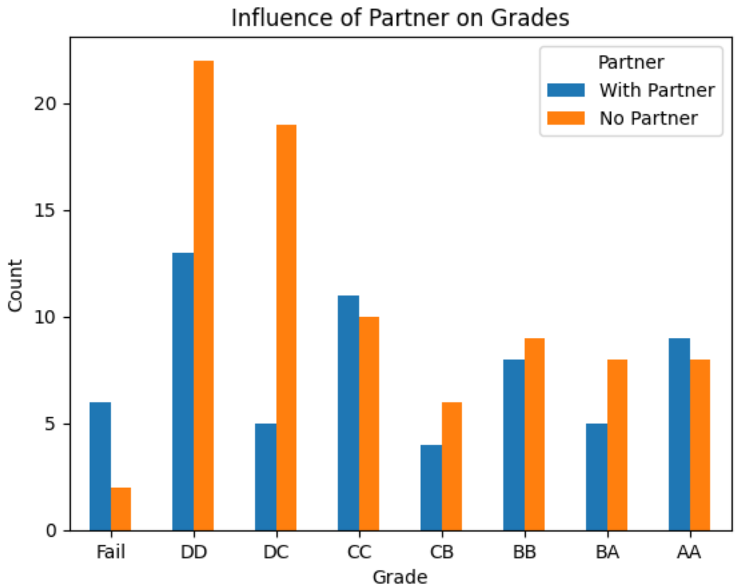
```
# Count the number of occurrences for each grade based on partner
grade_counts = df.groupby(['GRADE', 'PARTNER']).size().unstack()

# Create a bar chart
plt.figure(figsize=(10, 6))
grade_counts.plot(kind='bar', stacked=True)

# Set axis labels and title
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Influence of Partner on Grades')

# Set legend and its title
partner_labels = ['With Partner', 'Without Partner']
plt.legend(partner_labels, title='Partner', loc='upper right')

# Set x-axis tick labels
grade_labels = ['Fail', 'DD', 'DC', 'CC', 'CB', 'BB', 'BA', 'AA']
plt.xticks(range(len(grade_labels)), grade_labels, rotation=0)

# Display the chart
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```




```
# Count the number of occurrences for each grade based on partner
grade_counts = df.groupby(['GRADE', 'PARTNER']).size().unstack()

# Create a bar chart
plt.figure(figsize=(10, 6))
grade_counts.plot(kind='bar')

# Set axis labels and title
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Influence of Partner on Grades')

# Set legend and its title
partner_labels = ['With Partner', 'No Partner']
plt.legend(partner_labels, title='Partner', loc='upper right')

# Set x-axis tick labels
grade_labels = ['Fail', 'DD', 'DC', 'CC', 'CB', 'BB', 'BA', 'AA']
plt.xticks(range(len(grade_labels)), grade_labels, rotation=0)

# Display the chart
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Creating ML Model (Classification)

```
from sklearn.cluster import KMeans

from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier as dtc
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import mutual_info_classif
```

```
data = pd.read_csv("/content/student_prediction.csv")
```

```
data.head()
```

	STUDENTID	AGE	GENDER	HS_TYPE	SCHOLARSHIP	WORK	ACTIVITY	PARTNER	SALARY	TRANSP
0	STUDENT1	2	2	3	3	1	2	2	1	
1	STUDENT2	2	2	3	3	1	2	2	1	
2	STUDENT3	2	2	2	3	2	2	2	2	
3	STUDENT4	1	1	1	3	1	2	1	2	
4	STUDENT5	2	2	1	3	2	2	1	3	

5 rows × 33 columns

```
data.shape
```

```
(145, 33)
```

```
data.describe().T.style.background_gradient(cmap = "Reds")
```

	count	mean	std	min	25%	50%	75%
AGE	145.000000	1.620690	0.613154	1.000000	1.000000	2.000000	2.000000
GENDER	145.000000	1.600000	0.491596	1.000000	1.000000	2.000000	2.000000
HS_TYPE	145.000000	1.944828	0.537216	1.000000	2.000000	2.000000	2.000000
SCHOLARSHIP	145.000000	3.572414	0.805750	1.000000	3.000000	3.000000	4.000000
WORK	145.000000	1.662069	0.474644	1.000000	1.000000	2.000000	2.000000
ACTIVITY	145.000000	1.600000	0.491596	1.000000	1.000000	2.000000	2.000000
PARTNER	145.000000	1.579310	0.495381	1.000000	1.000000	2.000000	2.000000
SALARY	145.000000	1.627586	1.020245	1.000000	1.000000	1.000000	2.000000
TRANSPORT	145.000000	1.620690	1.061112	1.000000	1.000000	1.000000	2.000000
LIVING	145.000000	1.731034	0.783999	1.000000	1.000000	2.000000	2.000000
MOTHER_EDU	145.000000	2.282759	1.223062	1.000000	1.000000	2.000000	3.000000
FATHER_EDU	145.000000	2.634483	1.147544	1.000000	2.000000	3.000000	3.000000
#_SIBLINGS	145.000000	2.806897	1.360640	1.000000	2.000000	3.000000	4.000000
KIDS	145.000000	1.172414	0.490816	1.000000	1.000000	1.000000	1.000000
MOTHER_JOB	145.000000	2.358621	0.805156	1.000000	2.000000	2.000000	2.000000
FATHER_JOB	145.000000	2.806897	1.329664	1.000000	2.000000	3.000000	4.000000
STUDY_HRS	145.000000	2.200000	0.917424	1.000000	2.000000	2.000000	3.000000
READ_FREQ	145.000000	1.944828	0.562476	1.000000	2.000000	2.000000	2.000000
READ_FREQ_SCI	145.000000	2.013793	0.539884	1.000000	2.000000	2.000000	2.000000
ATTEND_DEPT	145.000000	1.213793	0.411404	1.000000	1.000000	1.000000	1.000000
IMPACT	145.000000	1.206897	0.588035	1.000000	1.000000	1.000000	1.000000
ATTEND	145.000000	1.241379	0.429403	1.000000	1.000000	1.000000	1.000000
PREP_STUDY	145.000000	1.337931	0.614870	1.000000	1.000000	1.000000	2.000000
PREP_EXAM	145.000000	1.165517	0.408483	1.000000	1.000000	1.000000	1.000000
NOTES	145.000000	2.544828	0.564940	1.000000	2.000000	3.000000	3.000000
LISTENS	145.000000	2.055172	0.674736	1.000000	2.000000	2.000000	3.000000
LIKES_DISCUSS	145.000000	2.393103	0.604343	1.000000	2.000000	2.000000	3.000000
CLASSROOM	145.000000	1.806897	0.810492	1.000000	1.000000	2.000000	2.000000

```
#look the unique value from Course ID
data["COURSE ID"].unique()
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
data.describe(include=object)
```

	STUDENTID
count	145
unique	145
top	STUDENT1
freq	1

```
#drop feature that have unique value
data = data.drop('STUDENTID', axis=1)
```

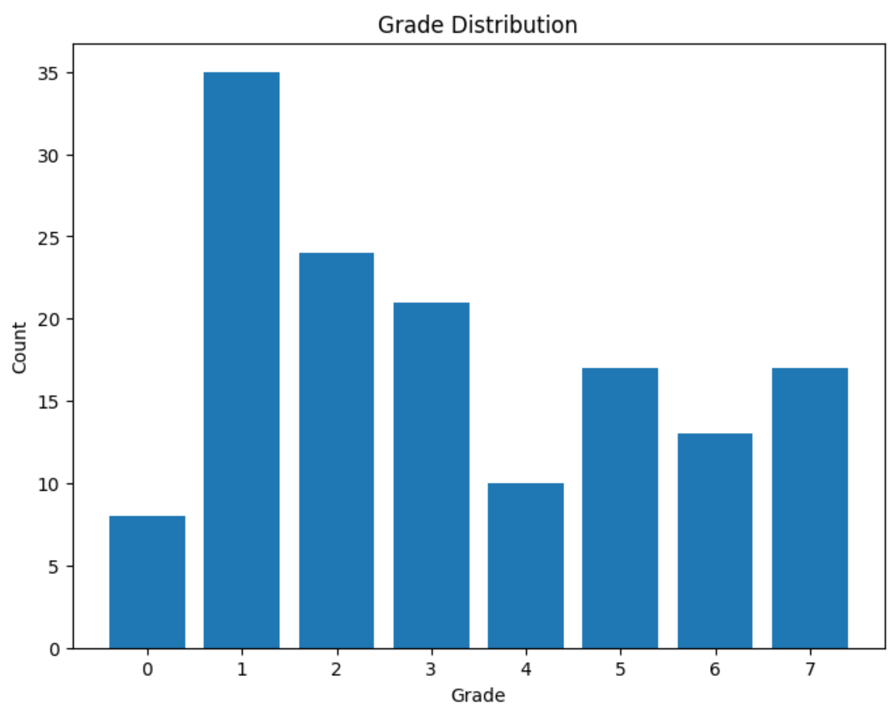
```
#check duplicate
duplicate = data[data.duplicated()]
duplicate
```

AGE	GENDER	HS_TYPE	SCHOLARSHIP	WORK	ACTIVITY	PARTNER	SALARY	TRANSPORT	LIVING

```
data["GRADE"].value_counts()
```

```
1    35
2    24
3    21
5    17
7    17
6    13
4    10
0     8
Name: GRADE, dtype: int64
```

```
grade_counts = data["GRADE"].value_counts()

# Create a bar chart
plt.figure(figsize=(8, 6))
plt.bar(grade_counts.index, grade_counts.values)

# Set axis labels and title
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Grade Distribution')

# Display the chart
plt.show()
```



feature selection

```
X = data.drop('GRADE', axis=1)
y = data['GRADE']

# list discrete features that have integer dtypes for using MI (Mutual Information)
discrete_features = X.dtypes == int


def make_mi_scores(X, y, discrete_features):
    mi_scores = mutual_info_classif(X, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores
```

```
mi_scores = make_mi_scores(X, y, discrete_features)
mi_scores  # show a few features with their MI scores
```

```
COURSE ID        0.623271
CUML_GPA         0.250852
MOTHER_EDU       0.184561
STUDY_HRS        0.152261
SCHOLARSHIP      0.142022
FATHER_JOB       0.134495
EXP_GPA          0.127521
GENDER           0.120415
SALARY           0.117921
#_SIBLINGS       0.104088
MOTHER_JOB       0.102861
AGE              0.100847
FATHER_EDU       0.097289
TRANSPORT        0.093659
IMPACT           0.081894
LIVING           0.077654
READ_FREQ        0.075782
READ_FREQ_SCI    0.066456
NOTES            0.065915
PREP_STUDY       0.059436
LIKES_DISCUSS    0.052028
HS_TYPE          0.050828
KIDS             0.047981
LISTENS          0.047148
ATTEND           0.039263
PARTNER          0.037048
WORK             0.032927
ACTIVITY         0.032521
CLASSROOM        0.029650
PREP_EXAM        0.029195
ATTEND_DEPT      0.028835
Name: MI Scores, dtype: float64
```

```
# Define a threshold to determine informative features
threshold = 0.1

# Select the informative features based on the MI scores
informative_features = mi_scores[mi_scores > threshold].index

# Create a new DataFrame with only the informative features
data_informative = data[informative_features]

# Print the informative features
print(data_informative.columns)
```

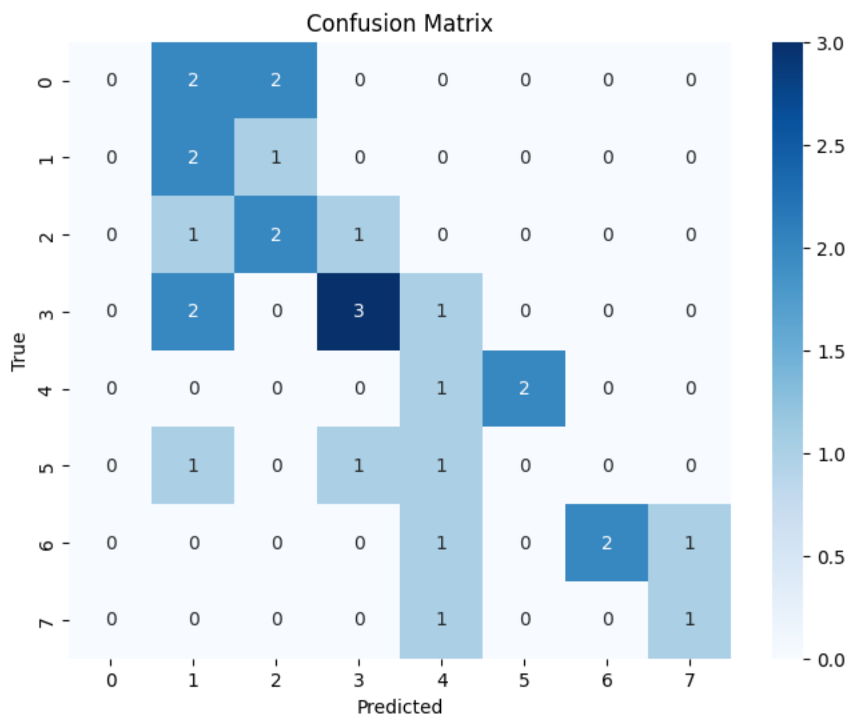
```
Index(['COURSE ID', 'CUML_GPA', 'MOTHER_EDU', 'STUDY_HRS', 'SCHOLARSHIP',
       'FATHER_JOB', 'EXP_GPA', 'GENDER', 'SALARY', '#_SIBLINGS', 'MOTHER_JOB',
       'AGE'],
      dtype='object')
```

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


# Select the relevant features based on MI scores
selected_features = ['COURSE ID', 'CUML_GPA', 'MOTHER_EDU', 'STUDY_HRS', 'SCHOLARSHIP',
                     'FATHER_JOB', 'EXP_GPA', 'GENDER', 'SALARY', '#_SIBLINGS', 'MOTHER_JOB', 'AGE']

# Extract the selected features and the target variable
X = data[selected_features]
y = data['GRADE']

# Encode categorical variables if necessary
label_encoder = LabelEncoder()
X['GENDER'] = label_encoder.fit_transform(X['GENDER'])
# Perform label encoding for other categorical features if needed

# 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=42)

# Initialize the classifier
clf = DecisionTreeClassifier()
```

```
# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy for all columns:", accuracy*100)
```

```
Accuracy for all columns: 37.93103448275862
```

```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```



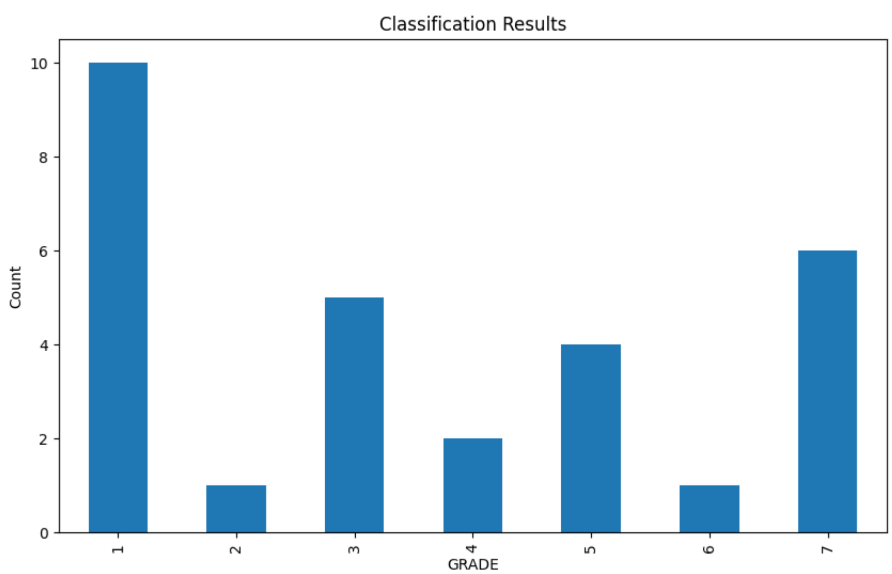

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.25      0.67      0.36         3
           2       0.40      0.50      0.44         4
           3       0.60      0.50      0.55         6
           4       0.20      0.33      0.25         3
           5       0.00      0.00      0.00         3
           6       1.00      0.50      0.67         4
           7       0.50      0.50      0.50         2

    accuracy                           0.38        29
   macro avg       0.37      0.38      0.35        29
weighted avg       0.40      0.38      0.36        29
```

```
# Select the relevant features
selected_features = ['MOTHER_JOB', 'STUDY_HRS', 'AGE']
```

```
X = data[selected_features]
y = data['GRADE']

# Encode categorical variables if necessary
label_encoder = LabelEncoder()
X['MOTHER_JOB'] = label_encoder.fit_transform(X['MOTHER_JOB'])

# 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=42)

# Train the random forest classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

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

           0       0.00      0.00      0.00         4
           1       0.00      0.00      0.00         3
           2       1.00      0.25      0.40         4
           3       0.60      0.50      0.55         6
           4       0.00      0.00      0.00         3
           5       0.25      0.33      0.29         3
           6       0.00      0.00      0.00         4
           7       0.00      0.00      0.00         2

    accuracy                           0.17        29
   macro avg       0.23      0.14      0.15        29
weighted avg       0.29      0.17      0.20        29
```

```
# Create a bar plot of the classification results
plt.figure(figsize=(10, 6))
pd.Series(y_pred).value_counts().sort_index().plot(kind='bar')
plt.xlabel('GRADE')
plt.ylabel('Count')
plt.title('Classification Results')
plt.show()
```

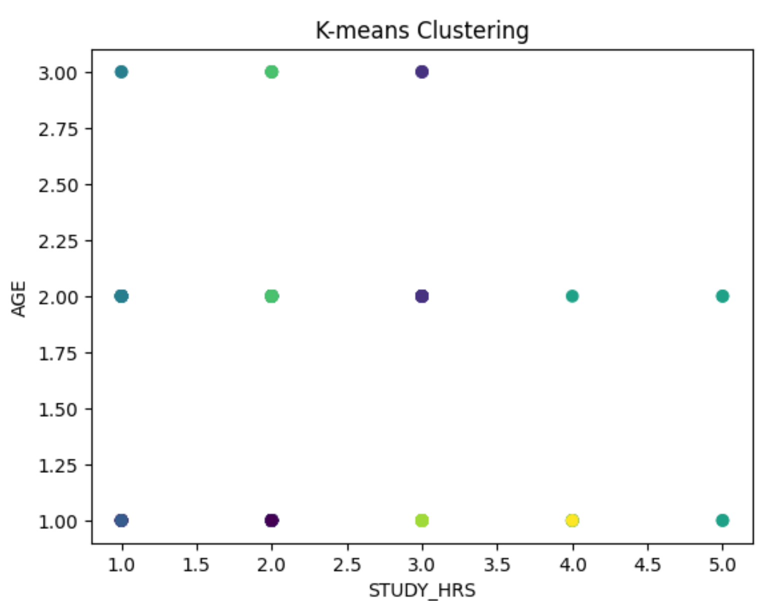


k-means

```
# Select the relevant features
selected_features = ['MOTHER_JOB', 'STUDY_HRS', 'AGE']
X = data[selected_features]

# Encode categorical variables if necessary
label_encoder = LabelEncoder()
X['MOTHER_JOB'] = label_encoder.fit_transform(X['MOTHER_JOB'])

# Perform K-means clustering
k = 4  # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Add the cluster labels to the DataFrame
data['Cluster'] = labels

# Print the count of data points in each cluster
print(data['Cluster'].value_counts())
```

```
2    55
1    47
0    31
3    12
Name: Cluster, dtype: int64
```

```
from sklearn.cluster import KMeans

# Select the relevant columns for clustering
X = df[['MOTHER_JOB', 'STUDY_HRS', 'AGE']]

# Perform K-means clustering
kmeans = KMeans(n_clusters=8, random_state=42)
kmeans.fit(X)

# Add the cluster labels to the DataFrame
df['Cluster'] = kmeans.labels_

# Plot the clusters
plt.scatter(df['STUDY_HRS'], df['AGE'], c=df['Cluster'], cmap='viridis')
plt.xlabel('STUDY_HRS')
plt.ylabel('AGE')
plt.title('K-means Clustering')
plt.show()
```




```
# Get the predicted cluster labels from k-means
y_pred_kmeans = kmeans.labels_
```

```
# Calculate accuracy for k-means clustering
kmeans_accuracy = accuracy_score(y, y_pred_kmeans)

print(kmeans_accuracy*100)
```

```
11.724137931034482
```

Comparing

```
# Create a bar plot of the k-means clusters
plt.figure(figsize=(10, 6))
data['Cluster'].value_counts().sort_index().plot(kind='bar')
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.title('K-means Clustering Results')
plt.show()

# Create a bar plot of the classification results
plt.figure(figsize=(10, 6))
pd.Series(y_pred).value_counts().sort_index().plot(kind='bar')
plt.xlabel('GRADE')
plt.ylabel('Count')
plt.title('Classification Results')
plt.show()
```


```
# Calculate accuracy for classification
classification_accuracy = accuracy_score(y_test, y_pred)

# Calculate accuracy differences
accuracy_difference = classification_accuracy - kmeans_accuracy

# Print the accuracy differences
print("Accuracy Difference - K-means Clustering vs Classification:", accuracy_difference*100)
```

```
Accuracy Difference - K-means Clustering vs Classification: 5.517241379310346
```

SVM

```
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Select the relevant features
selected_features = ['CUML_GPA', 'MOTHER_EDU', 'STUDY_HRS', 'SCHOLARSHIP', 'AGE']
X = data[selected_features]
y = data['GRADE']

# Encode categorical variables if necessary
label_encoder = LabelEncoder()
X['MOTHER_EDU'] = label_encoder.fit_transform(X['MOTHER_EDU'])

# 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=42)

# Create and train the SVM classifier
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
```

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

           0       0.00      0.00      0.00         4
           1       0.08      0.33      0.13         3
           2       0.20      0.25      0.22         4
           3       1.00      0.17      0.29         6
           4       0.00      0.00      0.00         3
           5       0.14      0.33      0.20         3
           6       0.00      0.00      0.00         4
           7       0.00      0.00      0.00         2

    accuracy                           0.14        29
   macro avg       0.18      0.14      0.11        29
weighted avg       0.26      0.14      0.12        29

Accuracy: 13.793103448275861
```

comparing all the models

```
# Select the relevant features for clustering
selected_features_cluster = ['CUML_GPA', 'MOTHER_EDU', 'STUDY_HRS', 'SCHOLARSHIP', 'AGE']
X_cluster = data[selected_features_cluster]

# Perform K-means clustering
kmeans = KMeans(n_clusters=8, random_state=42)
kmeans.fit(X_cluster)
data['Cluster'] = kmeans.labels_

# Train the Random Forest classifier
random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)

# Train the SVM classifier
support_vector_machine = SVC(kernel='linear')
support_vector_machine.fit(X_train, y_train)

# Make predictions on the test set for each model
y_pred_kmeans = kmeans.predict(X_test)
y_pred_rf = random_forest.predict(X_test)
y_pred_svm = support_vector_machine.predict(X_test)

# Calculate accuracy for each model
kmeans_accuracy = accuracy_score(y_test, y_pred_kmeans)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
svm_accuracy = accuracy_score(y_test, y_pred_svm)

# Print the accuracy scores
print("K-means Clustering Accuracy:", kmeans_accuracy*100)
print("Random Forest Accuracy:", rf_accuracy*100)
print("Support Vector Machine Accuracy:", svm_accuracy*100)
```

```
K-means Clustering Accuracy: 13.793103448275861
Random Forest Accuracy: 27.586206896551722
Support Vector Machine Accuracy: 13.793103448275861
```

```
# Calculate accuracy for each model
accuracy_scores = [kmeans_accuracy, rf_accuracy, svm_accuracy]

# Create a bar plot to compare accuracy
models = ['K-means', 'Random Forest', 'SVM']
plt.bar(models, accuracy_scores)
plt.ylim(0, 1)  # Set the y-axis limit to 0-1 for better visualization
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Models')
plt.show()
```