**IB Computer Science HL - Decision Tree Project**

**Academic Predictions Based on Student Behavior - Zara Tekmen**

Introduction

**Research Objective:**

In this project, I am exploring the predictive power of a Decision Tree algorithm using a dataset that includes a wide range of student attributes, such as grades, demographic factors, family statistics, and school-related features.

My research objective is to develop and refine a Decision Tree model that can predict a student's final grade in high school Mathematics based on their individual characteristics and historical academic performance. This model will be trained using a dataset containing various predictive attributes gathered from two distinct educational datasets focusing on student performance in Mathematics and Portuguese language subjects.

The application of this model can help identify key factors of how well a student will do in high school, and serve as a tool to educational institutions to tailor teaching methods and support that align with the needs of their students based on the predictive analysis of this model.

**Data Source:** [**Student Performance Data Set**](https://www.kaggle.com/datasets/larsen0966/student-performance-data-set)

The dataset include attributes such as the students' grades, demographic, family statistics, and school-related features. The source provided two distinct datasets from these schools regarding student performance in the subjects: Mathematics and Portuguese language. I focused on the Mathematics subject because I figured it would be more universal for its worldwide implications.

**Research Question:**

* Can we use a set of attributes of a student to predict what their grade in their final year of high school will be using a Decision Tree algorithm?

***Key***

| school | student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira) |
| --- | --- |
| sex | student's sex (binary: 'F' - female or 'M' - male) |
| age | student's age (numeric: from 15 to 22) |
| address | student's home address type (binary: 'U' - urban or 'R' - rural) |
| famsize | family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3) |
| Pstatus | parent's cohabitation status (binary: 'T' - living together or 'A' - apart) |
| Medu | mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education) |
| Fedu | father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education) |
| Mjob | mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at\_home' or 'other') |
| Fjob | father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at\_home' or 'other') |
| reason | reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other') |
| guardian | student's guardian (nominal: 'mother', 'father' or 'other') |
| traveltime | home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour) |
| studytime | weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours) |
| failures | number of past class failures (numeric: n if 1<=n<3, else 4) |
| schoolsup | extra educational support (binary: yes or no) |
| famsup | family educational support (binary: yes or no) |
| paid | extra paid classes within the course subject (Math or Portuguese) (binary: yes or no) |
| activities | extra-curricular activities (binary: yes or no) |
| nursery | attended nursery school (binary: yes or no) |
| higher | wants to take higher education (binary: yes or no) |
| internet | Internet access at home (binary: yes or no) |
| romantic | with a romantic relationship (binary: yes or no) |
| famrel | quality of family relationships (numeric: from 1 - very bad to 5 - excellent) |
| freetime | free time after school (numeric: from 1 - very low to 5 - very high) |
| goout | going out with friends (numeric: from 1 - very low to 5 - very high) |
| Dalc | workday alcohol consumption (numeric: from 1 - very low to 5 - very high) |
| Walc | weekend alcohol consumption (numeric: from 1 - very low to 5 - very high) |
| health | current health status (numeric: from 1 - very bad to 5 - very good) |
| absences | number of school absences (numeric: from 0 to 93) |

***Categorical Variable Histogram of All Variables***

import pandas as pd

import matplotlib.pyplot as plt

import os

# load dataset

df\_x = pd.read\_excel('dataset.xlsx', engine='openpyxl')

df\_x.to\_csv('dataset.csv', index=False)

df = pd.read\_csv('dataset.csv')

histograms\_dir = 'Histograms'

if not os.path.exists(histograms\_dir):

os.makedirs(histograms\_dir)

# create histograms

def plot\_histograms(df):

for col in df.columns:

plt.figure()

if df[col].dtype in ['int64', 'float64']:

df[col].hist(bins=15)

plt.title(col)

plt.xlabel(col)

plt.ylabel('Frequency')

else:

df[col].value\_counts().plot(kind='bar')

plt.title(col)

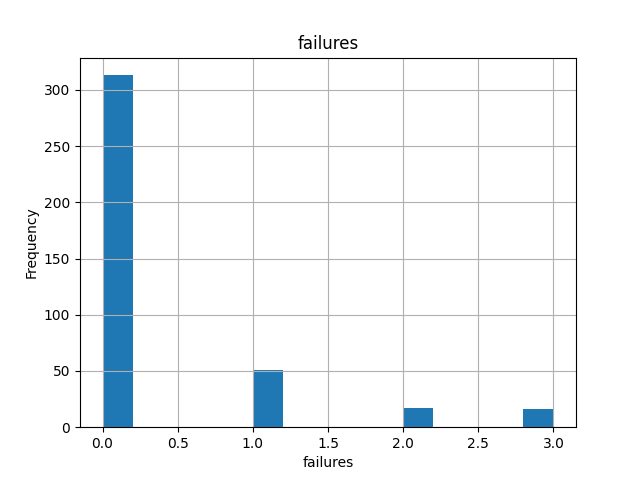
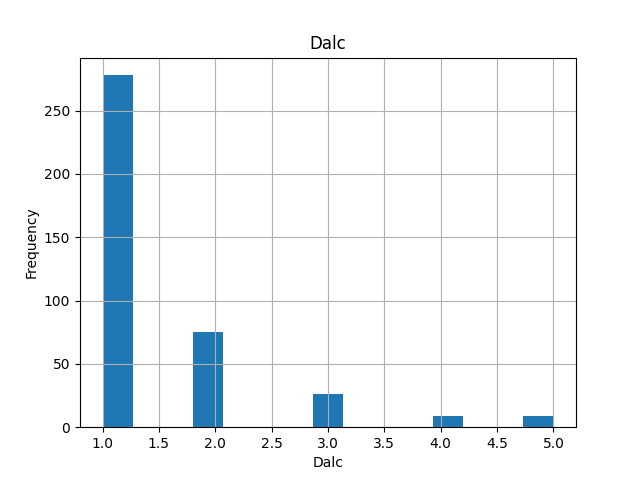
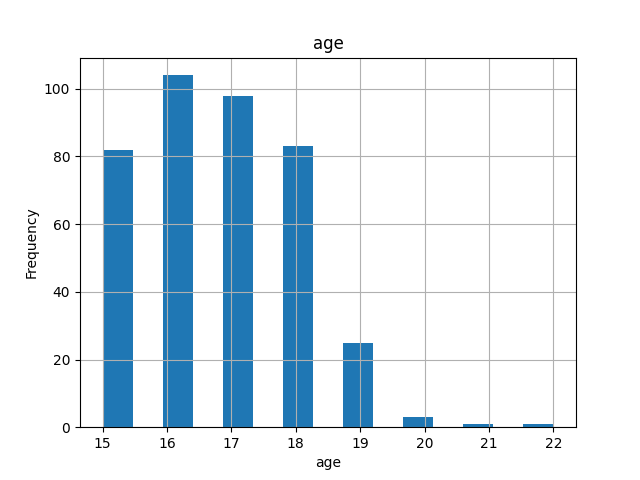
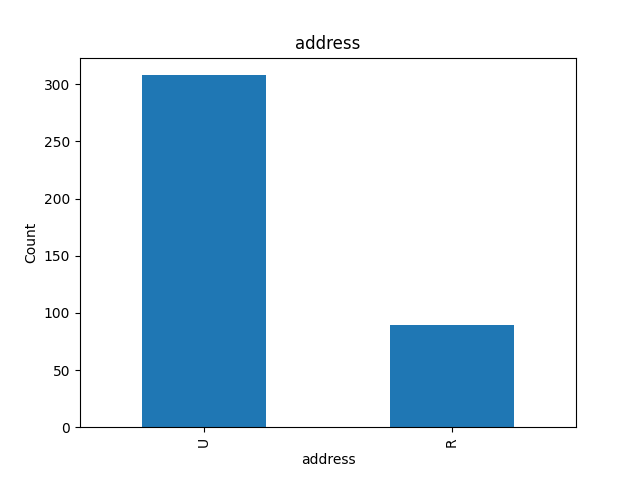
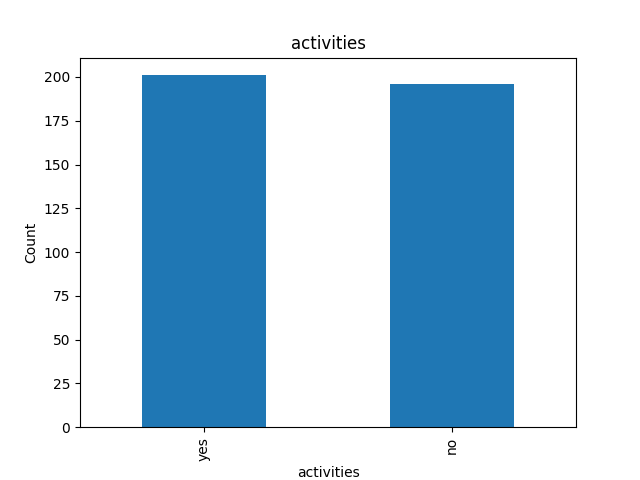
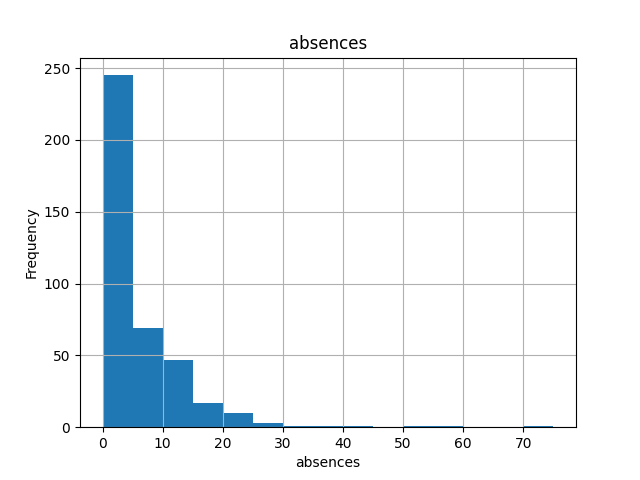
plt.xlabel(col)

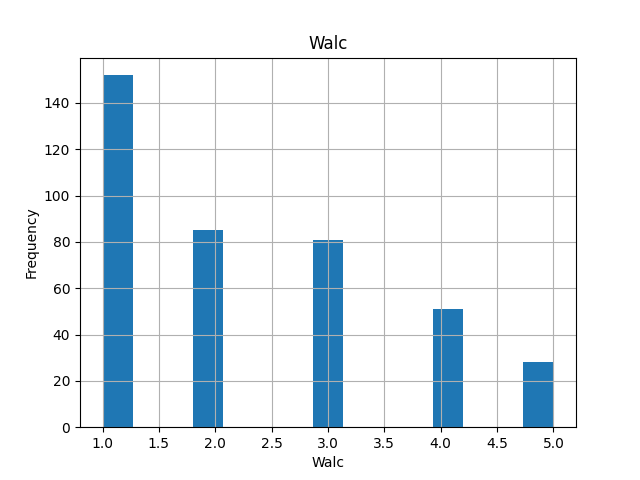
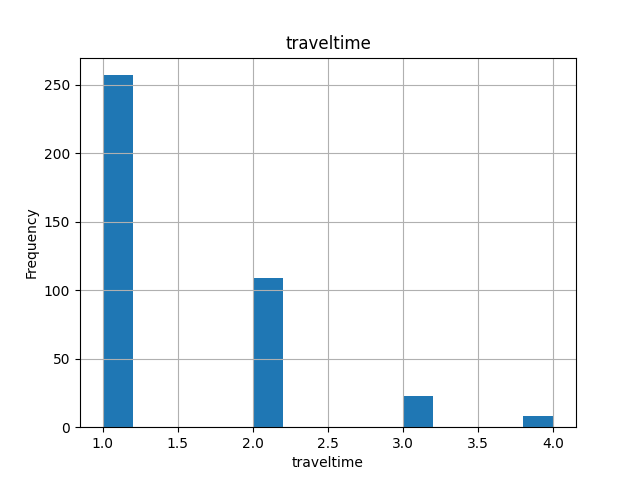
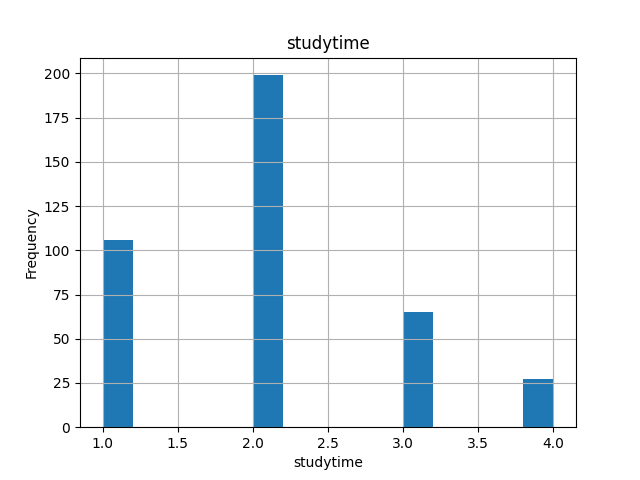
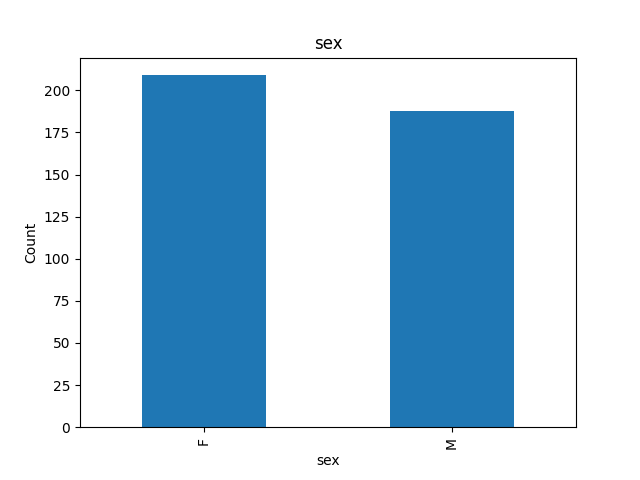
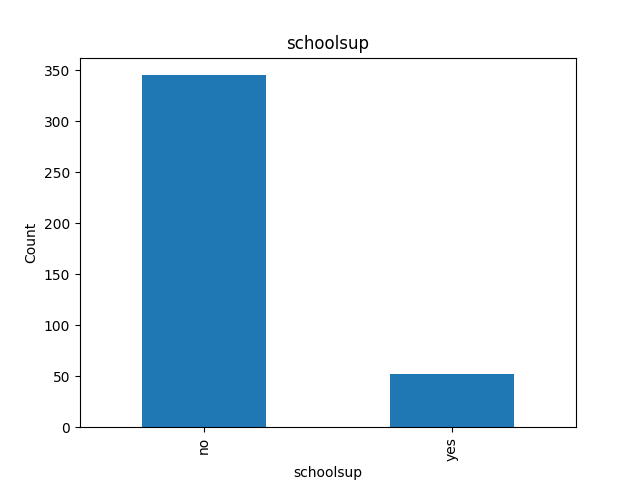
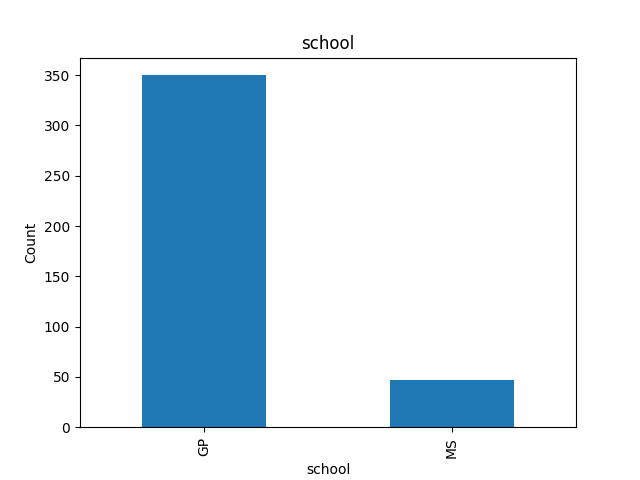
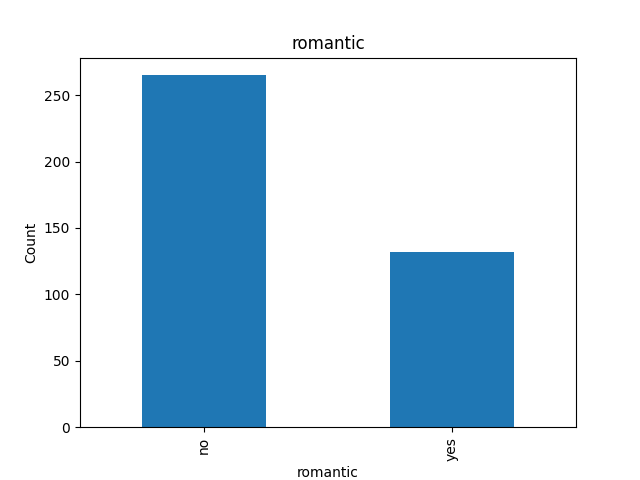
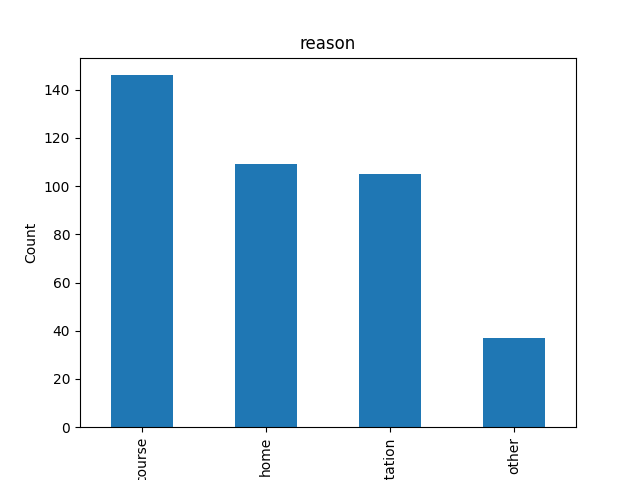
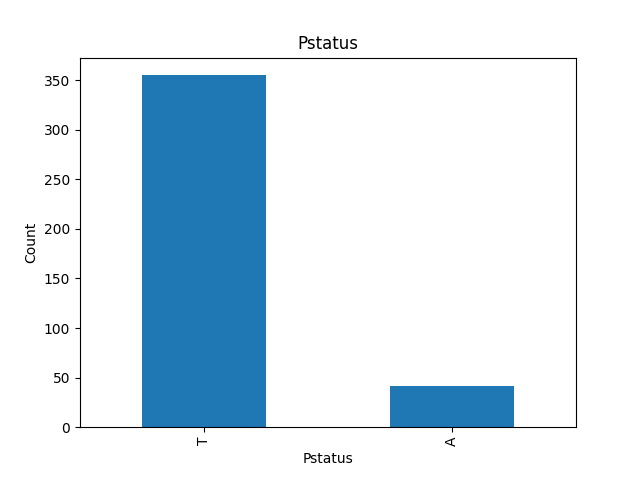
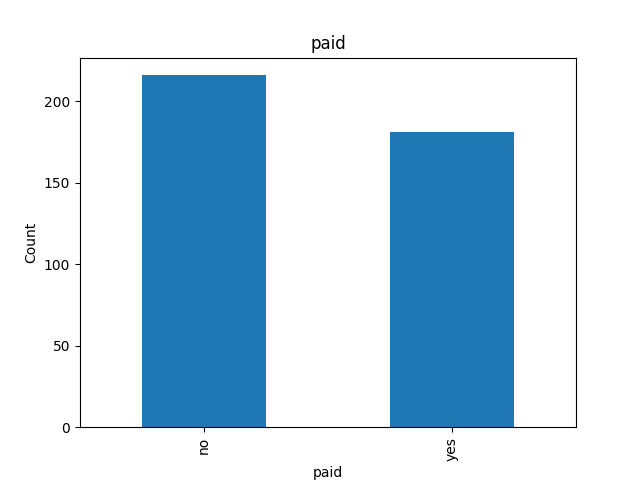
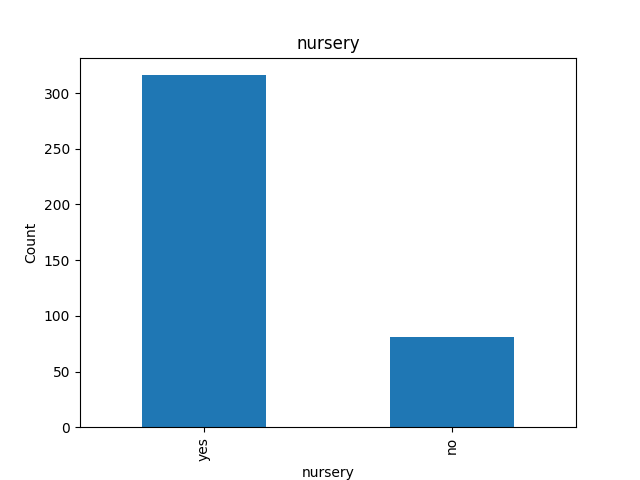
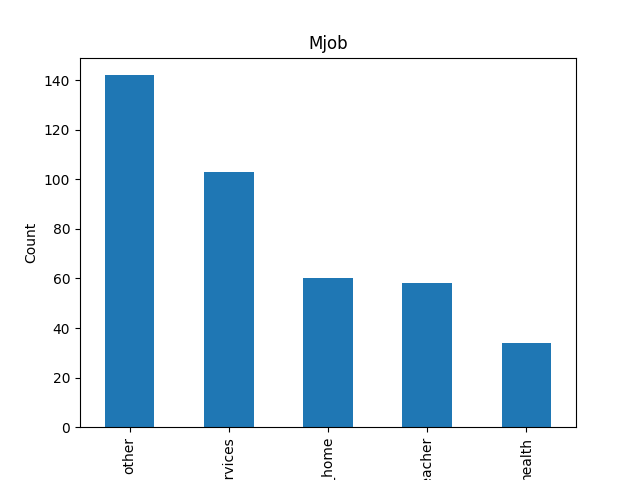
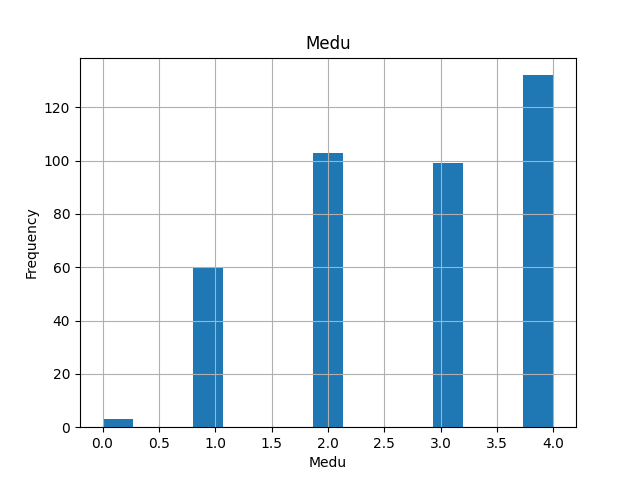
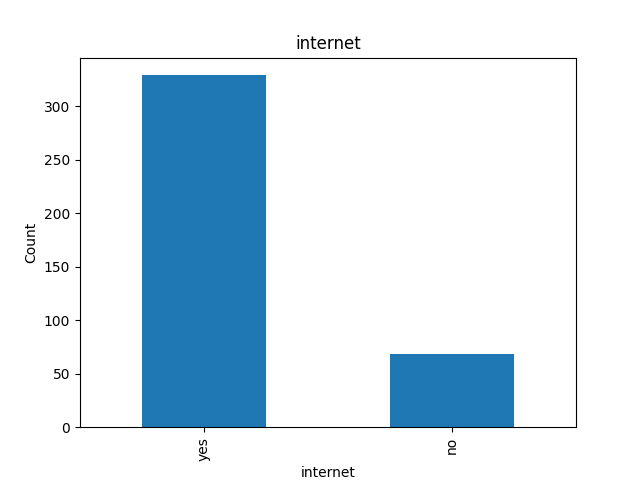
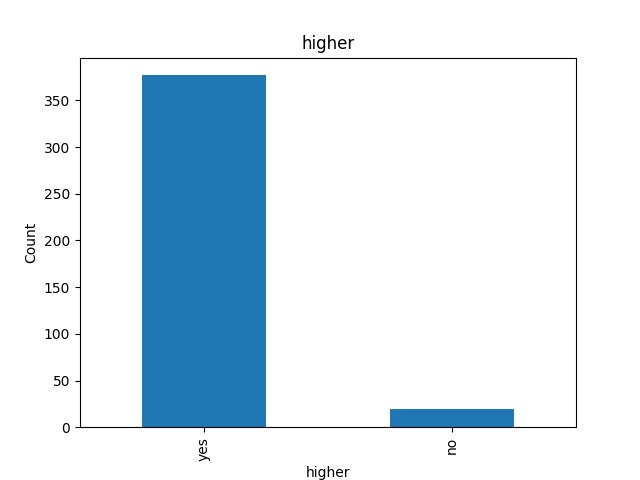
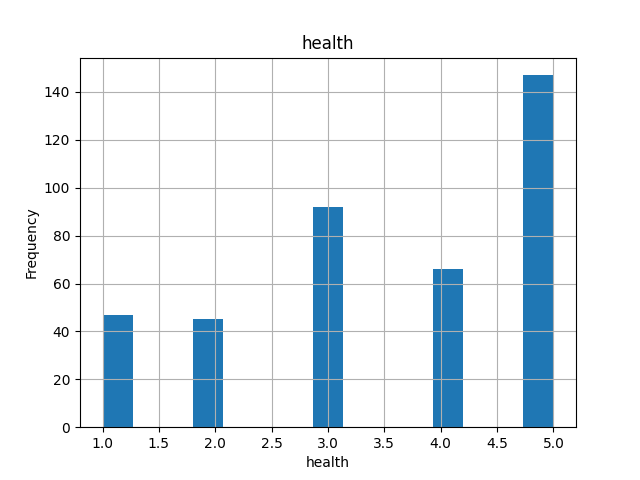
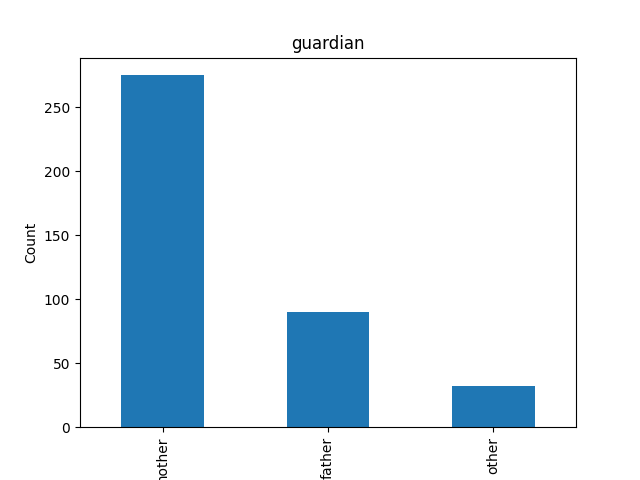
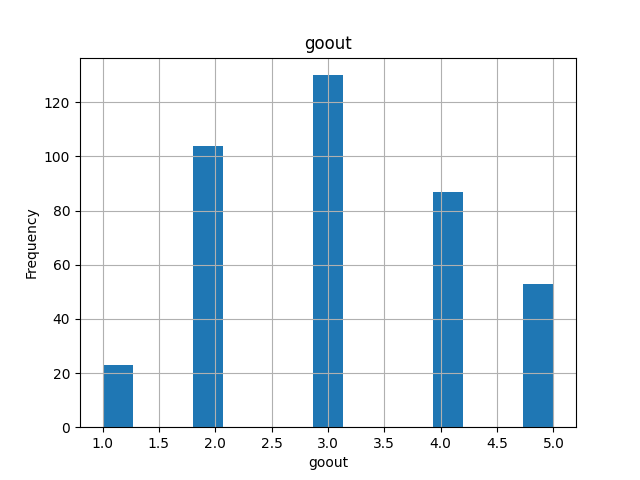
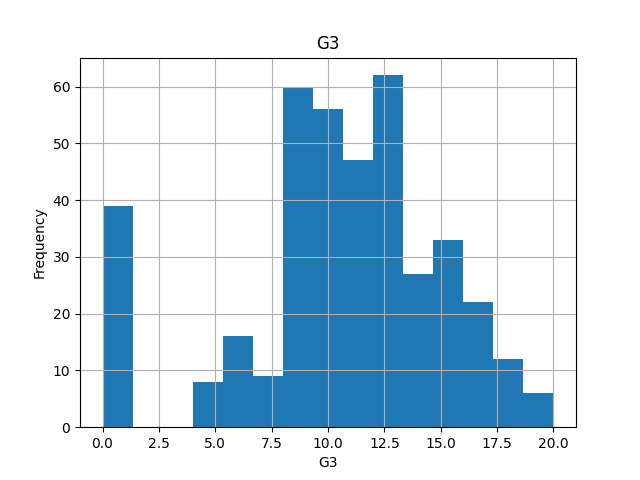
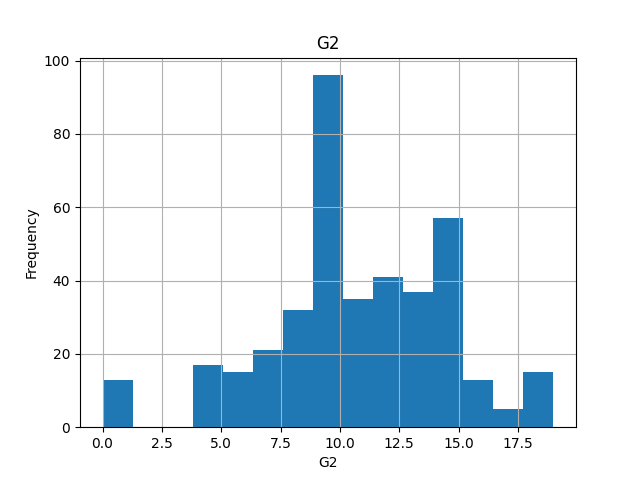
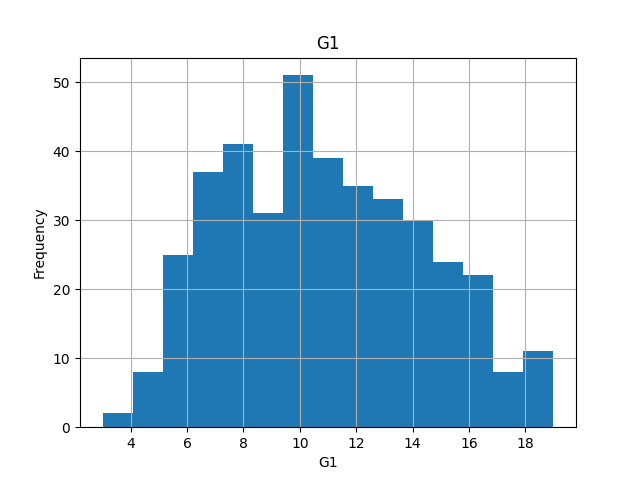
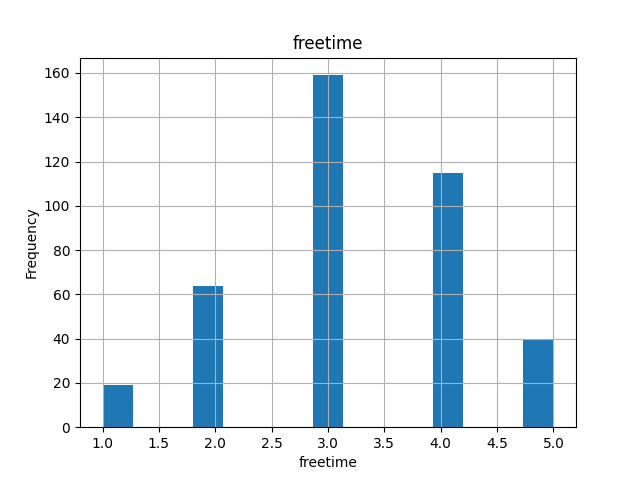
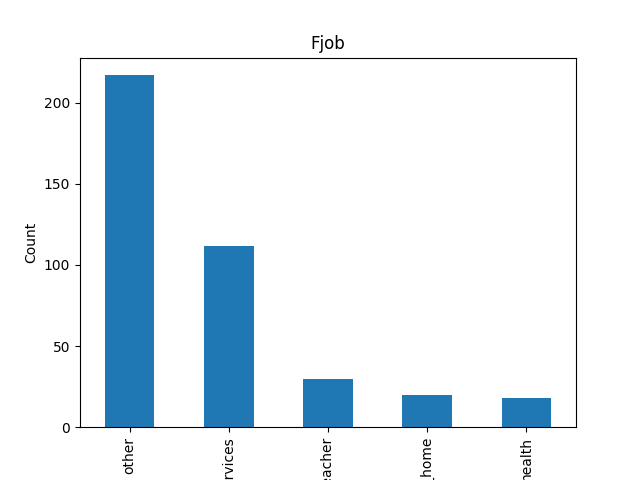
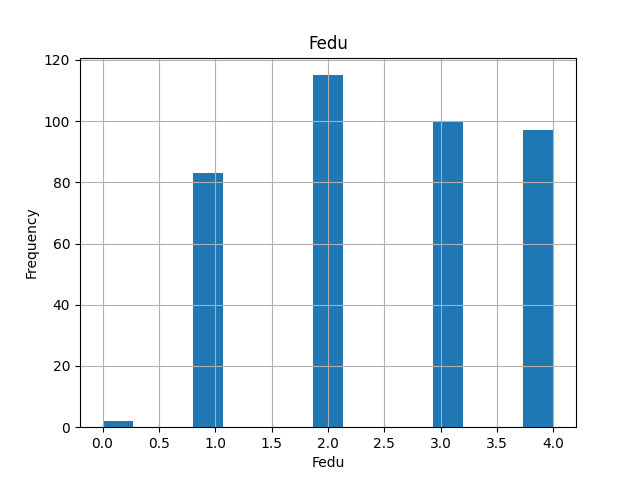
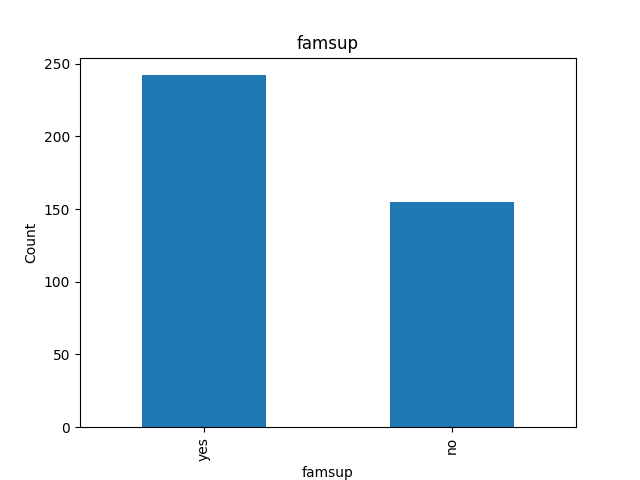
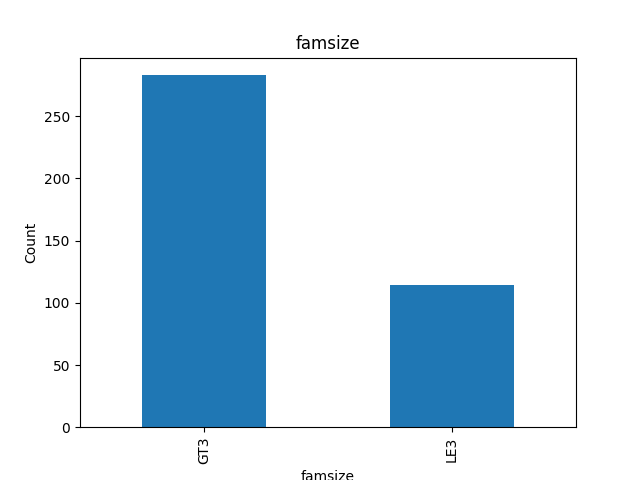
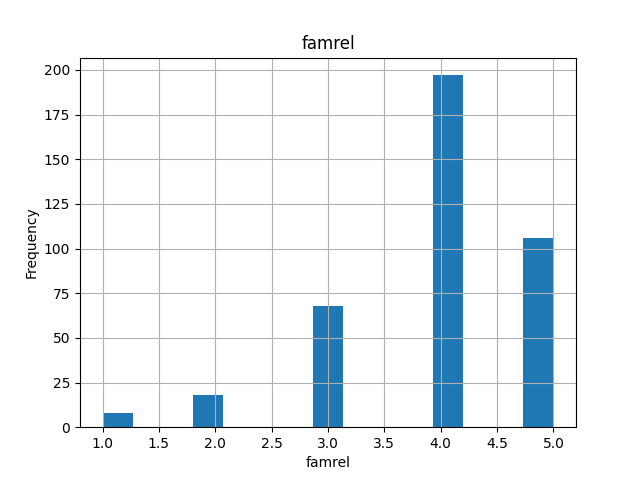
plt.ylabel('Count')

plt.savefig(f'{histograms\_dir}/{col}\_histogram.png')

plt.show()

plot\_histograms(df)





***Key Attributes and Their Correlation Through Scatter Plots***

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# load dataset

df\_x = pd.read\_excel('dataset.xlsx', engine='openpyxl')

df\_x.to\_csv('dataset.csv', index=False)

df = pd.read\_csv('dataset.csv')

selected\_columns = df[['absences', 'studytime', 'higher', 'freetime', 'G3']]

selected\_columns['higher'] = selected\_columns['higher'].map({'yes': 1, 'no': 0})

sns.pairplot(selected\_columns)

plt.suptitle('Scatter Matrix of Selected Variables', y=1.02) # Adjust title and its position

plt.savefig(fname = 'Blue Scatter Matrix.png', transparent = True)

plt.show()

***Analysis of Graphical Visualizations***

* Absence and G3: .
  + There appears to be a general trend where higher absenteeism corresponds to lower G3 scores. The scatter plot shows that many students with low attendance tend to have a wide range of grades, while the majority of those with very high academic achievement have low grades
* Study time and G3:
  + The scatter plot shows that longer study periods are generally associated with higher grades (G3). This is particularly evident in the interpolated data points showing that students who study more tend to have moderate to high grades, while less time spent studying is associated with perfect grades
* Higher (preference for higher education) and G3:
  + Students who want to go to university (high=1) seem to score in all G3, but focus on higher grades. Those with no aspirations (high=0) score even lower, suggesting that the desire for higher education may serve as a motivating factor for better academic performance.
* Free time with G3:
  + The relationship between relaxation time and G3 does not indicate any clear trend. Since the students’ leisure time varies, they seem to score points in G3 grades. This may suggest that leisure time alone is not a significant predictor of academic achievement.
* Diagonal (Histograms)l .
  + The histograms on the diagonal give the distribution of each variable. Notable:
  + Absenteeism is quite skewed, with most students having few absences.
  + Study time means most students have short to moderate study time.
  + Advanced Career Most students aspire to higher education.
  + Free time is evenly distributed but it shifts to more break time.
  + G3 is approximately normally distributed with a slight right skew, indicating that most students score around the median, with a few scoring very high or very low

******

***Decision Tree***

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn import tree

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.datasets import load\_iris # Example dataset

# load dataset

df = pd.read\_excel('dataset.xlsx', engine='openpyxl')

print(df.head())

print(df['G3'].value\_counts())

# preprocess data

df['higher'] = df['higher'].map({'yes': 1, 'no': 0}).fillna(0)

categorical\_cols = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'internet', 'romantic']

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

# select features, set g3 as target

X = df.drop('G3', axis=1) # Adjust if different features or target are desired

y = df['G3']

# split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# initilize dtree regressor

tree\_model = DecisionTreeRegressor(random\_state=42)

# train model

tree\_model.fit(X\_train, y\_train)

# feature importance

importances = tree\_model.feature\_importances\_

print(importances)

# predict and evaluate

y\_pred = tree\_model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print('Mean Squared Error:', mse)

print(accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

feature\_importance = sorted(zip(tree\_model.feature\_importances\_, X.columns), reverse=True)

print(feature\_importance[:10]) # top 10 features

print("All features:", sum(importances))

print("Top 10 features:", sum([imp for imp, \_ in feature\_importance[:10]]))

# visualize and save dtree

plt.figure(figsize=(15000/100, 3000/100), dpi=100)

tree\_plot = tree.plot\_tree(tree\_model,

filled=True,

feature\_names=X.columns,

fontsize=10,

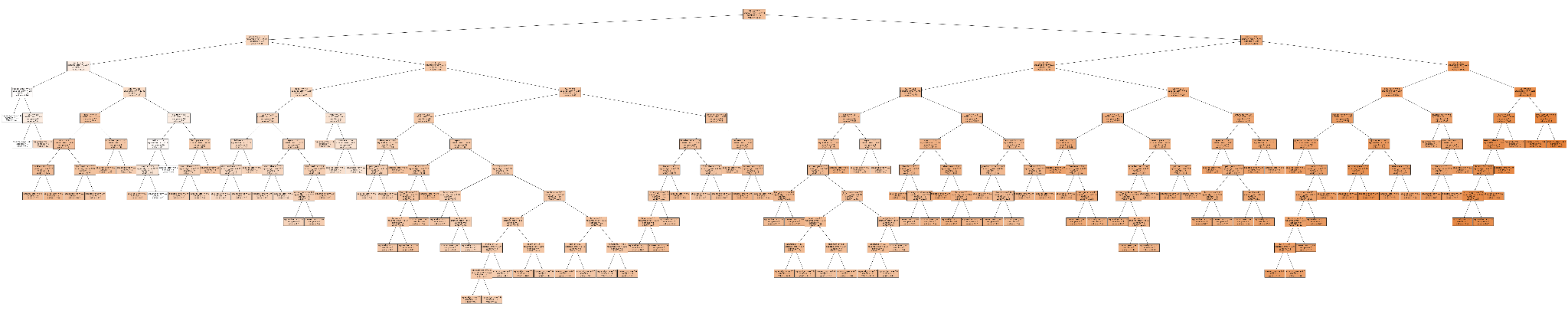
proportion=False,

precision=2)

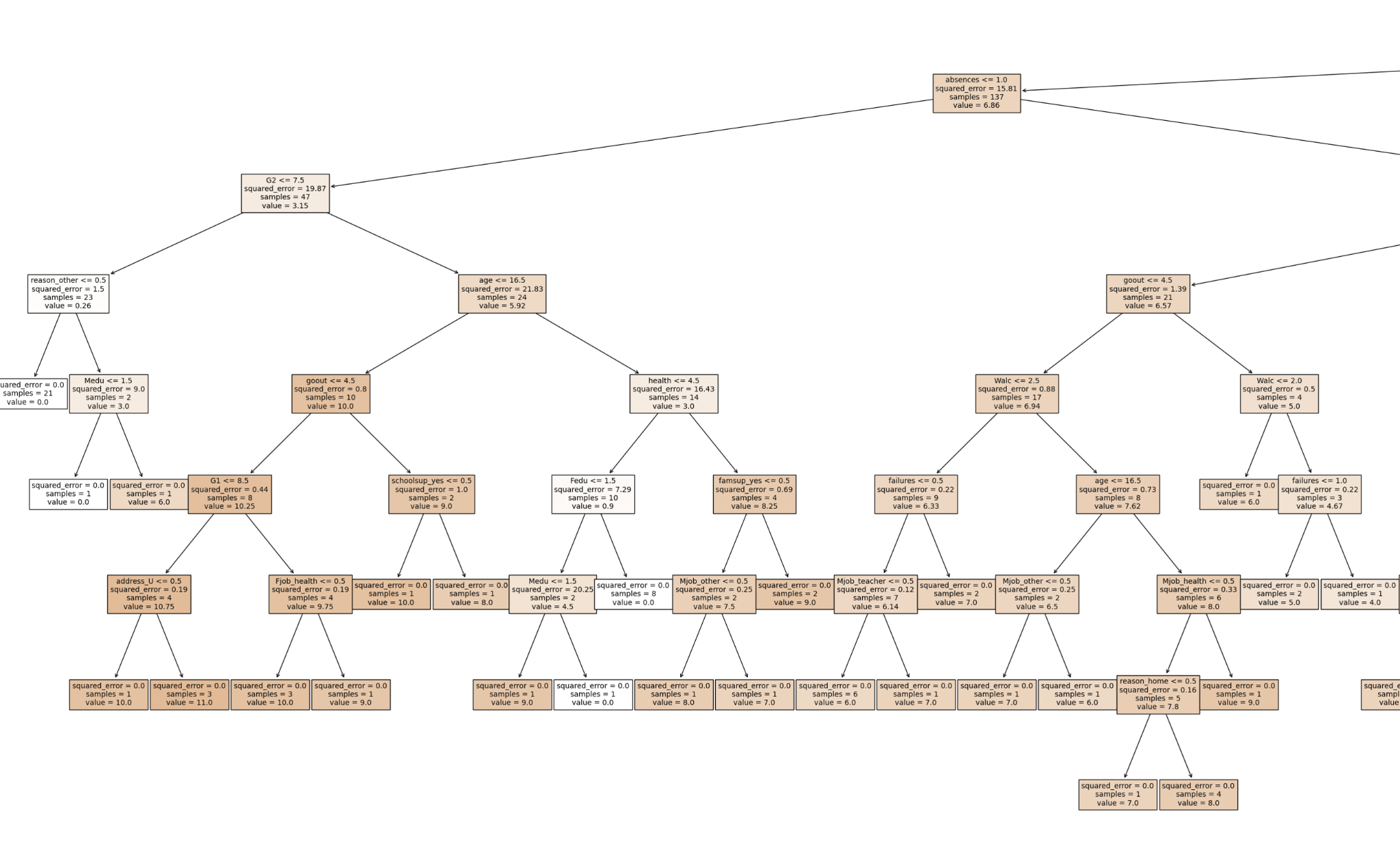
plt.savefig('decision\_tree\_wide.png', format='png', bbox\_inches='tight')

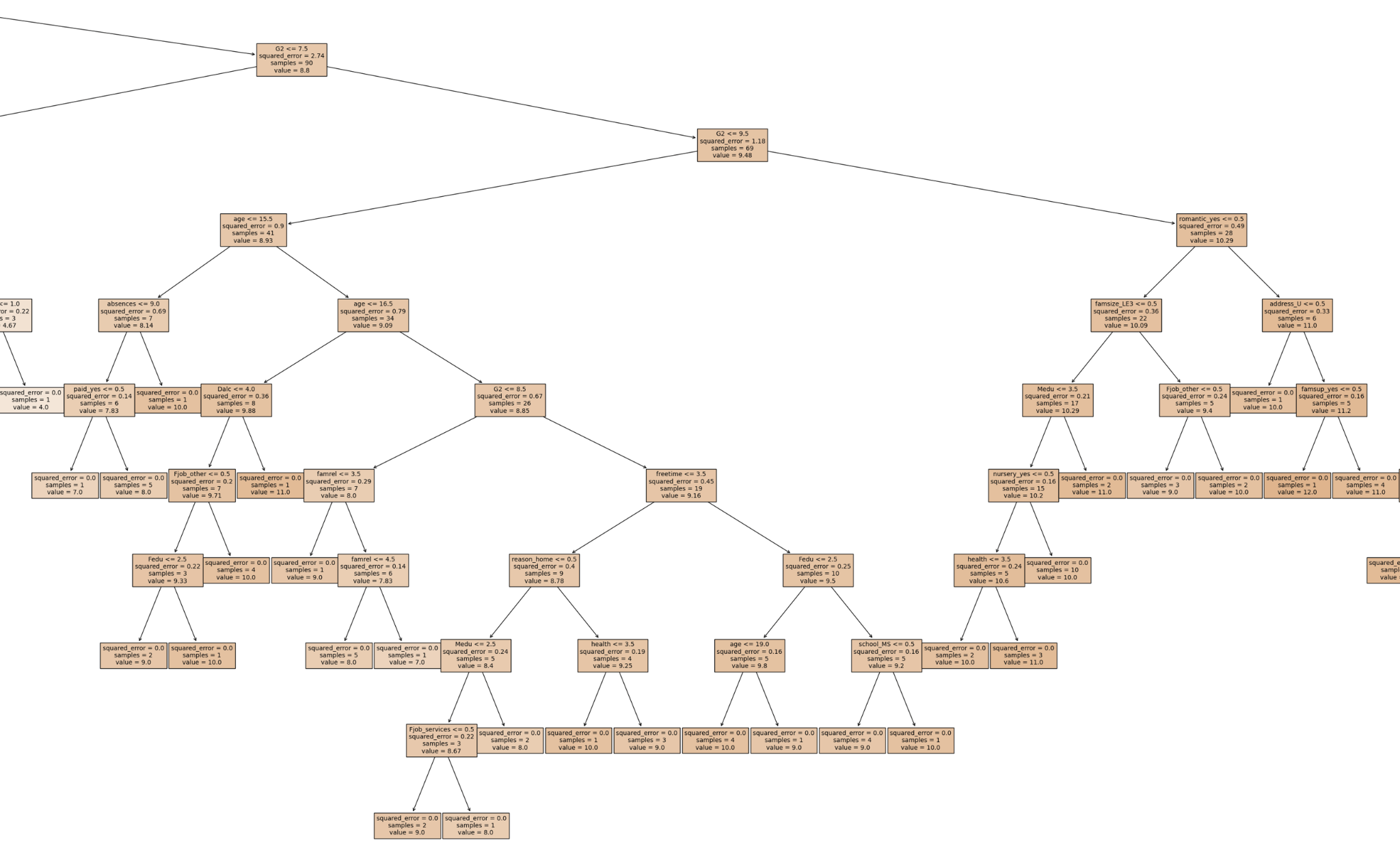
plt.show()

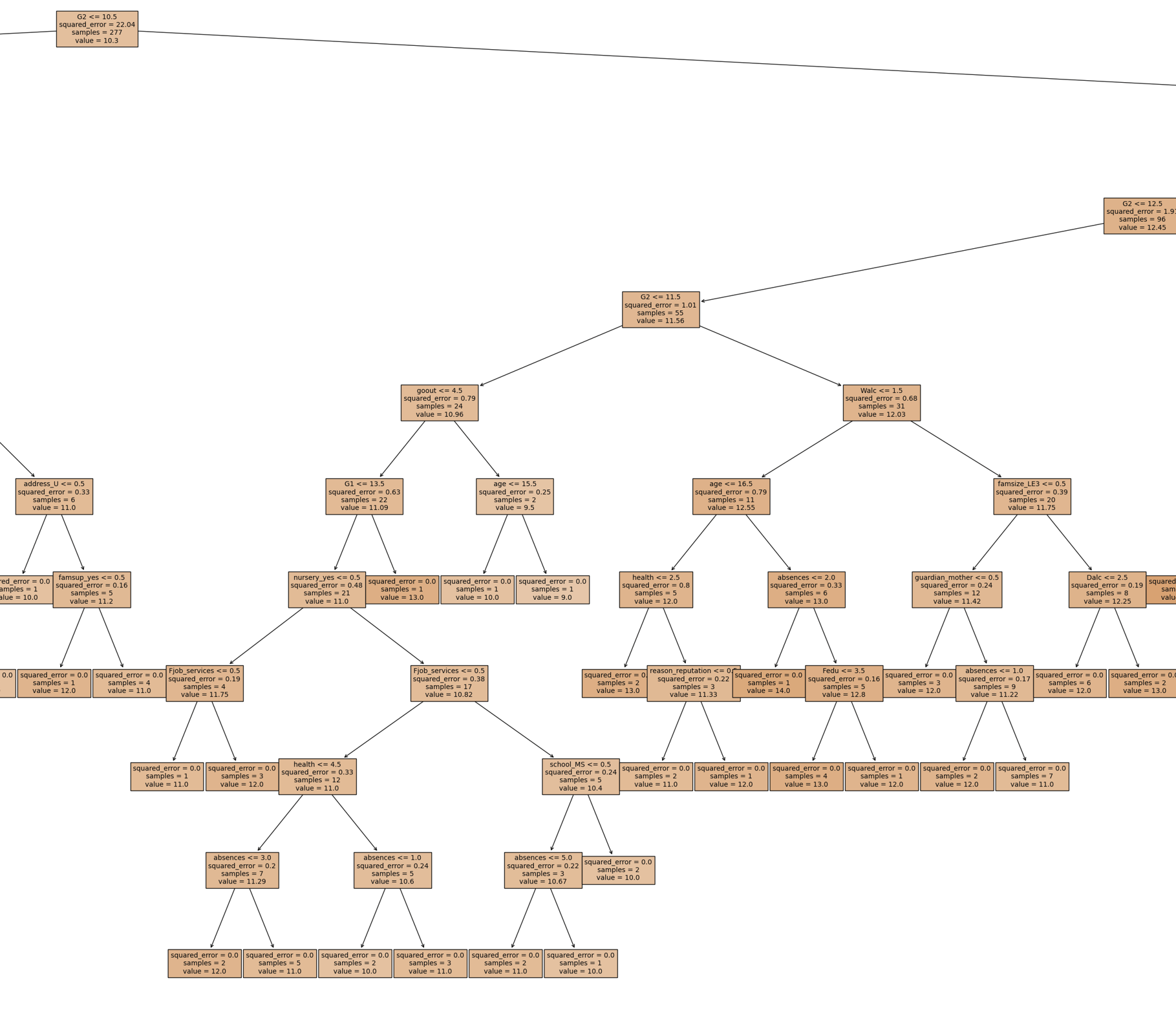
***Full Decision Tree***

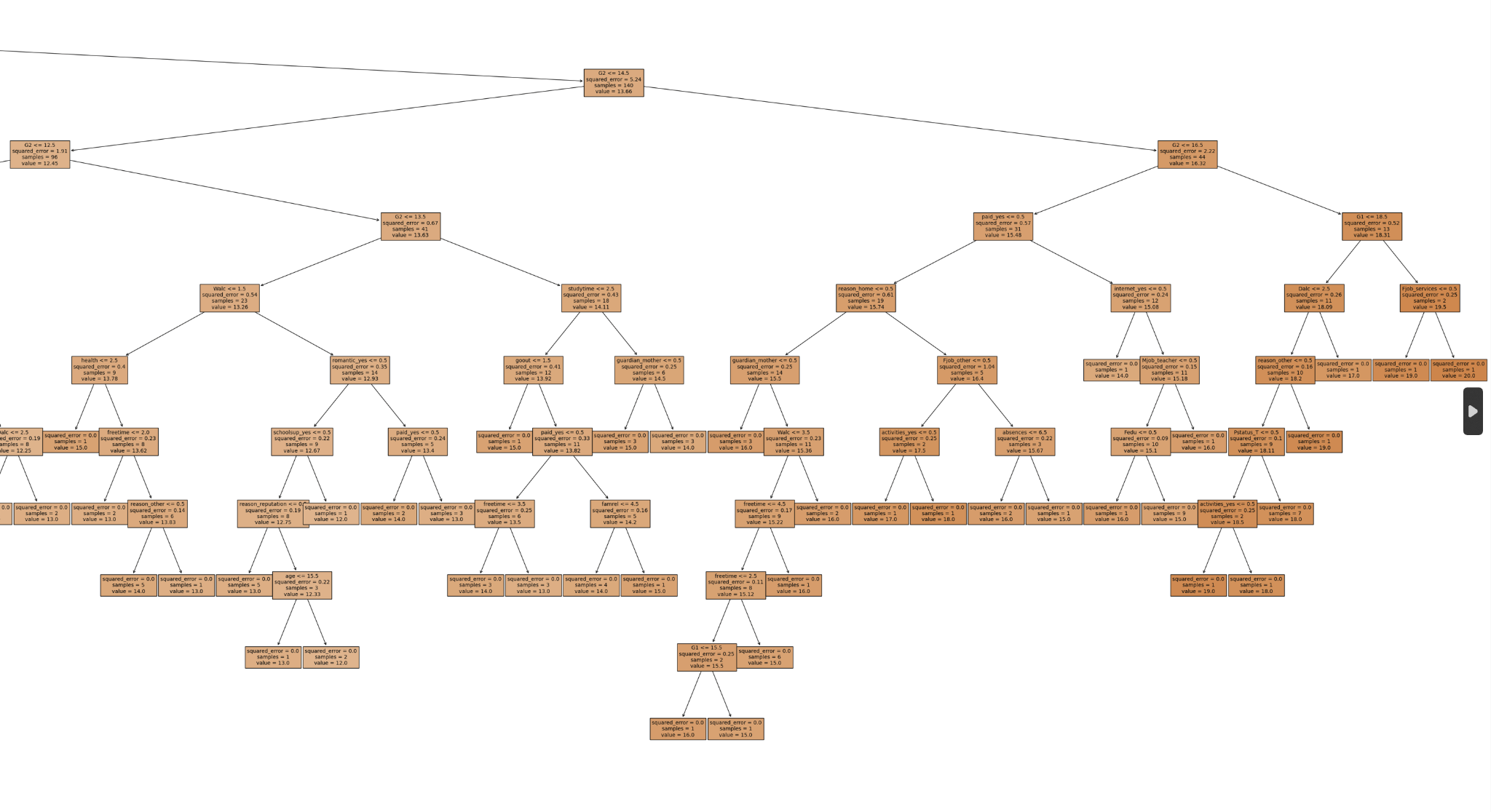


***Tree Broken Up into Parts for Readability***

******

******

******

******

precision recall f1-score support

0.0 0.64 0.88 0.74 8

4.0 0.00 0.00 0.00 0

5.0 0.00 0.00 0.00 5

6.0 0.67 0.57 0.62 7

7.0 0.10 1.00 0.18 1

8.0 0.25 0.15 0.19 13

9.0 0.00 0.00 0.00 6

10.0 0.50 0.33 0.40 18

11.0 0.38 0.36 0.37 14

12.0 0.09 0.11 0.10 9

13.0 0.50 0.57 0.53 7

14.0 0.33 0.38 0.35 8

15.0 0.40 0.60 0.48 10

16.0 0.33 0.20 0.25 5

17.0 0.00 0.00 0.00 4

18.0 0.40 0.67 0.50 3

19.0 0.00 0.00 0.00 2

accuracy 0.35 120

macro avg 0.27 0.34 0.28 120

weighted avg 0.34 0.35 0.34 120

[(0.7212207512492897, 'G2'), (0.1632908854220649, 'absences'), (0.050058895223000235, 'age'), (0.026651679833567123, 'health'), (0.009856072110825554, 'Medu'), (0.005843038545641484, 'Fedu'), (0.0033783905879605112, 'goout'), (0.0029454200941777973, 'reason\_other'), (0.002921060615034922, 'Walc'), (0.0015855366343050774, 'G1')]

All features: 1.0

Top 10 features: 0.9877517303158673

***Analysis of Decision Tree***

* The algorithm yielded a weighted accuracy of 0.34, or 34 % with 120 being the total number of samples used to compute the accuracy.
* The precision, which indicates the accuracy of positive predictions, was 0.34.
* The recall, aka sensitivity or true positive rate, which is the ratio of correctly predicted positive observations to all actual positives, was 0.35.
* The top 10 features based on importance were 2nd period grade, absences, age, health, mother's education, father's education, how often they go out, reason for attending the school, weekend alcohol consumption, and then they're 1st period grade.

***Using GridSearchCV to Tune Hyperparameters***

GridSearchCV is a tool provided by Scikit-learn that automates the process of tuning hyperparameters. It does this by performing an exhaustive search over a specified parameter grid for a model.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold

from sklearn.metrics import accuracy\_score, classification\_report

df = pd.read\_excel('dataset.xlsx', engine='openpyxl')

X = df[['G2', 'absences', 'age']]

y = df['G3']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

stratified\_cv = StratifiedKFold(n\_splits=5)

classifier = DecisionTreeClassifier(random\_state=42)

param\_grid = {

'criterion': ['gini', 'entropy'],

'max\_depth': [None, 10, 20, 30, 40],

'min\_samples\_split': [2, 10, 20, 40, 80],

'min\_samples\_leaf': [1, 2, 5, 10, 20]

}

grid\_search = GridSearchCV(estimator=classifier, param\_grid=param\_grid, cv=stratified\_cv, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

print("Best parameters:", grid\_search.best\_params\_)

print("Best training score:", grid\_search.best\_score\_)

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_test)

print("Test set accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred, zero\_division=0))

* max\_depth: The maximum depth of the tree. We tested values of None, 10, 20, and 30, to explore how tree depth affects complexity and potential overfitting.
* min\_samples\_split: Minimum number of samples required to split an internal node. We explored values like 2, 20, and 40 to determine the best trade-off between bias and variance.
* min\_samples\_leaf: Minimum number of samples required at a leaf node, with tested values of 1, 5, and 10 to see how it influences the granularity of the predictions.
* criterion: The function to measure the quality of a split, choosing between gini and entropy to see which criterion leads to better model performance.

Best parameters: {'criterion': 'entropy', 'max\_depth': None, 'min\_samples\_leaf': 5, 'min\_samples\_split': 20}

Best training score: 0.5094155844155843

Test set accuracy: 0.5

Classification Report:

precision recall f1-score support

0 0.67 1.00 0.80 8

5 0.00 0.00 0.00 5

6 0.43 0.86 0.57 7

7 0.00 0.00 0.00 1

8 0.45 0.38 0.42 13

9 0.20 0.17 0.18 6

10 0.58 0.61 0.59 18

11 0.86 0.43 0.57 14

12 0.40 0.22 0.29 9

13 0.29 0.71 0.42 7

14 0.57 0.50 0.53 8

15 0.64 0.90 0.75 10

16 0.00 0.00 0.00 5

17 0.00 0.00 0.00 4

18 0.43 1.00 0.60 3

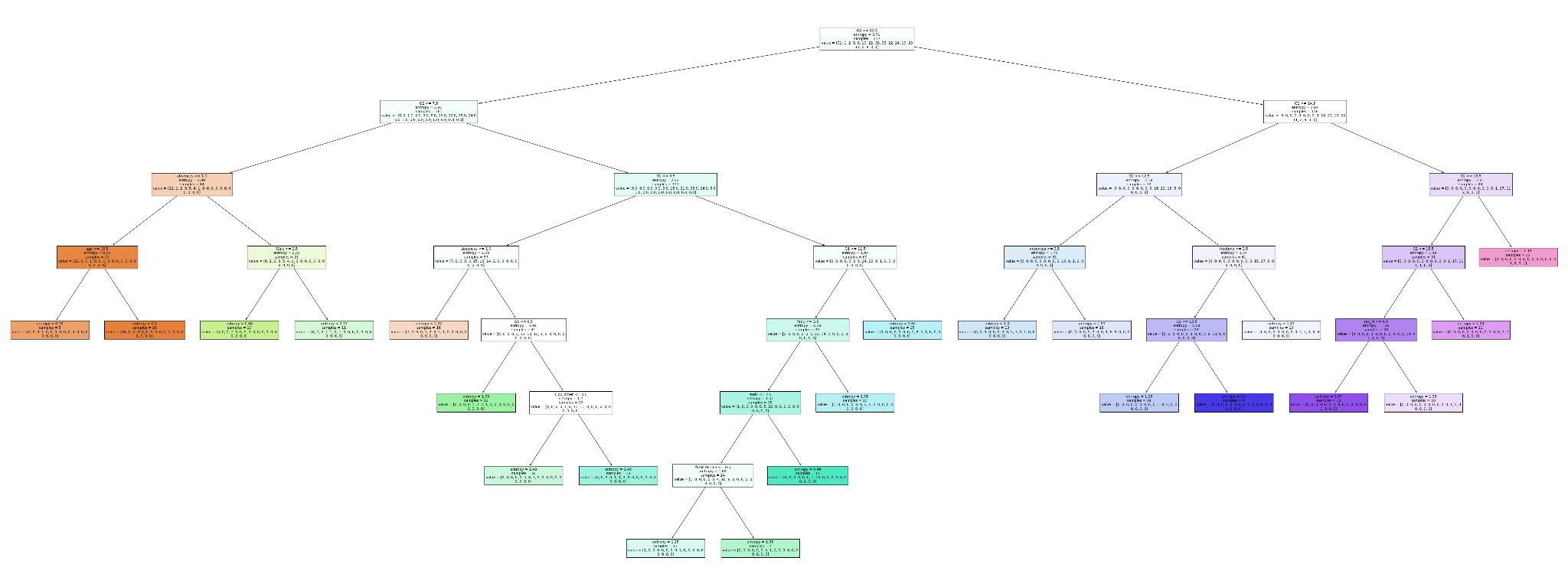
19 0.00 0.00 0.00 2

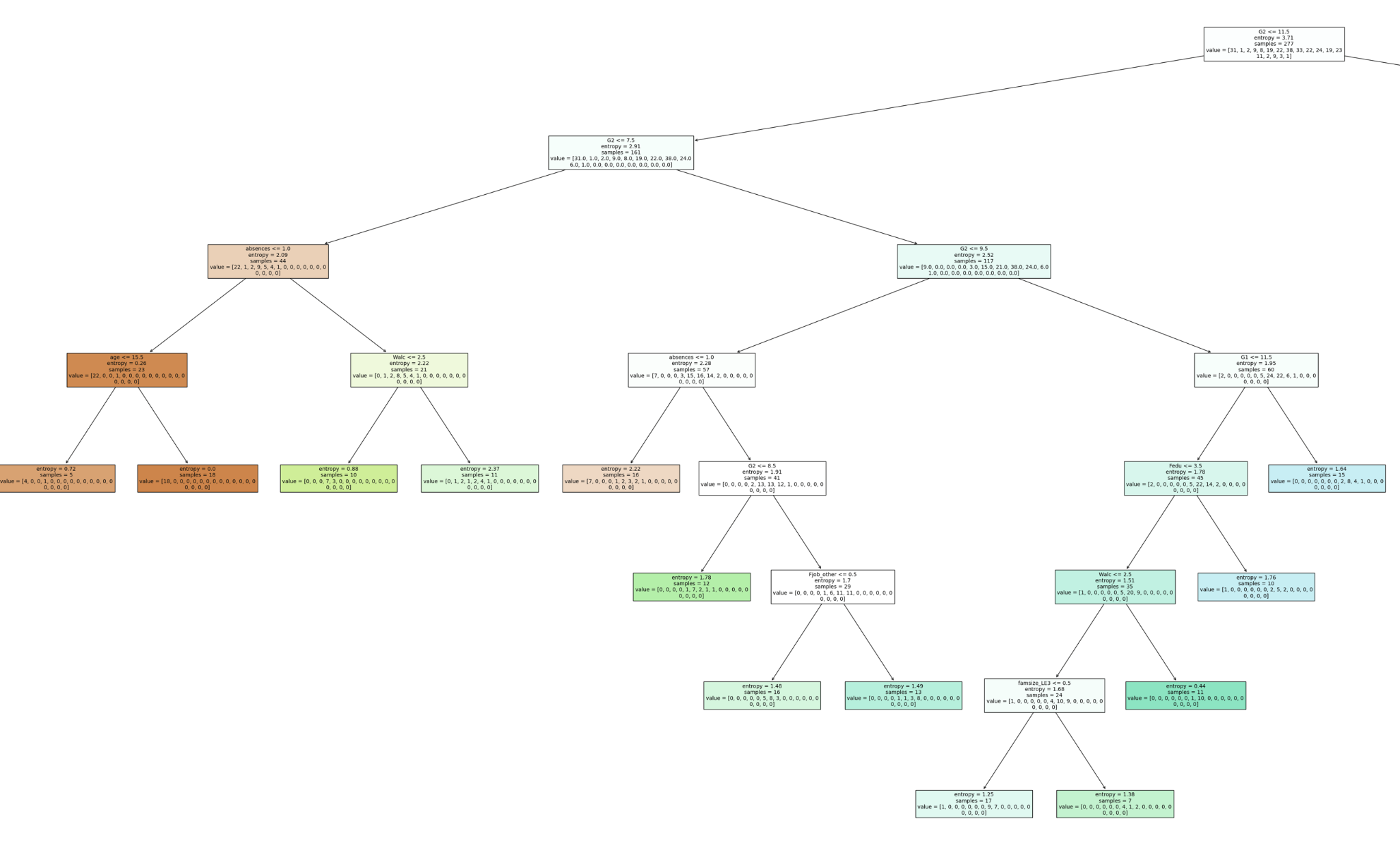
accuracy 0.50 120

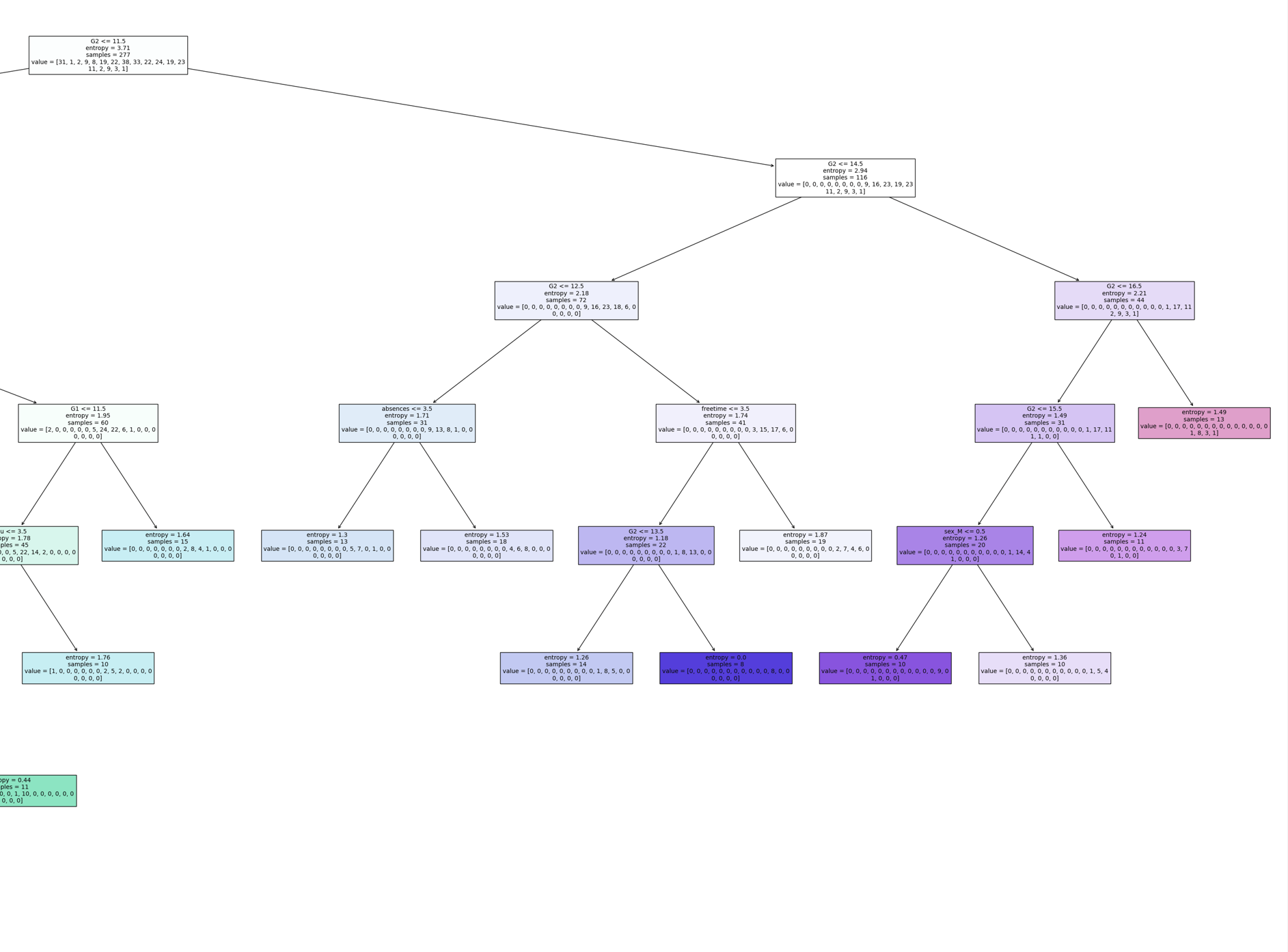
macro avg 0.35 0.42 0.36 120

weighted avg 0.47 0.50 0.46 120

The optimized decision tree achieved a training score of approximately 50% and a test accuracy of 50%. This is an improvement over the baseline model but still indicates possible underfitting or inherent limitations in the dataset's predictive power.







***Random Forest***

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

import matplotlib.pyplot as plt

import numpy as np

# load dataset

df\_x = pd.read\_excel('dataset.xlsx', engine='openpyxl')

df\_x.to\_csv('dataset.csv', index=False)

df = pd.read\_csv('dataset.csv')

df.fillna(0, inplace=True)

# split data into features and target

X = df.drop('G3', axis=1)

y = df['G3']

X = pd.get\_dummies(X, drop\_first=True)

model = RandomForestRegressor(n\_estimators=100)

model.fit(X, y)

importances = model.feature\_importances\_

features = X.columns

indices = np.argsort(importances)[::-1]

# plotting

plt.figure(figsize=(10, 8))

plt.title('Random Forest Feature Importance')

plt.bar(range(len(indices)), importances[indices], align='center')

plt.xticks(range(len(indices)), [features[i] for i in indices], rotation=90)

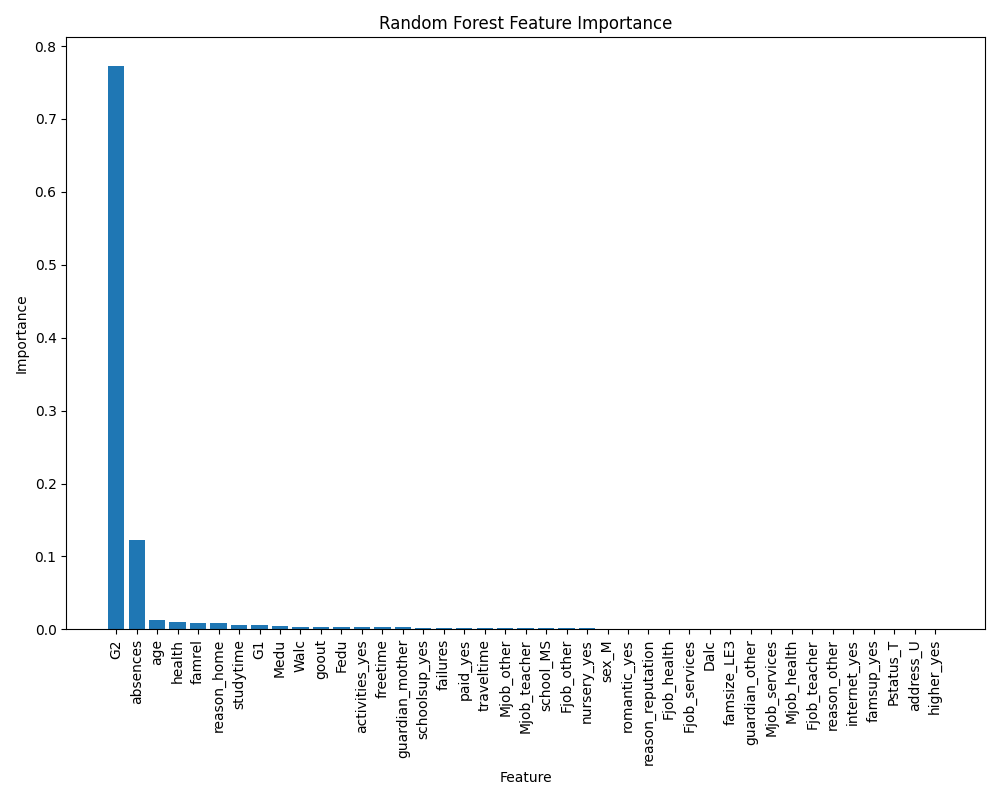
plt.xlabel('Feature')

plt.ylabel('Importance')

plt.tight\_layout() # Adjust layout to make room for rotated x-axis labels

plt.show()

plt.savefig(fname = 'Random Forest Feature Importance.png', transparent = True)



* The G2 feature, which represents the student's second period grade, is the most important determinant of G3. This indicates a strong correlation between students’ performance in the final grade period and their past performance, suggesting that past performance is a good predictor of future outcomes in this context.
* Though comparatively less important than the G2, number of absences is the next most important factor. This suggests that the number of times a student is absent can have some effect on the final grade, albeit less than the previous grade.
* Most of the other variables have relatively low importance scores, which may indicate that G2 considers that they have little individual predictive power on the G3 results This could be due to the predictive power of G2 a severe, affecting other variables analyzed under different conditions or when they overlap G2 is excluded from the model

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

from sklearn import tree

df = pd.read\_excel('dataset.xlsx', engine='openpyxl')

print(df.head())

print(df['G3'].value\_counts())

df['higher'] = df['higher'].map({'yes': 1, 'no': 0}).fillna(0)

categorical\_cols = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'internet', 'romantic']

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

X = df.drop('G3', axis=1)

y = df['G3']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

tree\_model = DecisionTreeClassifier(

criterion='entropy',

max\_depth=None,

min\_samples\_leaf=5,

min\_samples\_split=20,

random\_state=42

)

tree\_model.fit(X\_train, y\_train)

importances = tree\_model.feature\_importances\_

print(importances)

y\_pred = tree\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

feature\_importance = sorted(zip(tree\_model.feature\_importances\_, X.columns), reverse=True)

print("Top 10 features:", feature\_importance[:10])

print("All features:", sum(importances))

print("Top 10 features' importance sum:", sum([imp for imp, \_ in feature\_importance[:10]]))

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(

n\_estimators=100,

criterion='gini',

max\_depth=None,

min\_samples\_split=2,

min\_samples\_leaf=1,

max\_features='sqrt',

random\_state=42

)

# Train the RandomForest model

rf\_model.fit(X\_train, y\_train)

# Predict and evaluate

rf\_y\_pred = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_y\_pred)

print('Random Forest Accuracy:', rf\_accuracy)

print(classification\_report(y\_test, rf\_y\_pred))

print(confusion\_matrix(y\_test, rf\_y\_pred))

# Feature importance from Random Forest

rf\_importances = rf\_model.feature\_importances\_

sorted\_rf\_importance = sorted(zip(rf\_importances, X.columns), reverse=True)

print("Random Forest Top 10 features:", sorted\_rf\_importance[:10])

plt.figure(figsize=(8000/100, 3000/100), dpi=100)

tree\_plot = tree.plot\_tree(tree\_model,

filled=True,

feature\_names=X.columns,

fontsize=10,

proportion=False,

precision=2)

plt.savefig('random\_forest\_decision\_tree.png', format='png', bbox\_inches='tight', transparent = True)

plt.show()

***XGBoost***

import pandas as pd

import xgboost as xgb

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

import numpy as np

df = pd.read\_excel('dataset.xlsx', engine='openpyxl')

df['higher'] = df['higher'].map({'yes': 1, 'no': 0}).fillna(0)

categorical\_cols = [

'school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian',

'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'internet', 'romantic'

]

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

X = df.drop('G3', axis=1)

y = df['G3']

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(df['G3'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = xgb.XGBClassifier(

objective='multi:softprob',

n\_estimators=100,

learning\_rate=0.1,

max\_depth=3,

min\_child\_weight=1,

subsample=0.8,

colsample\_bytree=0.8,

seed=42

)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

importance = model.get\_booster().get\_score(importance\_type='weight')

sorted\_importance = sorted(importance.items(), key=lambda x: x[1], reverse=True)

features, values = zip(\*sorted\_importance)

colors = plt.cm.Purples(np.linspace(0.5, 1, len(values)))

fig, ax = plt.subplots(figsize=(12, 8))

y\_pos = np.arange(len(features))

bars = ax.barh(y\_pos, values, color=colors, align='center', edgecolor='black')

ax.set\_yticks(y\_pos)

ax.set\_yticklabels(features)

ax.invert\_yaxis()

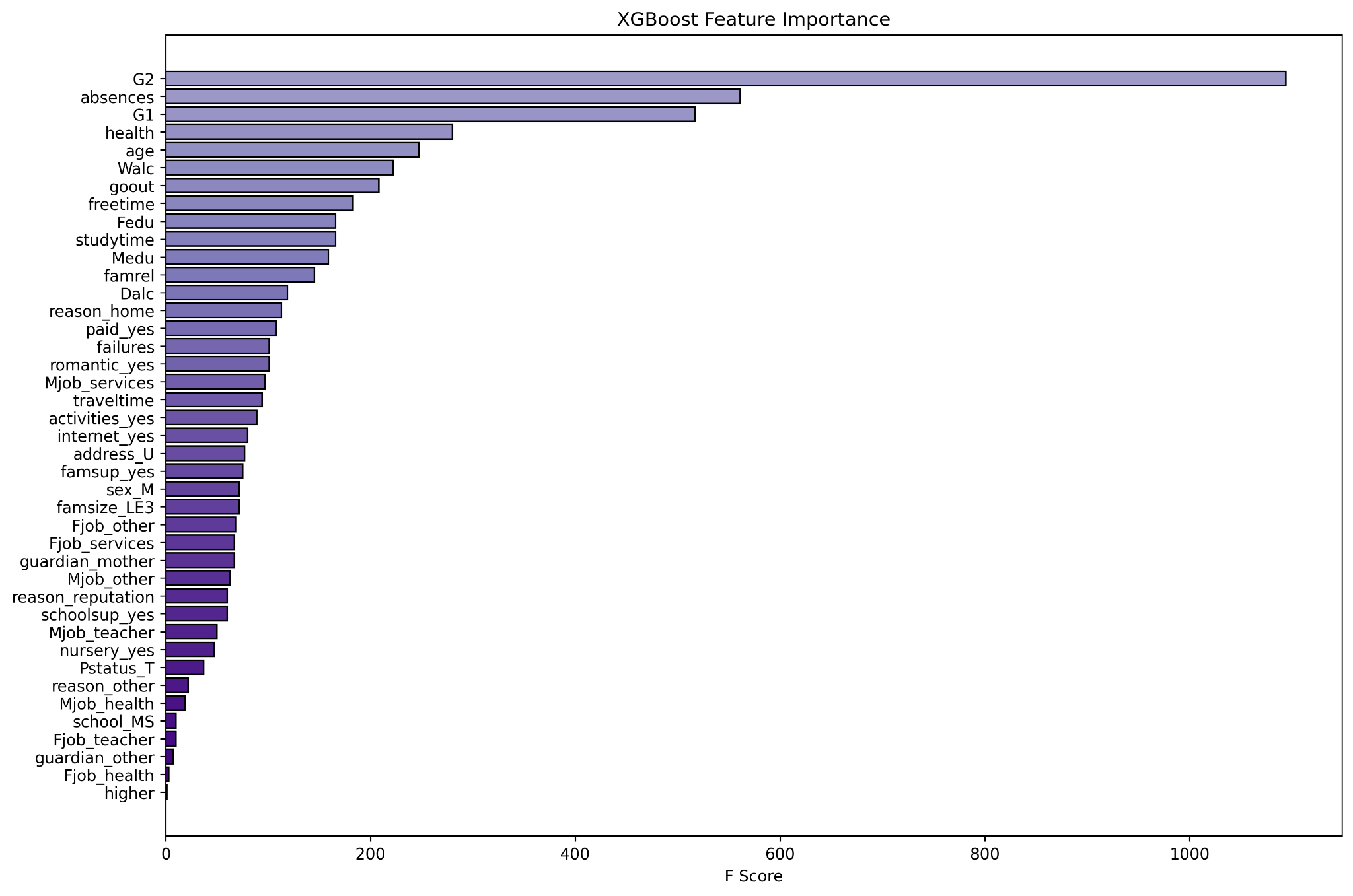
ax.set\_xlabel('F Score')

ax.set\_title('XGBoost Feature Importance')

plt.tight\_layout()

plt.savefig('feature\_importance\_corrected\_borders.png', format='png', dpi=300, bbox\_inches='tight', transparent = True)

plt.show()



* Again, G2 seems to have the highest F score, indicating it is the most significant predictor of the target variable in the dataset.
* Features like absences and G1 also show significant importance, unlike Random Forest, although less than G2. These features are used frequently by the model to make splits but are secondary to G2.
* Features towards the bottom of the list, such as Fjob\_health and guardian\_other, have very low F scores, meaning they are rarely used to make splits in the data and have minimal impact on the model's predictions.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import xgboost as xgb

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.preprocessing import LabelEncoder

df = pd.read\_excel('dataset.xlsx', engine='openpyxl')

print(df.head())

print(df['G3'].value\_counts())

df['higher'] = df['higher'].map({'yes': 1, 'no': 0}).fillna(0)

categorical\_cols = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'internet', 'romantic']

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

X = df.drop('G3', axis=1)

y = df['G3']

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.3, random\_state=42)

tree\_model = DecisionTreeClassifier(

criterion='entropy',

max\_depth=None,

min\_samples\_leaf=5,

min\_samples\_split=20,

random\_state=42

)

tree\_model.fit(X\_train, y\_train)

tree\_y\_pred = tree\_model.predict(X\_test)

tree\_accuracy = accuracy\_score(y\_test, tree\_y\_pred)

print('Decision Tree Accuracy:', tree\_accuracy)

print(classification\_report(y\_test, tree\_y\_pred))

print(confusion\_matrix(y\_test, tree\_y\_pred))

xgb\_model = xgb.XGBClassifier(

objective='multi:softprob',

n\_estimators=100,

learning\_rate=0.1,

max\_depth=3,

subsample=0.8,

colsample\_bytree=0.8,

random\_state=42,

num\_class=len(set(y\_encoded)) )

xgb\_model.fit(X\_train, y\_train)

xgb\_y\_pred = xgb\_model.predict(X\_test)

xgb\_accuracy = accuracy\_score(y\_test, xgb\_y\_pred)

print('XGBoost Accuracy:', xgb\_accuracy)

print(classification\_report(y\_test, xgb\_y\_pred))

print(confusion\_matrix(y\_test, xgb\_y\_pred))

plt.figure(figsize=(8000/100, 3000/100), dpi=100)

tree\_plot = plot\_tree(tree\_model,

filled=True,

feature\_names=X.columns,

fontsize=10,

proportion=False,

precision=2)

plt.savefig('decision\_tree.png', format='png', bbox\_inches='tight', transparent=True)

plt.show()

precision recall f1-score support

0 0.64 0.88 0.74 8

2 1.00 0.20 0.33 5

3 0.60 0.43 0.50 7

4 0.00 0.00 0.00 1

5 0.36 0.31 0.33 13

6 0.27 0.50 0.35 6

7 0.64 0.50 0.56 18

8 0.50 0.36 0.42 14

9 0.12 0.11 0.12 9

10 0.25 0.29 0.27 7

11 0.27 0.38 0.32 8

12 0.53 0.80 0.64 10

13 0.00 0.00 0.00 5

14 0.00 0.00 0.00 4

15 0.43 1.00 0.60 3

16 0.00 0.00 0.00 2

accuracy 0.41 120

macro avg 0.35 0.36 0.32 120

weighted avg 0.42 0.41 0.39 120

[[7 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]

[0 1 1 2 1 0 0 0 0 0 0 0 0 0 0 0]

[0 0 3 3 1 0 0 0 0 0 0 0 0 0 0 0]

[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]

[3 0 0 0 4 4 2 0 0 0 0 0 0 0 0 0]

[0 0 1 0 1 3 0 1 0 0 0 0 0 0 0 0]

[1 0 0 0 3 2 9 1 2 0 0 0 0 0 0 0]

[0 0 0 0 0 1 2 5 4 2 0 0 0 0 0 0]

[0 0 0 0 0 0 1 3 1 2 2 0 0 0 0 0]

[0 0 0 0 0 0 0 0 0 2 5 0 0 0 0 0]

[0 0 0 0 0 0 0 0 1 2 3 2 0 0 0 0]

[0 0 0 0 0 0 0 0 0 0 1 8 1 0 0 0]

[0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0]

[0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0]]