Problem Framing & Big Picture

1. Clearly communicate the problem and objective in business terms and how your solution will be used.

The objective of this project is to predict final grades of students, using this information we can identify which students require additional help to pass their subject. The business problem will be predicted by a list of each students attributes including family, age, and activites outside of school. A complete list will be given further into the project.

2. How should you frame this problem (supervised/unsupervised, online/offline, etc)? Briefly explain these terms since part of your audience is non-technical.

This is a supervised learning task because the model will be learning from labeled data. If the data wasn't labeled it would be considered unsupervised learning. Because the data set is static (does not change over time), it is considered offline and will not be performing real-time updates. If a project was online, it would be constantly making predictions such as stock market predictions.

3. Discuss the specific machine learning task that you are working on (regression/classification) and how it could solve the business problem. Briefly explain the difference since part of your audience is non-technical.

Making the choice depends on the data being used. Taking a glance at the CSV file provided, it shows that the data for our output target is discrete, meaning it is represented by integers, or, a number without decimal points. Because our target value is discrete, a Classification model will be used. If the value of the target attrubute had decimal placed (aka continuous data), a regression model would be used.

4. Identify the metrics that you will use to measure the model's performance.

The metrics used will be R2 Score, Precision, Recall, F1 Score and Accuracy. The R2 score will tell us how well the attributes are predicting our target attribute. Precision is used to see how well the data is classsifying our outcome. For example if we were trying to predict whether or not an e-mail was spam, precision would be used to classify this. Recall is a metric that determines how accurate our precision is. The F1 score combines recall and precision into one metric as their over-all performance. Accuracy is a metric used to tell us how well the entire model is doing at making our predictions.

5. Is there anything else that your director or board of directors need to know about this project?

The dataset is downloaded from UC Irvine's machine learning repository. The data is from two Portugese schools gathered from reports and questionnaires. The data is accurate as there are over 300 studnet submissions.

Get the Data

1. Correctly import your data (CodeGrade will assume that your data is in the same folder as your notebook just like with your other assignments)

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEr
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate
from sklearn.base import clone
from scipy import stats
```

+ above is a list of imports that will be used to perform our calculations

In [2]:
```
# Here our data is uploaded from a csv file in excel into our notebook
student_matrix = pd.read_csv("student-mat.csv")
```

2. *Check the size and type of data

In [3]:
```
student_matrix.size
# This number counts every individual entry
```

Out[3]:
```
13825
```

In [4]:
```
student_matrix.dtypes
#This tells us the datatype of every attribute. An object is either te
#Float64 is a number that includes a decimal up to 64. Int64 is a numb
```

Out[4]:
```
school          object
sex             object
age            float64
address         object
famsize         object
Pstatus         object
Medu             int64
Fedu             int64
Mjob            object
Fjob            object
reason          object
guardian        object
traveltime       int64
studytime        int64
failures         int64
schoolsup       object
famsup          object
paid            object
activities      object
nursery         object
higher          object
internet        object
romantic        object
famrel           int64
freetime         int64
goout            int64
Dalc             int64
Walc             int64
health           int64
absences_G1    float64
absences_G2    float64
absences_G3    float64
G1               int64
G2               int64
G3               int64
dtype: object
```

3. List the available features and their data descriptions so that your director/board of directors can understand your work

1. school - student's school ("GP" - Gabriel Pereira or "MS" - Mousinho da Silveira)

2. sex - student's sex ("F" - female or "M" - male)
3. age - student's age (numeric from 15 to 22)
4. address - student's home address type ("U" - urban or "R" - rural)
5. famsize - family size ("LE3" - less or equal to 3 or "GT3" - greater than 3)
6. Pstatus - parent's cohabitation status ("T" - living together or "A" - apart)
7. Medu - mother's education (0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
8. Fedu - father's education (0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
9. Mjob - mother's job ("teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
10. Fjob - father's job ("teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
11. reason - reason to choose this school (close to "home", school "reputation", "course" preference or "other")
12. guardian - student's guardian ("mother", "father" or "other")
13. traveltime - home to school travel time (1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14. studytime - weekly study time (1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15. failures - number of past class failures (n if 1<=n<3, else 4)
16. schoolsup - extra educational support (yes or no)
17. famsup - family educational support (yes or no)
18. paid - extra paid classes within the course subject (Math or Portuguese) (yes or no)
19. activities - extra-curricular activities (yes or no)
20. nursery - attended nursery school (yes or no)
21. higher - wants to take higher education (yes or no)
22. internet - Internet access at home (yes or no)
23. romantic - with a romantic relationship (yes or no)
24. famrel - quality of family relationships (from 1 - very bad to 5 - excellent)
25. freetime - free time after school (from 1 - very low to 5 - very high)
26. goout - going out with friends (from 1 - very low to 5 - very high)
27. Dalc - workday alcohol consumption (from 1 - very low to 5 - very high)
28. Walc - weekend alcohol consumption (from 1 - very low to 5 - very high)
29. health - current health status (from 1 - very bad to 5 - very good)
30. absences_G1 - number of school absences for G1 term (numeric)
31. absences_G2 - number of school absences for G2 term (numeric)
32. absences_G3 - number of school absences for G3 term (numeric)

these grades are related with the course math subject

33. G1 - first term grade (numeric: from 0 to 20)
34. G2 - second term grade (numeric: from 0 to 20)
35. G3 - final grade (numeric: from 0 to 20, ← this is your output target)

4. Identify the target or label attribute

The target attribute is G3 also known as the final grades. This will determine the students that need the most help.

In [5]:
```
Y = student_matrix["G3"]
X = student_matrix.drop(["G3"], axis=1)
#Here the data is split into the dependent variable (Y), also our targ
```

converting grades to US equivalent

In [6]:
```
## converting G3 column to US Grade equivalent
US_GRADE_EQUAL ={
    0: 'F',
    1: 'F',
    2: 'F',
    3: 'F',
    4: 'F',
    5: 'F',
    6: 'F',
    7: 'F',
    8: 'F',
    9: 'F',
    10: 'C',
    11: 'C',
    12: 'C',
    13: 'C',
    14: 'B',
    15: 'B',
    16: 'A',
    17: 'A',
    18: 'A+',
    19: 'A+',
    20: 'A+'
}

Y = Y.map(US_GRADE_EQUAL)

Y.value_counts()
```

Out[6]:
```
C     165
F     130
B      60
A      22
A+     18
Name: G3, dtype: int64
```

In [7]:
```
y_le = LabelEncoder()
Y = y_le.fit_transform(Y)
Y = pd.Series(Y)
```

In [8]:
```
y_le.classes_.tolist(), y_le.transform(y_le.classes_.tolist())
```

Out[8]:
```
(['A', 'A+', 'B', 'C', 'F'], array([0, 1, 2, 3, 4]))
```

Because we are going with a classification model, the Portugese grades are converted to the Alphabetical equivalent as per the three previous cells

In [9]:
```
student_matrix.select_dtypes(include="object").columns
#This is a list of attributes that are non-numeric aka categorical
```

Out[9]:
```
Index(['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjo
b',
       'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activiti
es',
       'nursery', 'higher', 'internet', 'romantic'],
      dtype='object')
```

5. Correctly split your data into a training and test set

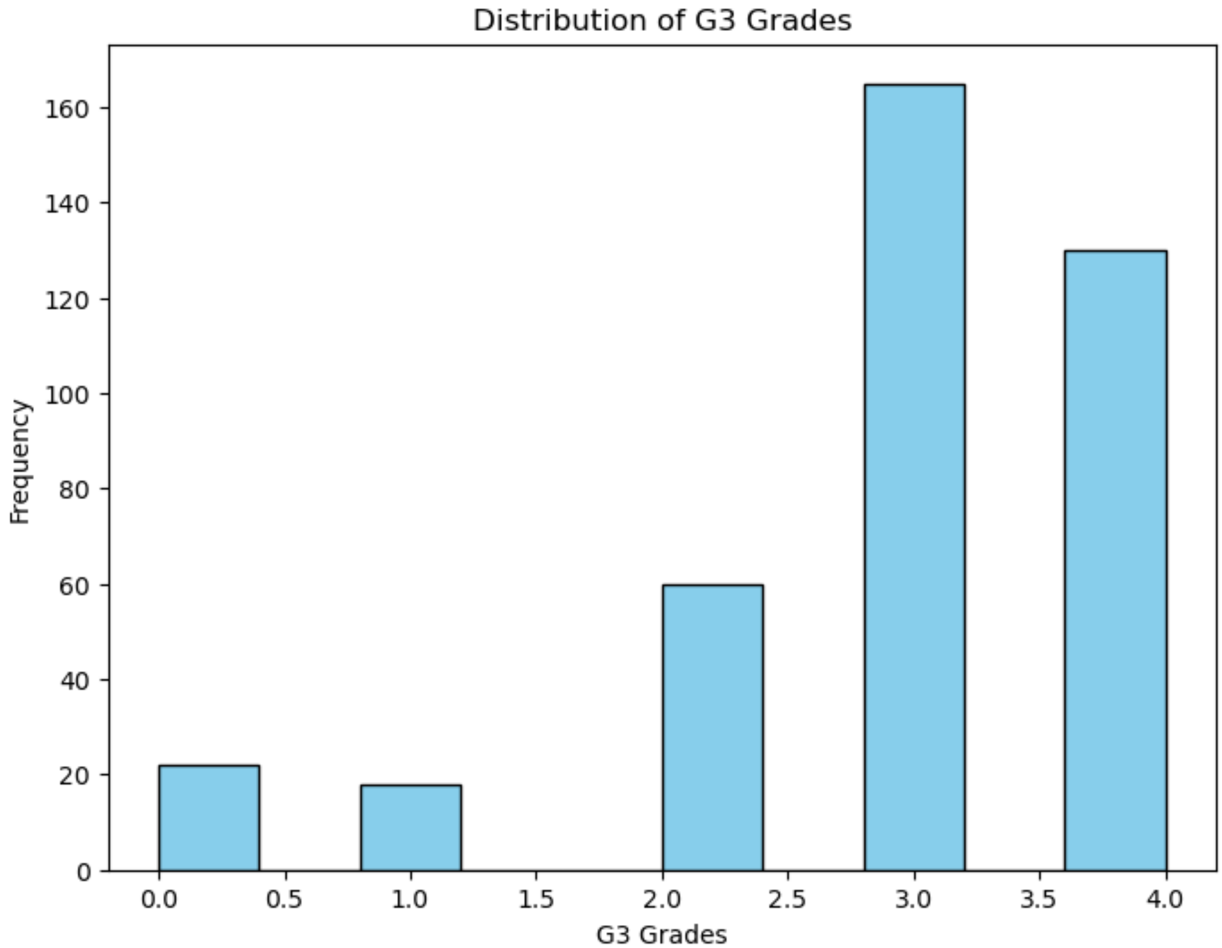
In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.
#The data is being split into a training set and test set. The Trainin
#test set evaluates the models perofrmance.
```

Explore the Data

1. *Thoroughly study the training set attributes and their characteristics

1.1 Analysing the Target Variable

In [11]:
```
# The graph shows the distribution of final grades
grades = Y

plt.figure(figsize=(8, 6))
plt.hist(grades, bins=10, color='skyblue', edgecolor='black')  # Adjus
plt.xlabel('G3 Grades')
plt.ylabel('Frequency')
plt.title('Distribution of G3 Grades')
plt.show()
```



1.2 Missing Value Analysis

In [12]:
```
X_train.isna().sum()
```

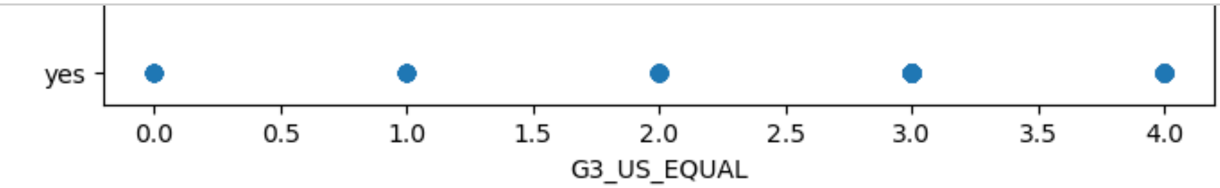
Out[12]:
```
school         0
sex            0
age           11
address        0
famsize        0
Pstatus        0
Medu           0
Fedu           0
Mjob           0
Fjob           0
reason         0
guardian       0
traveltime     0
studytime      0
failures       0
schoolsup      0
famsup         0
paid           0
activities     0
nursery        0
higher         0
internet       0
romantic       0
famrel         0
freetime       0
goout          0
Dalc           0
Walc           0
health         0
absences_G1   11
absences_G2   11
absences_G3   11
G1             0
G2             0
dtype: int64
```

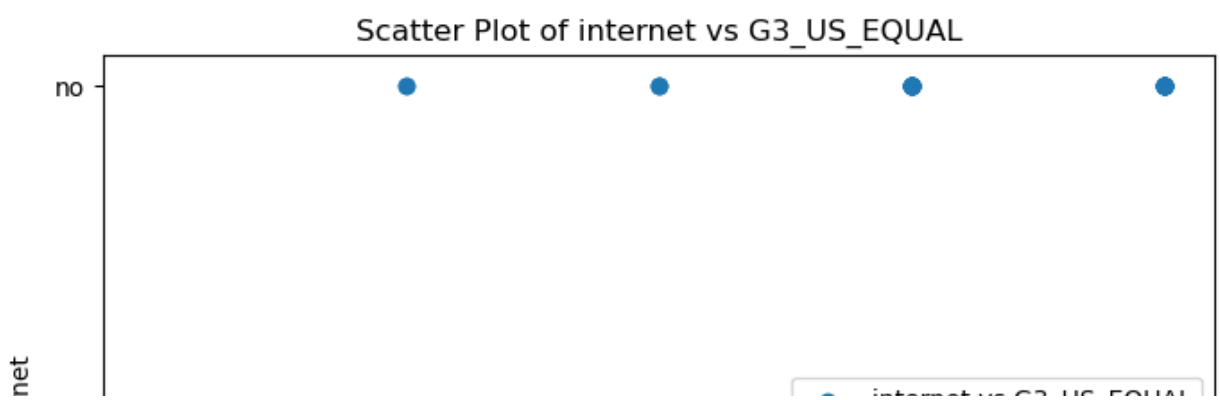
Here we see where there are missing values in the data. There are no columns with a very large missing data so we will remove them

2. *Produce at least four visualizations using your training data to assist in exploring the data. You should ensure that your visuals are informative and visually appealing, not purely using the default plots. Explain what each chart shows, why it’s important, and what insights did you obtain from the plots. (matplotlib and seaborn are the only packages available in CodeGrade)

1.1 Distribution of each variable against Target variable

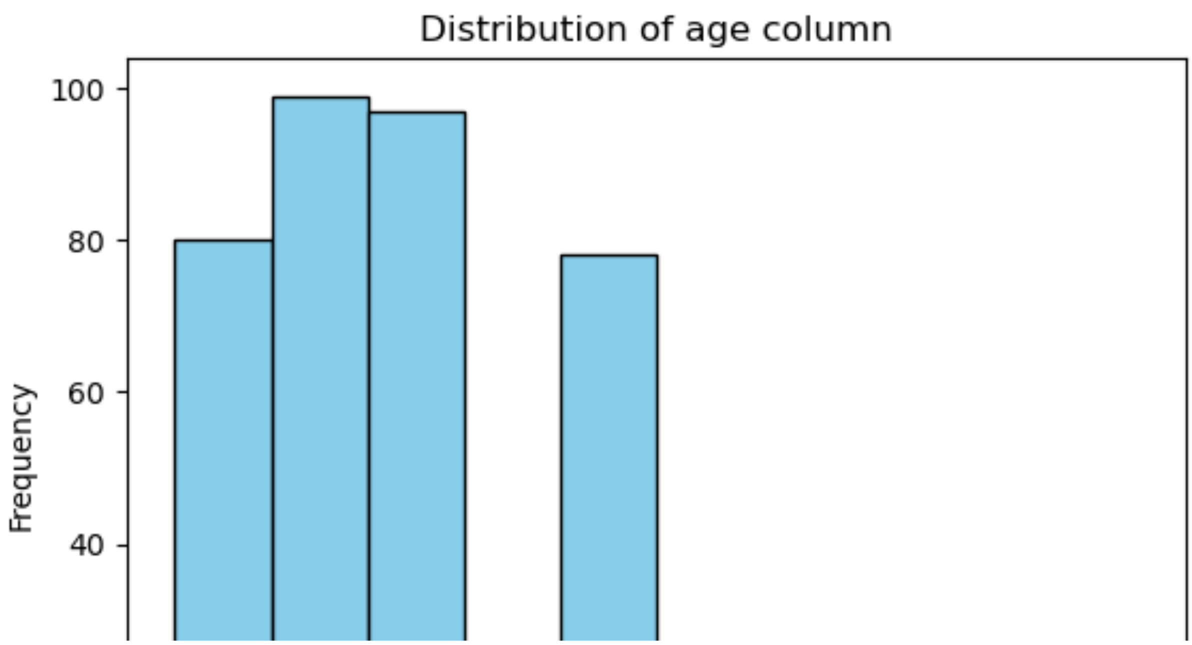
In [13]:
```
# The scatter plots below give a visualization of how each attribute c
for column in X_train.columns:
    plt.figure(figsize=(8, 5))
    plt.scatter(y_train, X_train[column], label=f'{column} vs G3_US_EQ
    plt.title(f'Scatter Plot of {column} vs G3_US_EQUAL')
    plt.xlabel("G3_US_EQUAL")
    plt.ylabel(column)
    plt.legend()
    plt.show()
```





1.2 Distribution analysis of each variable

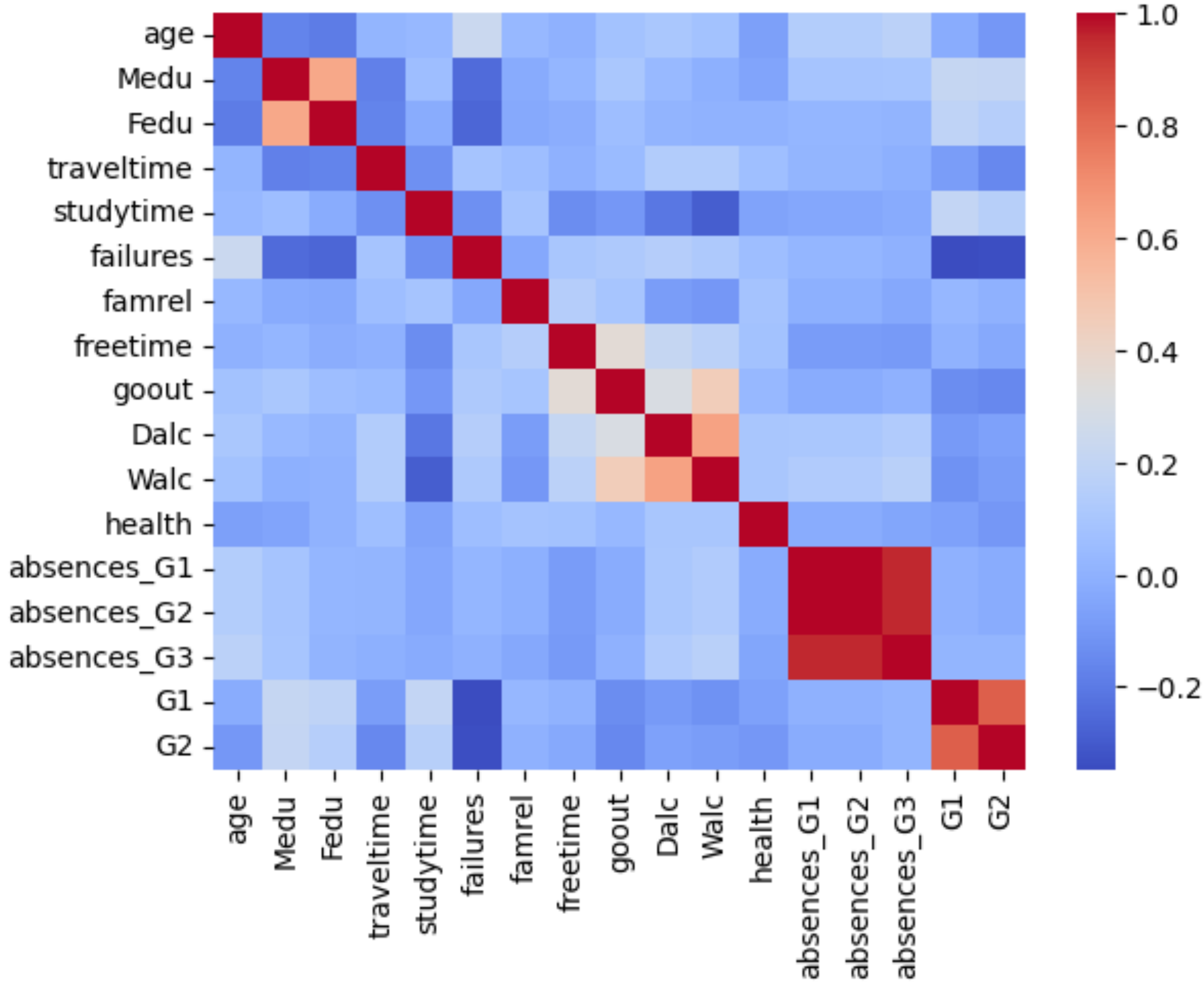
In [14]:
```
#Below is teh distribution of each varaiable. This will help us determ
#cleaning the data.
plt.figure(figsize=(10, 10))
print(X_train.columns)
for column in X_train.columns:
    plt.hist(student_matrix[column], bins=10, color='skyblue', edgecol
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title(f"""Distribution of {column} column""")
    plt.show()
```



3. *Study the correlations between attributes

In [15]:
```
sns.heatmap(X_train.corr(numeric_only=True), annot=False, cmap='coolwa
```

Out[15]: <Axes: >



As seen from the correlation plot, almost all the correlation boxes are approximately blue hence there is a very small amount of direct correlation or connection. We will have to engineer a few features to check

In [16]:
```
data = X_train.copy()
data["NumericTarget"] = y_train
categorical_columns = data.select_dtypes(include=['object']).columns

# Create subplots - setting up the plot grid
fig, axes = plt.subplots(nrows=len(categorical_columns), ncols=1, figs

# Ensure axes is an array even if there's only one plot
if len(categorical_columns) == 1:
    axes = [axes]

# Loop through categorical columns
for idx, column in enumerate(categorical_columns):
    # Perform ANOVA
    groups = [group['NumericTarget'].values for name, group in data.gr
    fvalue, pvalue = stats.f_oneway(*groups)
    print(f"ANOVA results for {column}: F-value = {fvalue}, P-value = 

    # Create a box plot
    sns.boxplot(x=column, y='NumericTarget', data=data, ax=axes[idx])
    axes[idx].set_title(f'Box Plot of NumericTarget by {column}\nF-val
    axes[idx].set_xlabel(column)
    axes[idx].set_ylabel('NumericTarget')

# Adjust layout
plt.tight_layout()
plt.show()
```

```
ANOVA results for school: F-value = 0.9960720406299977, P-value = 0.3
190313084941424
ANOVA results for sex: F-value = 1.323975720871076, P-value = 0.25075
52731088256
ANOVA results for address: F-value = 0.5158372509213973, P-value = 0.
4731568606833193
ANOVA results for famsize: F-value = 0.594665492083526, P-value = 0.4
412005386617871
ANOVA results for Pstatus: F-value = 0.9288782531047133, P-value = 0.
3358962213902889 6
ANOVA results for Mjob: F-value = 2.2708677132352717, P-value = 0.061
55046104697915
ANOVA results for Fjob: F-value = 1.0012114655397053, P-value = 0.407
08415789924146
ANOVA results for reason: F-value = 1.242743095439898, P-value = 0.29
4282599177183
ANOVA results for guardian: F-value = 1.0156572682472473, P-value =
0.3633547168525363
ANOVA results for schoolsup: F-value = 5.737732972644318, P-value =
```

Above an Anova test was performed and then graphed for visual interpretation. The F and P values were interpretted to see which groups are more connected to the target attribute. The higher the F value, the more signicance it is in correlation. The lower the P value the less likely the information registered was by chance.

Prepare the Data

1. Based on your exploration of the data above, perform feature selection to narrow down your data. (While not required, we would suggest that you create a function or custom transformer to handle this step so that you can more easily transform your test data.)

Based on the above findings the data, feature selection will be performed which will decide which are the most relevant features to train our model on. Using custom transformers, this will help reduce the dimentionality or range of our data so that the model will run easier and helps get rid of "noisy", irrelevant, or redundant data.

In [17]:
```
student_matrix.famsize.unique()
```

Out[17]:
```
array(['GT3', 'LE3'], dtype=object)
```

In [18]:
```
## write a sklearn transformer to drop na values
class DropNaTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        print("Transforming DropNaTransformer")
        return X.dropna()


test_df = pd.DataFrame({"a":[0, np.NaN], "b":[1, 2]})
print(test_df)
transformer = DropNaTransformer()
transformed_df = transformer.transform(test_df)
print("After droppping na values:")
print(transformed_df)
```

```
     a  b
0  0.0  1
1  NaN  2
Transforming DropNaTransformer
After droppping na values:
     a  b
0  0.0  1
```

In [19]:
```
class AbsencesFeatureGenerator(BaseEstimator, TransformerMixin):
    def __init__(self, remove=False):
        # Initialize the transformer with the option to remove the ori
        self.remove = remove

    def fit(self, X, y=None):
        # This transformer does not need to learn anything from the da
        # so the fit method just returns self.
        return self

    def transform(self, X):
        # Check if input is a DataFrame
        print("Transforming AbsencesFeatureGenerator")
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame")

        # Ensure the necessary columns are in the DataFrame
        required_columns = ['absences_G1', 'absences_G2', 'absences_G3
        if not all(col in X.columns for col in required_columns):
            raise ValueError(f"DataFrame must contain the following co

        # Create the new 'absences' column by summing the specified co
        X = X.copy()  # Create a copy of the DataFrame to avoid changi
        X['absences'] = X['absences_G1'] + X['absences_G2'] + X['absen

        # Remove the original columns if requested
        if self.remove:
            X.drop(['absences_G1', 'absences_G2', 'absences_G3'], axis

        return X

test_df = pd.DataFrame({"absences_G1":[0, 1], "absences_G2":[1, 2], "a
print(test_df)
transformer = AbsencesFeatureGenerator(remove=True)
transformed_df = transformer.transform(test_df)
print("After adding absences column:")
print(transformed_df)
```

```
   absences_G1  absences_G2  absences_G3
0            0            1            2
1            1            2            3
Transforming AbsencesFeatureGenerator
After adding absences column:
   absences
0         3
1         6
```

In [20]:
```
class FailuresCategorizer(TransformerMixin, BaseEstimator):
    def __init__(self, remove=False):
        # Initialize with the option to remove the original 'failures'
        self.remove = remove

    def fit(self, X, y=None):
        # No fitting process needed for this transformer, return self
        return self

    def transform(self, X):
        # Check if the dataframe contains the 'failures' column
        print("Transforming FailuresCategorizer")
        if 'failures' not in X.columns:
            raise ValueError("DataFrame must contain a 'failures' colu

        # Create a new column based on the 'failures' column
        X = X.copy()  # Make a copy to avoid changing the original dat
        X['failures_cat'] = X['failures'].apply(lambda x: 1 if x > 0 e

        if self.remove:
            # Remove the 'failures' column if specified
            X.drop(['failures'], axis=1, inplace=True)

        return X

test_df = pd.DataFrame({'failures': [0, 1, 2, 0, 3]})
print(test_df)
transformer = FailuresCategorizer(remove=True)
transformed_df = transformer.transform(test_df)
print("After adding failures_cat column:")
print(transformed_df)
```

```
   failures
0         0
1         1
2         2
3         0
4         3
Transforming FailuresCategorizer
After adding failures_cat column:
   failures_cat
0             0
1             1
2             1
3             0
4             1
```

In [21]:
```
class BaseFeatureTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, remove=False, name='BaseFeatureTransformer'):
        self.remove = remove
        self.name = name

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()  # Make a copy to avoid changing the original Dat

        print(f"Transforming {self.name}")
        if self.column_name not in X.columns:
            raise ValueError(f"DataFrame must contain the column '{sel

        X[self.new_column_name] = X[self.column_name].apply(self.categ
        if self.remove:
            X.drop([self.column_name], axis=1, inplace=True)

        return X
```

In [22]:
```
class AlcoholWeekendTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='AlcoholWeekendTransformer')
        self.column_name = 'Walc'
        self.new_column_name = 'Walc_cat'
        self.category_function = lambda walc: 0 if walc < 1.5 else (1
```

In [23]:
```
df = pd.DataFrame({'Walc': [0, 2, 4]})
transformer = AlcoholWeekendTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'Walc_cat': [0, 1, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming AlcoholWeekendTransformer
```

In [24]:
```
class HealthCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False, name='HealthCategoryTransformer')
        super().__init__(remove)
        self.column_name = 'health'
        self.new_column_name = 'health_cat'
        self.category_function = self.get_health_category

    def get_health_category(self, health):
        if health < 2.5:
            return 0
        elif health < 4.5:
            return 1
        else:
            return 2


df = pd.DataFrame({'health': [2, 5, 10]})
transformer = HealthCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'health_cat': [0, 2, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming BaseFeatureTransformer
```

In [25]:
```
class DailyAlcoholTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='DailyAlcoholTransformer')
        self.column_name = 'Dalc'
        self.new_column_name = 'Dalc_cat'
        self.category_function = lambda dalc: 0 if dalc < 1.5 else 1

df = pd.DataFrame({'Dalc': [0, 1, 3]})
transformer = DailyAlcoholTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'Dalc_cat': [0, 0, 1]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming DailyAlcoholTransformer
```

In [26]:

```
class GooutTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='GooutTransformer')
        self.column_name = 'goout'
        self.new_column_name = 'goout_cat'
        self.category_function = self.get_goout_category

    def get_goout_category(self,goout):
        if goout < 1.5:
            return 0
        elif goout < 3.5:
            return 2
        else:
            return 1

df = pd.DataFrame({'goout': [0, 3, 10]})
transformer = GooutTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'goout_cat': [0, 2, 1]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming GooutTransformer
```

In [27]:

```
class FreeTimeCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='FreeTimeCategoryTransformer')
        self.column_name = 'freetime'
        self.new_column_name = 'freetime_cat'
        self.category_function = self.get_freetime_category

    def get_freetime_category(self, freetime):
        if freetime < 1.5:
            return 0
        elif freetime < 4.5:
            return 1
        else:
            return 2

df = pd.DataFrame({'freetime': [0, 4, 10]})
transformer = FreeTimeCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'freetime_cat': [0, 1, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming FreeTimeCategoryTransformer
```

In [28]:
```
class FamilyRelativeTransormer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='FamilyRelativeTransormer')
        self.column_name = 'famrel'
        self.new_column_name = 'famrel_cat'
        self.category_function = self.get_famrel_category

    def get_famrel_category(self, famrel):
        if famrel <= 3:
            return 0
        else:
            return 1

df = pd.DataFrame({'famrel': [0, 4, 3]})
transformer = FamilyRelativeTransormer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'famrel_cat': [0, 1, 0]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming FamilyRelativeTransormer
```

In [29]:
```
class AgeCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='AgeCategoryTransformer')
        self.column_name = 'age'
        self.new_column_name = 'age_cat'
        self.category_function = self.get_age_category

    def get_age_category(self, age):
        if age <= 17:
            return 0
        elif age < 20:
            return 1
        else:
            return 2

df = pd.DataFrame({'age': [17, 19, 20]})
transformer = AgeCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'age_cat': [0, 1, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming AgeCategoryTransformer
```

In [30]:
```
class TravelTimeCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='TravelTimeCategoryTransformer')
        self.column_name = 'traveltime'
        self.new_column_name = 'traveltime_cat'
        self.category_function = self.get_traveltime_category

    def get_traveltime_category(self, traveltime):
        if traveltime < 1.5:
            return 0
        else:
            return 1

df = pd.DataFrame({'traveltime': [0, 3, 1]})
transformer = TravelTimeCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'traveltime_cat': [0, 1, 0]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming TravelTimeCategoryTransformer
```

In [31]:
```
class EduCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, column_name='fedu', remove=False):
        super().__init__(remove, name='EduCategoryTransformer')
        self.column_name = column_name
        self.new_column_name = f"""{self.column_name}_cat"""
        self.category_function = self.get_medu_fedu_category

    def get_medu_fedu_category(self, medu):
        if medu <=2:
            return 0
        elif medu <=3:
            return 1
        else:
            return 2

df = pd.DataFrame({'fedu': [0, 2, 3, 4]})
transformer = EduCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'fedu_cat': [0, 0, 1, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming EduCategoryTransformer
```

In [32]:
```
class StudyTimeCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='StudyTimeCategoryTransformer')
        self.column_name = 'studytime'
        self.new_column_name = 'studytime_cat'
        self.category_function = self.get_studytime_category

    def get_studytime_category(self, studytime):
        if studytime <=3:
            return 0
        elif studytime <=5:
            return 1
        else:
            return 2

df = pd.DataFrame({'studytime': [0, 4, 6]})
transformer = StudyTimeCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'studytime_cat': [0, 1, 2]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming StudyTimeCategoryTransformer
```

In [33]:
```
class FamsizeCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='FamsizeCategoryTransformer')
        self.column_name = 'famsize'
        self.new_column_name = 'famsize_cat'
        self.category_function = self.get_famsize_category

    def get_famsize_category(self, famsize):
        if famsize == 'LE3':
            return 0
        else:
            return 1

df = pd.DataFrame({'famsize': ['GT3', 'LE3', 'GT3']})
transformer = FamsizeCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'famsize_cat': [1, 0, 1]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming FamsizeCategoryTransformer
```

In [34]:
```
class MJobCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='MJobCategoryTransformer')
        self.column_name = 'Mjob'
        self.new_column_name = 'Mjob_cat'
        self.category_function = self.get_mjob_category

    def get_mjob_category(self, mjob):
        if mjob in ['at_home', 'teacher', 'health']:
            return 0
        else:
            return 1

df = pd.DataFrame({'Mjob': ['other', 'services', 'other', 'at_home']})
transformer = MJobCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'Mjob_cat': [1, 1, 1, 0]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming MJobCategoryTransformer
```

In [35]:
```
class FJobCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='FJobCategoryTransformer')
        self.column_name = 'Fjob'
        self.new_column_name = 'Fjob_cat'
        self.category_function = self.get_fjob_category

    def get_fjob_category(self, fjob):
        if fjob in ['at_home', 'teacher', 'health']:
            return 0
        else:
            return 1

df = pd.DataFrame({'Fjob': ['other', 'services', 'other', 'at_home']})
transformer = FJobCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'Fjob_cat': [1, 1, 1, 0]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming FJobCategoryTransformer
```

In [36]:
```
class GuardianCategoryTransformer(BaseFeatureTransformer):
    def __init__(self, remove=False):
        super().__init__(remove, name='GuardianCategoryTransformer')
        self.column_name = 'guardian'
        self.new_column_name = 'guardian_cat'
        self.category_function = self.get_guardian_category

    def get_guardian_category(self, guardian):
        if guardian in ['mother']:
            return 0
        else:
            return 1

df = pd.DataFrame({'guardian': ['mother', 'father', 'mother', 'father'
transformer = GuardianCategoryTransformer(remove=True)
transformed_df = transformer.transform(df)
expected_output = pd.DataFrame({'guardian_cat': [0, 1, 0, 1]})
pd.testing.assert_frame_equal(transformed_df, expected_output)
```

```
Transforming GuardianCategoryTransformer
```

2. Create at least one data pipeline to handle the data preparation steps

In [37]:
```
X_train.isna().sum()
```

Out[37]:
```
school         0
sex            0
age           11
address        0
famsize        0
Pstatus        0
Medu           0
Fedu           0
Mjob           0
Fjob           0
reason         0
guardian       0
traveltime     0
studytime      0
failures       0
schoolsup      0
famsup         0
paid           0
activities     0
nursery        0
higher         0
internet       0
romantic       0
famrel         0
freetime       0
goout          0
Dalc           0
Walc           0
health         0
absences_G1   11
absences_G2   11
absences_G3   11
G1             0
G2             0
dtype: int64
```

In [38]:
```
# A pipeline is made to streamline the transformers into the data maki

pipeline = Pipeline([
    ('failures', FailuresCategorizer(remove=True)),
    ('age_cat', AgeCategoryTransformer(remove=True)),
    ('traveltime_cat', TravelTimeCategoryTransformer(remove=True)),
    ('famrel_cat', FamilyRelativeTransormer(remove=True)),
    ('freetime_cat', FreeTimeCategoryTransformer(remove=True)),
    ('goout_cat', GooutTransformer(remove=True)),
    ('Dalc_cat', DailyAlcoholTransformer(remove=True)),
    ('health_cat', HealthCategoryTransformer(remove=True)),
    ('edu_medu_cat', EduCategoryTransformer(column_name='Medu', remove
    ('edu_fedu_cat', EduCategoryTransformer(column_name='Fedu', remove
    ('studytime_cat', StudyTimeCategoryTransformer(remove=True)),
    ('famsize_cat', FamsizeCategoryTransformer(remove=True)),
    ('mjob_cat', MJobCategoryTransformer(remove=True)),
    ('fjob_cat', FJobCategoryTransformer(remove=True)),
    ('guardian_cat', GuardianCategoryTransformer(remove=True)),
])
```

3. Fill in missing values or drop the rows or columns with missing values inside your pipeline

In [39]:
```
pipeline.steps.append(('na_removal', DropNaTransformer()))
```

4. Create a custom transformer in your pipeline that:

- creates a new column in the data that sums the absences_G1, absences_G2, and absences_G3 data and then drops those three columns.
- has a parameter that when equal to True, drops the G1 and G2 columns, and when False, leaves the columns in the data

In [40]:
```
pipeline.steps.append(('absences', AbsencesFeatureGenerator(remove=Tru
```

In [41]:
```
class G1G2RemovalTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, remove=False):
        self.remove = remove
        self.columns = ['G1', 'G2']

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        if self.remove:
            X = X.drop(self.columns, axis=1)
        return X

testing_frame  = pd.DataFrame({
    'G1': [1, 2, 3],
    'G2': [4, 5, 6],
    'G3': [7, 8, 9],
})

transformer = G1G2RemovalTransformer(remove=True)
transformed_frame = transformer.fit_transform(testing_frame)
print(transformed_frame)

## we will use this transformer at the very end in the pipeline to rem
```

```
   G3
0   7
1   8
2   9
```

5. Perform feature scaling on continuous numeric data in your pipeline

In [42]:
```
#Feature scaling is used on our numeric data so they may be on a simil
#This makes it easier for the features to calculate.
class MinMaxScalerTransformer(TransformerMixin, BaseEstimator):
    def __init__(self):
        self.numeric_columns = None
        self.scaler = MinMaxScaler()

    def fit(self, X, y=None):
        self.numeric_columns = X.select_dtypes(include='number').colum
        self.scaler.fit(X[self.numeric_columns])
        return self

    def transform(self, X):
```

```
        X = X.copy()
        print("Transforming MinMaxScalerTransformer")
        X[self.numeric_columns] = self.scaler.transform(X[self.numeric
        return X



testing_frame = pd.DataFrame({
    'a': [1, 2, 3],
    'b': [4, 5, 6],
    'c': [7, 8, 9],
})

transformer = MinMaxScalerTransformer()
transformed_frame = transformer.fit_transform(testing_frame)
print(transformed_frame)

pipeline.steps.append(('scaler', MinMaxScalerTransformer()))

test_df = pipeline.fit_transform(X_train)
```

```
Transforming MinMaxScalerTransformer
     a    b    c
0  0.0  0.0  0.0
1  0.5  0.5  0.5
2  1.0  1.0  1.0
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
```

6. Ordinal encode features that are either binary or that are ordinal in nature; and/or one-hot encode nominal or categorical data in a pipeline

In [43]:
```
#OneHotEncoding turns our categorical data with two values into a bina
class OneHotEncodingTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, remove=False):
        self.remove = remove
        self.encoders = None

    def fit(self, X, y=None):
        self.columns = ["health_cat","Dalc_cat","goout_cat","freetime_
        self.encoders ={col: OneHotEncoder(handle_unknown='ignore') fo
        # Fitting the encoders
        for col in self.encoders:
            self.encoders[col].fit(X[[col]])
        return self

    def transform(self, X):
        X = X.copy()
        print("Transforming OneHotEncodingTransformer")

        # Transforming the data
        for col in self.columns:
            encoded = self.encoders[col].transform(X[[col]]).toarray()
            encoded_df = pd.DataFrame(encoded, columns=[f"{col}_{cat}"
            X = pd.concat([X, encoded_df], axis=1)

        return X


#OneHotEncodingTransformer(remove=True).fit_transform(test_df)["Dalc_c

# ohe = OneHotEncoder().fit(test_df[["Dalc_cat"]])
# ohe.transform(test_df[["Dalc_cat"]]).toarray()
pipeline.steps.append(('one_hot_encoder', OneHotEncodingTransformer(re
```

In [44]:
```
test_df.Dalc_cat.value_counts()
```

Out[44]:
```
0.0    209
1.0     96
Name: Dalc_cat, dtype: int64
```

In [45]:
```
X_train
```

Out[45]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetim
181	GP	M	16.0	U	GT3	T	3	3	services	other	...	
194	GP	M	16.0	U	GT3	T	2	3	other	other	...	
173	GP	F	16.0	U	GT3	T	1	3	at_home	services	...	
63	GP	F	16.0	U	GT3	T	4	3	teacher	health	...	
253	GP	M	16.0	R	GT3	T	2	1	other	other	...	
...	...	...	...	...	...	...	...	...	...	...	...	
71	GP	M	15.0	U	GT3	T	4	2	other	other	...	
106	GP	F	15.0	U	GT3	T	2	2	other	other	...	
270	GP	F	19.0	U	GT3	T	3	3	other	services	...	
348	GP	F	17.0	U	GT3	T	4	3	health	other	...	
102	GP	M	15.0	U	GT3	T	4	4	services	other	...	

316 rows × 34 columns

7. Create a Column Transformer to transform your numeric and categorical data

In [46]:
```
## create a transformer which converts only categorical columns to num
class CategoricalToNumericalTransformer(BaseEstimator, TransformerMixi
    def fit(self, X, y=None):
        # Get the categorical columns
        self.categorical_columns_ = X.select_dtypes(include='object').
        self.le_dict = {}
        for col in self.categorical_columns_:
            le = LabelEncoder()
            le = le.fit(X[col])
            self.le_dict[col] = le
        return self

    def transform(self, X):
        # Apply LabelEncoder to categorical columns
        print("Transforming CategoricalToNumericalTransformer")
        X_cat = X.copy()
        for col in self.categorical_columns_:
            X_cat[col] = self.le_dict[col].transform(X_cat[col])
        return X_cat

# Sample data
testing_frame = pd.DataFrame({
    'school': ['GP', 'GP', 'MS', 'GP', 'GP']
})

# Creating and using the transformer
transformer = CategoricalToNumericalTransformer()
transformed_frame = transformer.fit_transform(testing_frame)
print(transformed_frame)

pipeline.steps.append(('cat_to_num_encoder', CategoricalToNumericalTra
```

```
Transforming CategoricalToNumericalTransformer
   school
0       0
1       0
2       1
3       0
4       0
```

8. Correctly transform your training data using the above data preparation steps and pipelines. You should have two distinct sets of transformed training data: one containing the G1/G2 columns and another without the G1/G2 columns.

In [47]:
```
pipe_no_g1_g2 = clone(pipeline)
pipe_no_g1_g2.steps.append(('g1_g2_removal', G1G2RemovalTransformer(re

X1_train_encoded = pipeline.fit_transform(X_train)
X1_not_G1_G2 = pipe_no_g1_g2.fit_transform(X_train)

y_train_clean = y_train.loc[X1_train_encoded.index]
```

```
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
Transforming OneHotEncodingTransformer
Transforming CategoricalToNumericalTransformer
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
Transforming OneHotEncodingTransformer
Transforming CategoricalToNumericalTransformer
```

9. *Output the shape of your two transformed training sets to show your custom transformer correctly removed the two columns

In [48]:
```
print(X1_train_encoded.shape)
print(X1_not_G1_G2.shape)
```

```
(305, 45)
(305, 43)
```

Shortlist Promising Models

1. Fit three or more promising models to your data using your transformed data

The models being used are best use for Classification.

In [49]:
```
y_train_clean = y_train.loc[X1_train_encoded.index]
```

In [50]:
```
randomForestNotG1G2 = RandomForestClassifier()
randomForestNotG1G2.fit(X1_not_G1_G2, y_train_clean)

randomForest = RandomForestClassifier()
randomForest.fit(X1_train_encoded, y_train_clean)

print(randomForestNotG1G2.score(X1_not_G1_G2, y_train_clean))
print(randomForest.score(X1_train_encoded, y_train_clean))


## testing

X_test_encoded = pipeline.transform(X_test)
y_test_clean = y_test.loc[X_test_encoded.index]
X_test_not_G1_G2 = X_test_encoded.drop(["G1","G2"], axis=1)

accuracy_rf = accuracy_score(y_test_clean, randomForest.predict(X_test
print("Accuracy", accuracy_rf)
precision_rf = precision_score(y_test_clean, randomForest.predict(X_te
print("Precision", precision_rf)
recall_rf = recall_score(y_test_clean, randomForest.predict(X_test_enc
print("Recall", recall_rf)
f1_rf = f1_score(y_test_clean, randomForest.predict(X_test_encoded) ,
print("F1", f1_rf)
r2_score_rf = r2_score(y_test_clean, randomForest.predict(X_test_encoc
```

```
print("r2_score", r2_score_rf)


accuracy_rf_notG1G2 = accuracy_score(y_test_clean, randomForestNotG1G2
print("Accuracy", accuracy_rf_notG1G2)
precision_rf_notG1G2 = precision_score(y_test_clean, randomForestNotG1
print("Precision", precision_rf_notG1G2)
recall_rf_notG1G2 = recall_score(y_test_clean, randomForestNotG1G2.pre
print("Recall", recall_rf_notG1G2)
f1_rf_notG1G2 = f1_score(y_test_clean, randomForestNotG1G2.predict(X_t
print("F1", f1_rf_notG1G2)
r2_score_rf_notG1G2 = r2_score(y_test_clean, randomForestNotG1G2.predi
print("r2_score", r2_score_rf_notG1G2)
```

```
1.0
1.0
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
Transforming OneHotEncodingTransformer
Transforming CategoricalToNumericalTransformer
Accuracy 0.7763157894736842
Precision 0.6508403361344538
Recall 0.6975
F1 0.6729559748427673
r2_score 0.6151672162034856
Accuracy 0.3684210526315789
Precision 0.2534366576819407
Recall 0.2375
F1 0.21923076923076926
r2_score 0.006594441827602493
```

```
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
```

```
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [51]:
```
decisionTree = DecisionTreeClassifier()

decisionTree.fit(X1_train_encoded, y_train_clean)


decisionTreeNotG1G2 = DecisionTreeClassifier()
decisionTreeNotG1G2.fit(X1_not_G1_G2, y_train_clean)

## testing

accuracy_dt = accuracy_score(y_test_clean, decisionTree.predict(X_test

print(accuracy_dt)
precision_dt = precision_score(y_test_clean, decisionTree.predict(X_te

print(precision_dt)

recall_dt = recall_score(y_test_clean, decisionTree.predict(X_test_enc
print(recall_dt)

f1_dt = f1_score(y_test_clean, decisionTree.predict(X_test_encoded) ,
print(f1_dt)

r2_dt = r2_score(y_test_clean, decisionTree.predict(X_test_encoded))
print(r2_dt)

accuracy_dt_notG1G2 = accuracy_score(y_test_clean, decisionTreeNotG1G2

print(accuracy_dt_notG1G2)
precision_dt_notG1G2 = precision_score(y_test_clean, decisionTreeNotG1

print(precision_dt_notG1G2)

recall_dt_notG1G2 = recall_score(y_test_clean, decisionTreeNotG1G2.pre
print(recall_dt_notG1G2)

f1_dt_notG1G2 = f1_score(y_test_clean, decisionTreeNotG1G2.predict(X_t
print(f1_dt_notG1G2)

r2_dt_notG1G2 = r2_score(y_test_clean, decisionTreeNotG1G2.predict(X_t
print(r2_dt_notG1G2)
```

```
0.7236842105263158
0.6552745098039214
0.715642857142857
0.6733444650587507
0.6062176165803109
0.3026315789473684
0.22465608465608464
0.21807142857142855
0.2167912857335213
-1.1747527084314648
```

In [52]:

```
XGBoost = GradientBoostingClassifier()

XGBoost.fit(X1_train_encoded, y_train_clean)

XGBoostNotG1G2 = GradientBoostingClassifier()
XGBoostNotG1G2.fit(X1_not_G1_G2, y_train_clean)

## testing

accuracy_xgb = accuracy_score(y_test_clean, XGBoost.predict(X_test_enc

print(accuracy_xgb)
precision_xgb = precision_score(y_test_clean, XGBoost.predict(X_test_e

print(precision_xgb)

recall_xgb = recall_score(y_test_clean, XGBoost.predict(X_test_encoded
print(recall_xgb)

f1_xgb = f1_score(y_test_clean, XGBoost.predict(X_test_encoded) , aver
print(f1_xgb)

r2_xgb = r2_score(y_test_clean, XGBoost.predict(X_test_encoded))
print(r2_xgb)

print("-------------------")
accuracy_xgb_notG1G2 = accuracy_score(y_test_clean, XGBoostNotG1G2.pre

print(accuracy_xgb_notG1G2)
precision_xgb_notG1G2 = precision_score(y_test_clean, XGBoostNotG1G2.p

print(precision_xgb_notG1G2)

recall_xgb_notG1G2 = recall_score(y_test_clean, XGBoostNotG1G2.predict
print(recall_xgb_notG1G2)

f1_xgb_notG1G2 = f1_score(y_test_clean, XGBoostNotG1G2.predict(X_test_
```

```
print(f1_xgb_notG1G2)

r2_xgb_notG1G2 = r2_score(y_test_clean, XGBoostNotG1G2.predict(X_test_
print(r2_xgb_notG1G2)
```

```
0.7763157894736842
0.6818165815877487
0.7315714285714285
0.6686601307189541
0.7404616109279322
-------------------
0.42105263157894735
0.33327812284334024
0.38233333333333333
0.33288283797832496
0.11398963730569955

/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [53]:

```
lor = LogisticRegression()

lor.fit(X1_train_encoded, y_train_clean)

lorNotG1G2 = LogisticRegression()
lorNotG1G2.fit(X1_not_G1_G2, y_train_clean)

## testing

accuracy_lor = accuracy_score(y_test_clean, lor.predict(X_test_encoded

print(accuracy_lor)
precision_lor = precision_score(y_test_clean, lor.predict(X_test_encod

print(precision_lor)

recall_lor = recall_score(y_test_clean, lor.predict(X_test_encoded), a
print(recall_lor)

f1_lor = f1_score(y_test_clean, lor.predict(X_test_encoded) , average=
print(f1_lor)

r2_lor = r2_score(y_test_clean, lor.predict(X_test_encoded))
print(r2_lor)

print("-------------------")
accuracy_lor_notG1G2 = accuracy_score(y_test_clean, lorNotG1G2.predict
```

```
print(accuracy_lor_notG1G2)
precision_lor_notG1G2 = precision_score(y_test_clean, lorNotG1G2.predi

print(precision_lor_notG1G2)

recall_lor_notG1G2 = recall_score(y_test_clean, lorNotG1G2.predict(X_t
print(recall_xgb_notG1G2)

f1_lor_notG1G2 = f1_score(y_test_clean, lorNotG1G2.predict(X_test_not_
print(f1_xgb_notG1G2)

r2_lor_notG1G2 = r2_score(y_test_clean, lorNotG1G2.predict(X_test_not_
print(r2_lor_notG1G2)
```

```
0.618421052631579
0.6053695324283559
0.5995714285714285
0.5614979505926664
0.4182760244936411
-------------------
0.3815789473684211
0.3811654135338346
0.3823333333333333
0.33288283797832496
0.04239284032030155

/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/line
ar_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to conver
ge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sho
wn in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver option
s:
    https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression (https://scikit-learn.org/stable/modules/linear_model.ht
ml#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/line
ar_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to conver
ge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sho
wn in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
```

```
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver option
s:
    https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression (https://scikit-learn.org/stable/modules/linear_model.ht
ml#logistic-regression)
  n_iter_i = _check_optimize_result(
```

2. Compare all three models both with and without the G1/G2 columns with crossvalidation. Output your results.

In [54]:

```python
## perform crossvalidation to check for overfitting and how the models
models = {
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "Logistic Regression": LogisticRegression()
}

scoring = {'accuracy': make_scorer(accuracy_score),
           'precision': make_scorer(precision_score, average='macro'),
           'recall': make_scorer(recall_score, average='macro'),
           'r2_score': make_scorer(r2_score),
           'f1_score': make_scorer(f1_score, average='macro')}

results = {}

for name, model in models.items():
    result = cross_validate(model, X1_train_encoded, y_train_clean, cv
    print(result)
    results[name] = result
    print(f"{name}: Mean Accuracy: {np.mean(result['test_accuracy']):.


# Set up the matplotlib figure and axes
fig, ax = plt.subplots(5, 1, figsize=(14, 20), sharey=True)
ax[0].set_title('Comparison of Accuracy')
ax[1].set_title('Comparison of Precision')
ax[2].set_title('Comparison of Recall')
ax[3].set_title('Comparison of R2 Score')
ax[4].set_title('Comparison of F1 Score')

index=0
# Plotting
for idx, metric in enumerate(['test_accuracy', 'test_precision', 'test
    for name in results:
        ax[idx].bar(name, np.mean(results[name][metric]), label=f"{nam
    ax[idx].set_xticklabels(labels=models.keys(), rotation=45)
    ax[idx].set_ylabel('Score')
```

```
    index+=1

plt.legend()
plt.tight_layout()
plt.show()
```

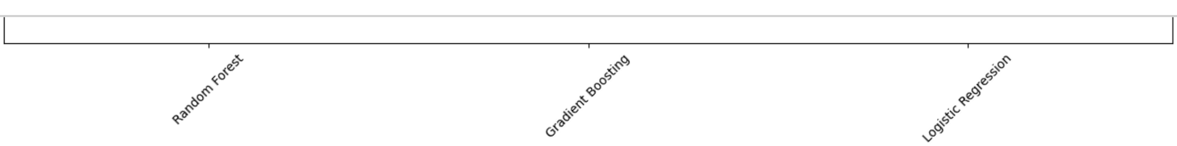
```
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

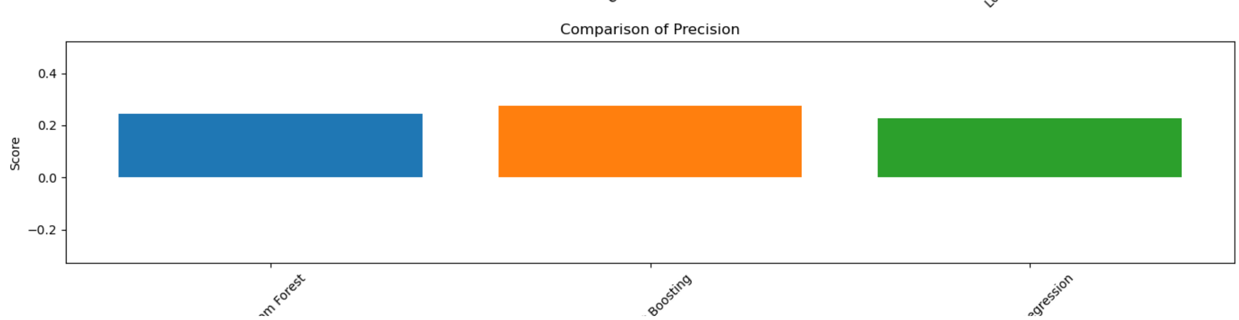
In [55]:
```
results
```

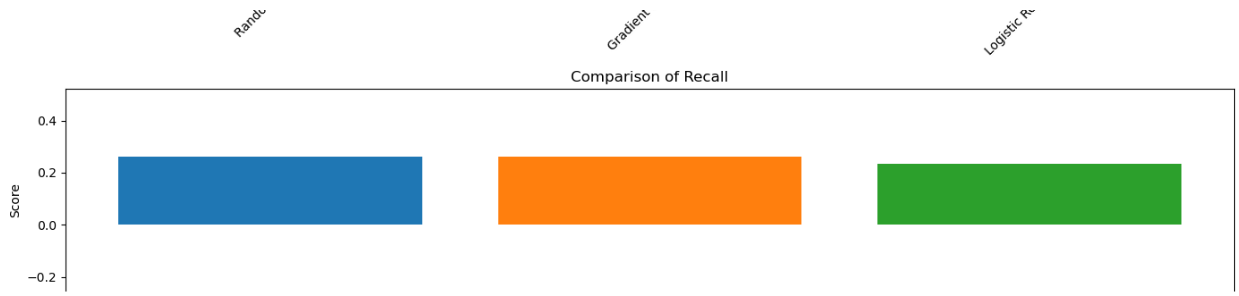
Out[55]:
```
{'Random Forest': {'fit_time': array([0.20082092, 0.16680312, 0.18070
722, 0.1631732 , 0.18131399]),
  'score_time': array([0.02196407, 0.02080798, 0.02348423, 0.0202250
5, 0.02189803]),
  'test_accuracy': array([0.75409836, 0.68852459, 0.78688525, 0.81967
213, 0.63934426]),
  'test_precision': array([0.6137276 , 0.41083333, 0.64659933, 0.6707
6401, 0.3751634 ]),
  'test_recall': array([0.5122807 , 0.45296296, 0.6262963 , 0.5723076
9, 0.38811966]),
  'test_r2_score': array([0.63958641, 0.34183423, 0.59612555, 0.66045
499, 0.45377541]),
  'test_f1_score': array([0.5287651 , 0.43032901, 0.62772487, 0.58636
977, 0.37659367])},
 'Gradient Boosting': {'fit_time': array([0.69831491, 0.67987108, 0.6
9769526, 0.67255878, 0.66838765]),
  'score_time': array([0.00989318, 0.01089621, 0.01215792, 0.0099222
7, 0.00988317]),
  'test_accuracy': array([0.78688525, 0.70491803, 0.7704918 , 0.77049
18 , 0.78688525]),
  'test_precision': array([0.71727163, 0.45415584, 0.59974747, 0.6362
8594, 0.81785488]),
  'test_recall': array([0.70175439, 0.49481481, 0.66055556, 0.6526495
7, 0.78358974]),
  'test_r2_score': array([0.57951748, 0.59612555, 0.61108386, 0.61616
651, 0.71950629]),
  'test_f1_score': array([0.70365915, 0.47080929, 0.62521008, 0.64407
814, 0.79608819])},
 'Logistic Regression': {'fit_time': array([0.03066802, 0.02944112,
0.02925277, 0.03077292, 0.02895594]),
  'score_time': array([0.00921202, 0.0087409 , 0.00872922, 0.0086250
3, 0.00859213]),
  'test_accuracy': array([0.60655738, 0.55737705, 0.63934426, 0.59016
393, 0.54098361]),
  'test_precision': array([0.33070175, 0.35467492, 0.34247649, 0.5241
285 , 0.3131063 ]),
  'test_recall': array([0.34736842, 0.33314815, 0.36814815, 0.3995726
5, 0.29606838]),
  'test_r2_score': array([0.30920729, 0.22216773, 0.50637567, 0.18804
453, 0.1732817 ]),
  'test_f1_score': array([0.33403509, 0.33430353, 0.3547619 , 0.42588
005, 0.28806306])}}
```

In [56]:
```
## perform crossvalidation to check for overfitting and how well the m
models = {
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
```

```
    "Logistic Regression": LogisticRegression()
}

scoring = {'accuracy': make_scorer(accuracy_score),
           'precision': make_scorer(precision_score, average='macro'),
           'recall': make_scorer(recall_score, average='macro'),
           'r2_score': make_scorer(r2_score),
           'f1_score': make_scorer(f1_score, average='macro')}

results = {}

for name, model in models.items():
    result = cross_validate(model, X1_not_G1_G2, y_train_clean, cv=5,
    results[name] = result
    print(f"{name}: Mean Accuracy: {np.mean(result['test_accuracy']):.


# Set up the matplotlib figure and axes
fig, ax = plt.subplots(5, 1, figsize=(14, 20), sharey=True)
ax[0].set_title('Comparison of Accuracy')
ax[1].set_title('Comparison of Precision')
ax[2].set_title('Comparison of Recall')
ax[3].set_title('Comparison of R2 Score')
ax[4].set_title('Comparison of F1 Score')

index=0
# Plotting
for idx, metric in enumerate(['test_accuracy', 'test_precision', 'test
    for name in results:
        ax[idx].bar(name, np.mean(results[name][metric]), label=f"{nam
    ax[idx].set_xticklabels(labels=models.keys(), rotation=45)
    ax[idx].set_ylabel('Score')
    index+=1

plt.legend()
plt.tight_layout()
plt.show()
```







Fine-Tune the System

1. Pick one model and use at least one grid search to fine-tune hyperparameters

In [57]:
```
## since the best performing model in both scenarios is the XGBoost, w
# Initialize the XGBoost classifier
xgb_clf = GradientBoostingClassifier()

# Define the parameter grid
param_grid = {
    # 'max_depth': [3, 5, 7],
    # 'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    # 'subsample': [0.8, 0.9, 1.0]
}

# Perform grid search
grid_search = GridSearchCV(xgb_clf, param_grid, cv=5, scoring='f1_macr
grid_search.fit(X1_train_encoded, y_train_clean)

# Print the best hyperparameters
print("Best hyperparameters:", grid_search.best_params_)

# Print the best score
print("Best score:", grid_search.best_score_)

best_model = grid_search.best_estimator_
```

```
Best hyperparameters: {'n_estimators': 200}
Best score: 0.6526617863841502
```

2. Correctly transform your testing data using your data preparation pipeline(s).

In [58]:

```
X_test_encoded = pipeline.transform(X_test)
y_test_clean = y_test.loc[X_test_encoded.index]
X_test_not_G1_G2 = pipe_no_g1_g2.transform(X_test)
```

```
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
Transforming OneHotEncodingTransformer
Transforming CategoricalToNumericalTransformer
Transforming FailuresCategorizer
Transforming AgeCategoryTransformer
Transforming TravelTimeCategoryTransformer
Transforming FamilyRelativeTransormer
Transforming FreeTimeCategoryTransformer
Transforming GooutTransformer
Transforming DailyAlcoholTransformer
Transforming BaseFeatureTransformer
Transforming EduCategoryTransformer
Transforming EduCategoryTransformer
Transforming StudyTimeCategoryTransformer
Transforming FamsizeCategoryTransformer
Transforming MJobCategoryTransformer
Transforming FJobCategoryTransformer
Transforming GuardianCategoryTransformer
Transforming DropNaTransformer
Transforming AbsencesFeatureGenerator
Transforming MinMaxScalerTransformer
Transforming OneHotEncodingTransformer
Transforming CategoricalToNumericalTransformer
```

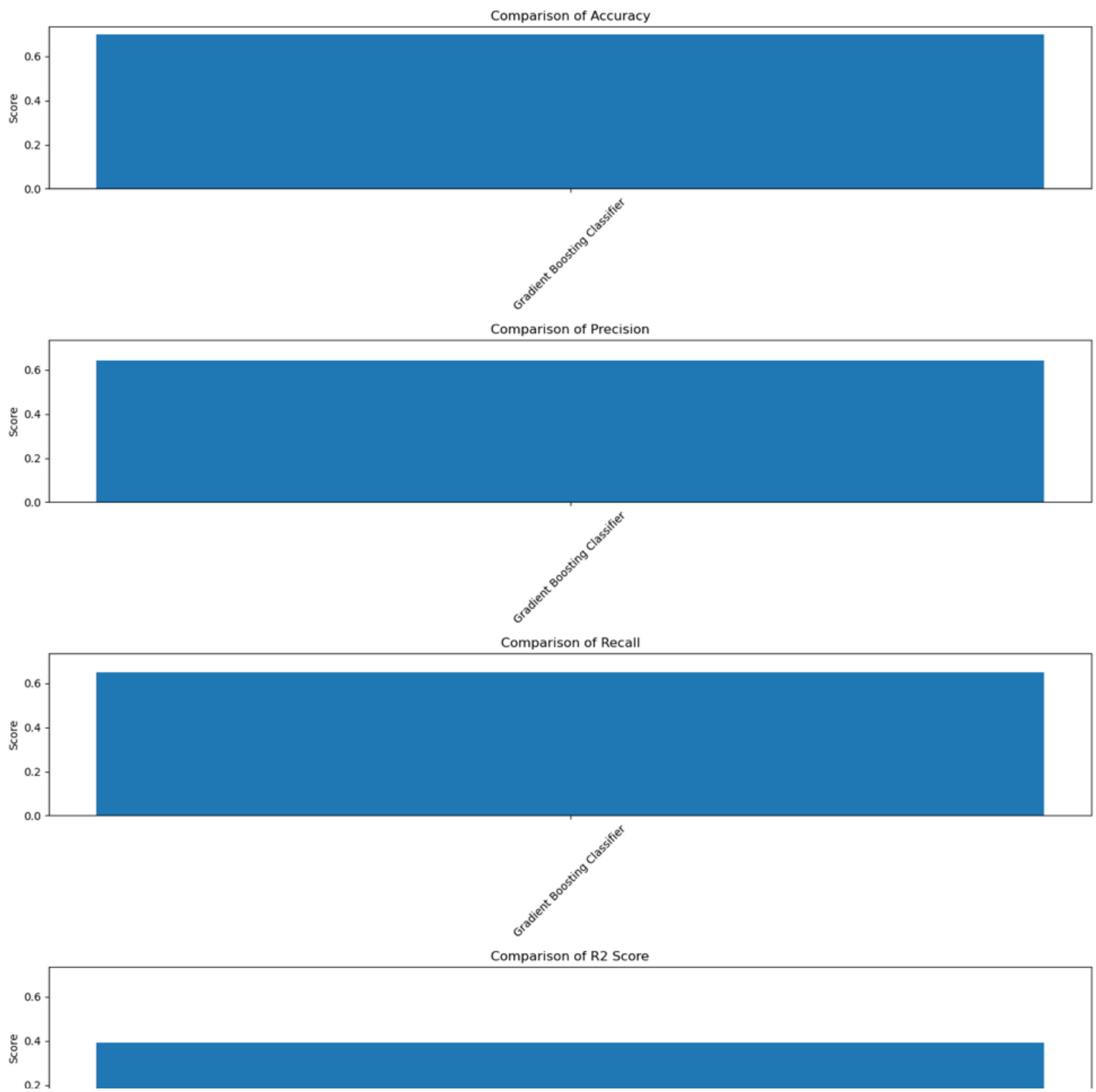
3. Select your final model and measure its performance on the test set

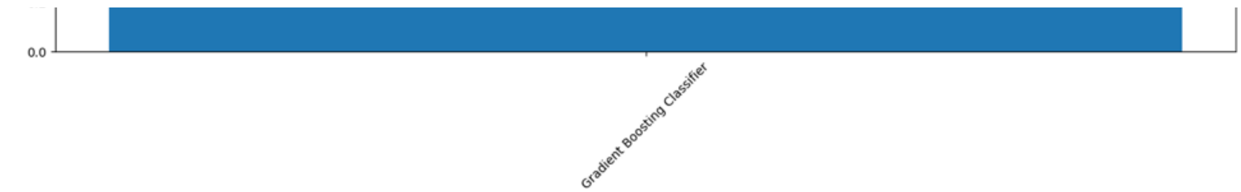
In [59]:

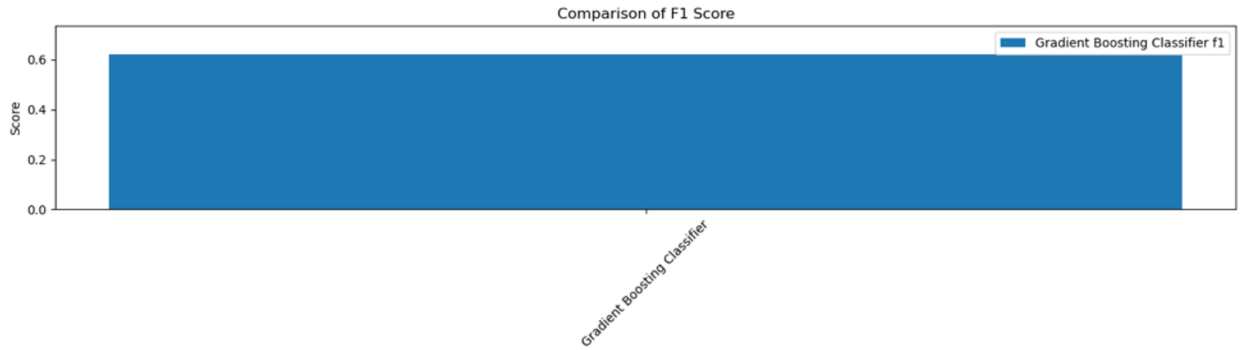
```
## perform crossvalidation to check for overfitting
models = {
    "Gradient Boosting Classifier": best_model
}

scoring = {'accuracy': make_scorer(accuracy_score),
           'precision': make_scorer(precision_score, average='macro'),
           'recall': make_scorer(recall_score, average='macro'),
           'r2_score': make_scorer(r2_score),
           'f1_score': make_scorer(f1_score, average='macro')}

results = {}

for name, model in models.items():
    result = cross_validate(model, X_test_encoded, y_test_clean, cv=5,
    results[name] = result
    print(f"{name}: Mean Accuracy: {np.mean(result['test_accuracy']):.


# Set up the matplotlib figure and axes
fig, ax = plt.subplots(5, 1, figsize=(14, 20), sharey=True)
ax[0].set_title('Comparison of Accuracy')
ax[1].set_title('Comparison of Precision')
ax[2].set_title('Comparison of Recall')
ax[3].set_title('Comparison of R2 Score')
ax[4].set_title('Comparison of F1 Score')

index=0
# Plotting
for idx, metric in enumerate(['test_accuracy', 'test_precision', 'test
    for name in results:
        ax[idx].bar(name, np.mean(results[name][metric]), label=f"{nam
    ax[idx].set_xticklabels(labels=models.keys(), rotation=45)
    ax[idx].set_ylabel('Score')
    index+=1

plt.legend()
plt.tight_layout()
plt.show()
```

```
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/mode
l_selection/_split.py:725: UserWarning: The least populated class in
y has only 3 members, which is less than n_splits=5.
  warnings.warn(
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
```

```
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/amymoschos/anaconda3/lib/python3.11/site-packages/sklearn/metr
ics/_classification.py:1469: UndefinedMetricWarning: Precision is ill
-defined and being set to 0.0 in labels with no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/var/folders/7v/8544rkhj2vn_rd_7t5dmcbk80000gn/T/ipykernel_90759/1915
936359.py:33: UserWarning: FixedFormatter should only be used togethe
r with FixedLocator
  ax[idx].set_xticklabels(labels=models.keys(), rotation=45)

Gradient Boosting Classifier: Mean Accuracy: 0.699, Mean F1 Score: 0.
622
```



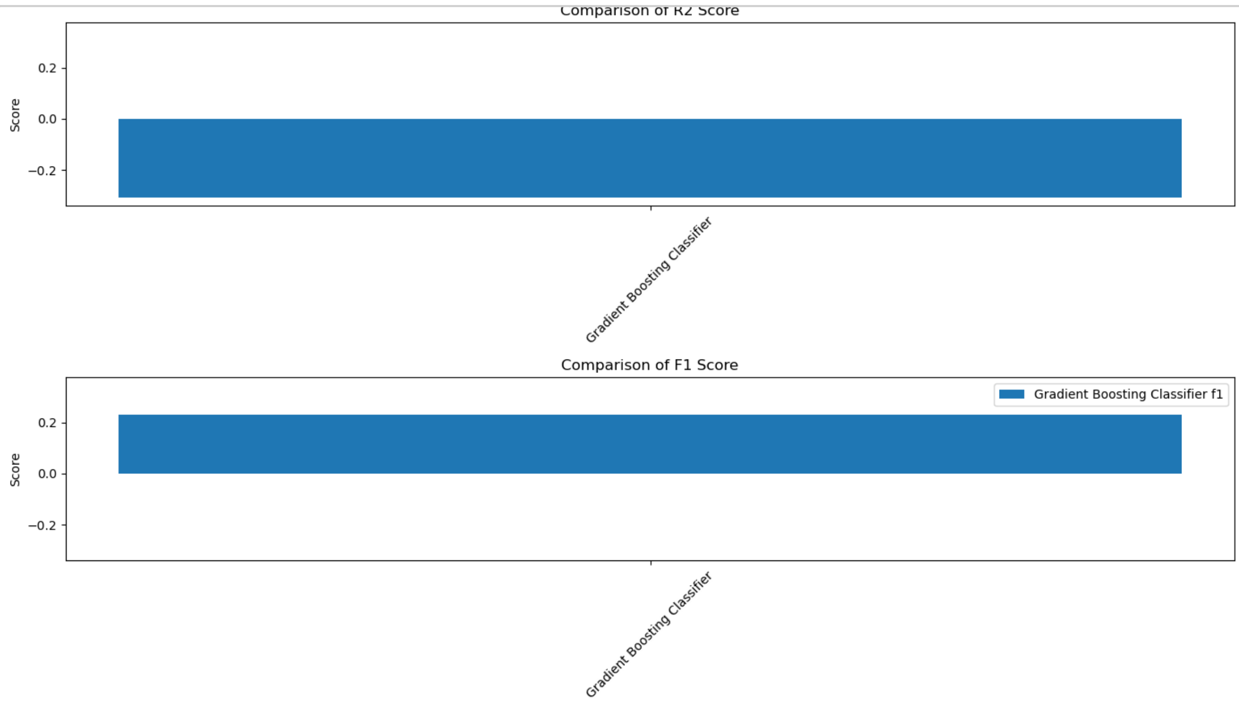




In [60]:

```
## perform crossvalidation to check for overfitting and make sure the
models = {
    "Gradient Boosting Classifier": best_model
}

scoring = {'accuracy': make_scorer(accuracy_score),
           'precision': make_scorer(precision_score, average='macro'),
           'recall': make_scorer(recall_score, average='macro'),
           'r2_score': make_scorer(r2_score),
           'f1_score': make_scorer(f1_score, average='macro')}

results = {}

for name, model in models.items():
    result = cross_validate(model, X_test_not_G1_G2, y_test_clean, cv=
    results[name] = result
    print(f"{name}: Mean Accuracy: {np.mean(result['test_accuracy']):.


# Set up the matplotlib figure and axes
fig, ax = plt.subplots(5, 1, figsize=(14, 20), sharey=True)
ax[0].set_title('Comparison of Accuracy')
ax[1].set_title('Comparison of Precision')
ax[2].set_title('Comparison of Recall')
ax[3].set_title('Comparison of R2 Score')
ax[4].set_title('Comparison of F1 Score')

index=0
# Plotting
for idx, metric in enumerate(['test_accuracy', 'test_precision', 'test
    for name in results:
        ax[idx].bar(name, np.mean(results[name][metric]), label=f"{nam
    ax[idx].set_xticklabels(labels=models.keys(), rotation=45)
```

```
    ax[idx].set_ylabel('Score')
    index+=1

plt.legend()
plt.tight_layout()
plt.show()
```



Present Your Solution See below

See the below additional rubric categories for the items related to presenting your solution to your executive team (in other words, presenting your work in this Jupyter notebook). Your project should include an overview and concluding section

```
####
```