Predictive Analytics for Student Performance

Objective: Use historical student data to predict academic performance and identify factors affecting grades.

Outcome: Develop a model to predict student performance and provide insights for educational improvement.

In this notebook, we will :

- **Predict whether or not a student will pass the final exam based on certain information given**
- **Compare the 2 learning algorithms**
- **Find out what most affects student achievement**
- **Find the best algorithm with high accuracy**

We will be using three learning algorithms:

- **Logistic regression**
- **Gradient Boosting Trees (XGBoost, CatBoost)**

Reading data

In [1]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)
import seaborn as sns
sns.set_theme(style='white', palette='muted', font_scale=0.9)
import matplotlib.pyplot as plt
from time import time
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (
confusion_matrix, roc_curve, accuracy_score, f1_score, roc_auc_score, classificatio
)
from astropy.table import Table
```

In [2]:
```
df = pd.read_csv('student-data.csv')
df_copy = pd.read_csv('student-data.csv')
```

Dataset

Displaying the dataset

In [3]:
```
# first 10 rows
df.head(10)
```

Out[3]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course
1	GP	F	17	U	GT3	T	1	1	at_home	other	course
2	GP	F	15	U	LE3	T	1	1	at_home	other	other
3	GP	F	15	U	GT3	T	4	2	health	services	home
4	GP	F	16	U	GT3	T	3	3	other	other	home
5	GP	M	16	U	LE3	T	4	3	services	other	reputation
6	GP	M	16	U	LE3	T	2	2	other	other	home
7	GP	F	17	U	GT3	A	4	4	other	teacher	home
8	GP	M	15	U	LE3	A	3	2	services	other	home
9	GP	M	15	U	GT3	T	3	4	other	other	home

In [4]:
```
# last 10 rows
df.tail(10)
```

Out[4]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reas
385	MS	F	18	R	GT3	T	2	2	at_home	other	otl
386	MS	F	18	R	GT3	T	4	4	teacher	at_home	reputati
387	MS	F	19	R	GT3	T	2	3	services	other	cou
388	MS	F	18	U	LE3	T	3	1	teacher	services	cou
389	MS	F	18	U	GT3	T	1	1	other	other	cou
390	MS	M	20	U	LE3	A	2	2	services	services	cou
391	MS	M	17	U	LE3	T	3	1	services	services	cou
392	MS	M	21	R	GT3	T	1	1	other	other	cou
393	MS	M	18	R	LE3	T	3	2	services	other	cou
394	MS	M	19	U	LE3	T	1	1	other	at_home	cou

In [5]:
```
print(f'The dataset has {df.shape[0]} columns and {df.shape[1]} rows.')
```

```
The dataset has 395 columns and 31 rows.
```

In [6]:
```python
print(f'The total number of missing values in the dataset is {df.isnull().sum().sum
```

```
The total number of missing values in the dataset is 0.
```

Now let's explain every column in the dataframe

- `school` : student's school (binary: "GP" or "MS")
- `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 - 5th to 9th grade, 3 - secondary education or 4 - higher education)
- `Fedu` : father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, 3 - secondary education or 4 - 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)

The last column:

- `passed` : did the student pass the final exam or not (binary: yes or no)

Data processing

In [7]:
```
# Get columns of type 'object' or 'category'
categorical_columns = df.select_dtypes(include=['object', 'category']).columns.toli
print(f"The number of categorical/non-numerical columns in the dataset is {len(cate
```

```
The number of categorical/non-numerical columns in the dataset is 18 and they includ
e the following ::: ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjo
b', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'h
igher', 'internet', 'romantic', 'passed'].
```

In [8]:
```
# Show value counts for each of these columns
for col in categorical_columns:
    print(f"Value counts for column '{col}':")
    print(df[col].value_counts())
    print("\n" + "-"*40 + "\n")
```

```
Value counts for column 'school':
school
GP    349
MS     46
Name: count, dtype: int64

----------------------------------------

Value counts for column 'sex':
sex
F    208
M    187
Name: count, dtype: int64

----------------------------------------

Value counts for column 'address':
address
U    307
R     88
Name: count, dtype: int64

----------------------------------------

Value counts for column 'famsize':
famsize
GT3    281
LE3    114
Name: count, dtype: int64

----------------------------------------

Value counts for column 'Pstatus':
Pstatus
T    354
A     41
Name: count, dtype: int64

----------------------------------------

Value counts for column 'Mjob':
Mjob
other       141
services    103
at_home      59
teacher      58
health       34
Name: count, dtype: int64

----------------------------------------

Value counts for column 'Fjob':
Fjob
other       217
services    111
teacher      29
```

```
at_home      20
health       18
Name: count, dtype: int64

----------------------------------------

Value counts for column 'reason':
reason
course        145
home          109
reputation    105
other          36
Name: count, dtype: int64

----------------------------------------

Value counts for column 'guardian':
guardian
mother    273
father     90
other      32
Name: count, dtype: int64

----------------------------------------

Value counts for column 'schoolsup':
schoolsup
no     344
yes     51
Name: count, dtype: int64

----------------------------------------

Value counts for column 'famsup':
famsup
yes    242
no     153
Name: count, dtype: int64

----------------------------------------

Value counts for column 'paid':
paid
no     214
yes    181
Name: count, dtype: int64

----------------------------------------

Value counts for column 'activities':
activities
yes    201
no     194
Name: count, dtype: int64

----------------------------------------
```

```
Value counts for column 'nursery':
nursery
yes    314
no      81
Name: count, dtype: int64

----------------------------------------

Value counts for column 'higher':
higher
yes    375
no      20
Name: count, dtype: int64

----------------------------------------

Value counts for column 'internet':
internet
yes    329
no      66
Name: count, dtype: int64

----------------------------------------

Value counts for column 'romantic':
romantic
no     263
yes    132
Name: count, dtype: int64

----------------------------------------

Value counts for column 'passed':
passed
yes    265
no     130
Name: count, dtype: int64

----------------------------------------
```

Before proceeding further, we need to process the data to ensure it is properly prepared for training machine learning models. Data preprocessing is a crucial step that enhances model performance by handling missing values, encoding categorical features, and scaling features. Below are the functions that will be applied in this process.

1) Encoding Categorical Variables:

Many datasets contain categorical variables, which are non-numeric by nature, and machine learning models often cannot process such values directly. To make these variables usable, we convert categorical values into numerical ones using Scikit-learn's `LabelEncoder`. The function responsible for this task is:

```
def encode_categorical_columns(df)
```

- This function applies `LabelEncoder` to all categorical columns in the dataset, converting them to numeric values.
- **Why this is important:** Most machine learning algorithms work better or only accept numeric input. Encoding allows the model to interpret categorical variables effectively.

---

2) Feature Scaling:

Feature scaling ensures that all the input features have the same scale, which can significantly impact the performance of models, especially those relying on gradient descent optimization (e.g., logistic regression, neural networks). Without scaling, features with larger ranges might dominate the learning process, leading to suboptimal model performance. Here, we apply two types of scaling methods:

(a) Min-Max Scaling:

This method scales each feature to a range between 0 and 1 or another defined range. The formula is as follows:

$$\frac{col - \min(col)}{\max(col) - \min(col)}$$

This normalization method is useful when the distribution of data is unknown or not Gaussian.

- **Function:**

```
def min_max_scaling(df)
```

This function replaces each column with its normalized value based on the min-max scaling method, which brings all values between 0 and 1.

(b) Standardization (Z-score scaling):

Standardization rescales features so that they have the properties of a standard normal distribution (mean of 0 and standard deviation of 1). The formula is:

$$\frac{col - mean(col)}{std(col)}$$

Where:

- $mean(col)$: Mean of the column
- $std(col)$: Standard deviation of the column

This method is especially useful when the data follows a Gaussian (normal) distribution.

- **Function:**

`def standard_scaling(df)`
This function applies standardization to the dataset, ensuring that each feature is rescaled using the Z-score.

Why use both methods?

- **Min-Max Scaling** is useful for algorithms that rely on distances, such as K-Nearest Neighbors or neural networks, where bounded ranges help convergence.
- **Standardization** is ideal for algorithms like Support Vector Machines and Logistic Regression, which assume normally distributed data.

Function Invocation:

- To apply these scaling techniques, simply call the relevant function:

`scaled_df = feature_scaling(df)`
By scaling the data, we ensure that each feature contributes proportionally to the learning algorithm, thus speeding up convergence and improving model performance.

In [9]:
```
from sklearn.preprocessing import LabelEncoder

# Function to encode columns and display unique original and mapped values
def encode_categorical_columns(df, categorical_columns):
    # Loop over each categorical column
    for col in df[categorical_columns].columns:
        le = LabelEncoder()
        original_values = df[col].copy()  # Keep original values
        df[col] = le.fit_transform(df[col])  # Encode the column

        # Get unique pairs of original and encoded values
        unique_mappings = set(zip(original_values, df[col]))

        # Display original and encoded values
        print(f"Column: '{col}'")
        print("Original -> Encoded (Unique Values)")
        for orig, encoded in unique_mappings:
            print(f"{orig} -> {encoded}")
        print("\n")

# Call the function to encode and display unique mappings
encode_categorical_columns(df, categorical_columns=categorical_columns)
```

```
Column: 'school'
Original -> Encoded (Unique Values)
GP -> 0
MS -> 1


Column: 'sex'
Original -> Encoded (Unique Values)
M -> 1
F -> 0


Column: 'address'
Original -> Encoded (Unique Values)
U -> 1
R -> 0


Column: 'famsize'
Original -> Encoded (Unique Values)
LE3 -> 1
GT3 -> 0


Column: 'Pstatus'
Original -> Encoded (Unique Values)
A -> 0
T -> 1


Column: 'Mjob'
Original -> Encoded (Unique Values)
health -> 1
services -> 3
teacher -> 4
at_home -> 0
other -> 2


Column: 'Fjob'
Original -> Encoded (Unique Values)
services -> 3
health -> 1
teacher -> 4
at_home -> 0
other -> 2


Column: 'reason'
Original -> Encoded (Unique Values)
other -> 2
reputation -> 3
home -> 1
course -> 0
```

```
Column: 'guardian'
Original -> Encoded (Unique Values)
mother -> 1
father -> 0
other -> 2


Column: 'schoolsup'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'famsup'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'paid'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'activities'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'nursery'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'higher'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'internet'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'romantic'
Original -> Encoded (Unique Values)
no -> 0
yes -> 1


Column: 'passed'
```

```
Original -> Encoded (Unique Values)
no -> 0
yes -> 1
```

In [10]:
```
# let's check the data again
df
```

Out[10]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guar
0	0	0	18	1	0	0	4	4	0	4	0	
1	0	0	17	1	0	1	1	1	0	2	0	
2	0	0	15	1	1	1	1	1	0	2	2	
3	0	0	15	1	0	1	4	2	1	3	1	
4	0	0	16	1	0	1	3	3	2	2	1	
...	...	...	...	...	...	...	...	...	...	...	...	
390	1	1	20	1	1	0	2	2	3	3	0	
391	1	1	17	1	1	1	3	1	3	3	0	
392	1	1	21	0	0	1	1	1	2	2	0	
393	1	1	18	0	1	1	3	2	3	2	0	
394	1	1	19	1	1	1	1	1	2	0	0	

395 rows × 31 columns

Features scalling

In [11]:
```
from sklearn.preprocessing import StandardScaler

# Function to apply StandardScaler to non-categorical columns
def standard_scaling(df, categorical_columns):
    # Initialize StandardScaler
    scaler = StandardScaler()

    # Identify columns that are not categorical (numerical columns)
    non_categorical_columns = [col for col in df.columns if col not in categorical_

    # Apply scaling only to non-categorical (numerical) columns
    df[non_categorical_columns] = scaler.fit_transform(df[non_categorical_columns])

    # Return the scaled dataframe
    return df

df_scaled = standard_scaling(df, categorical_columns=categorical_columns)
df_scaled
```

Out[11]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob
0	0	0	1.023046	1	0	0	1.143856	1.360371	0	4
1	0	0	0.238380	1	0	1	-1.600009	-1.399970	0	2
2	0	0	-1.330954	1	1	1	-1.600009	-1.399970	0	2
3	0	0	-1.330954	1	0	1	1.143856	-0.479857	1	3
4	0	0	-0.546287	1	0	1	0.229234	0.440257	2	2
...	...	...	...	...	...	...	...	...	...	...
390	1	1	2.592380	1	1	0	-0.685387	-0.479857	3	3
391	1	1	0.238380	1	1	1	0.229234	-1.399970	3	3
392	1	1	3.377047	0	0	1	-1.600009	-1.399970	2	2
393	1	1	1.023046	0	1	1	0.229234	-0.479857	3	2
394	1	1	1.807713	1	1	1	-1.600009	-1.399970	2	0

395 rows × 31 columns

Exploratory Data Analysis

Firstly we are going to look deeper into each features by using multiple methods of visualisation such as distribution plot ,Density... After the visualisation we are going to understand wish features are most impactfull for student's performances.

If you are students,parents or teachers and you care about your kids or students academic performances you might want to have attention for next lectures, wi will provides you with summary of how you can achieve best social,demographic and school conditions to boost their academics potentials.

In [12]:
```
# we will use the copy of our original data in this case to perform Exploratory Dat
df_copy
```

Out[12]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course
1	GP	F	17	U	GT3	T	1	1	at_home	other	course
2	GP	F	15	U	LE3	T	1	1	at_home	other	other
3	GP	F	15	U	GT3	T	4	2	health	services	home
4	GP	F	16	U	GT3	T	3	3	other	other	home
...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	services	services	course
391	MS	M	17	U	LE3	T	3	1	services	services	course
392	MS	M	21	R	GT3	T	1	1	other	other	course
393	MS	M	18	R	LE3	T	3	2	services	other	course
394	MS	M	19	U	LE3	T	1	1	other	at_home	course

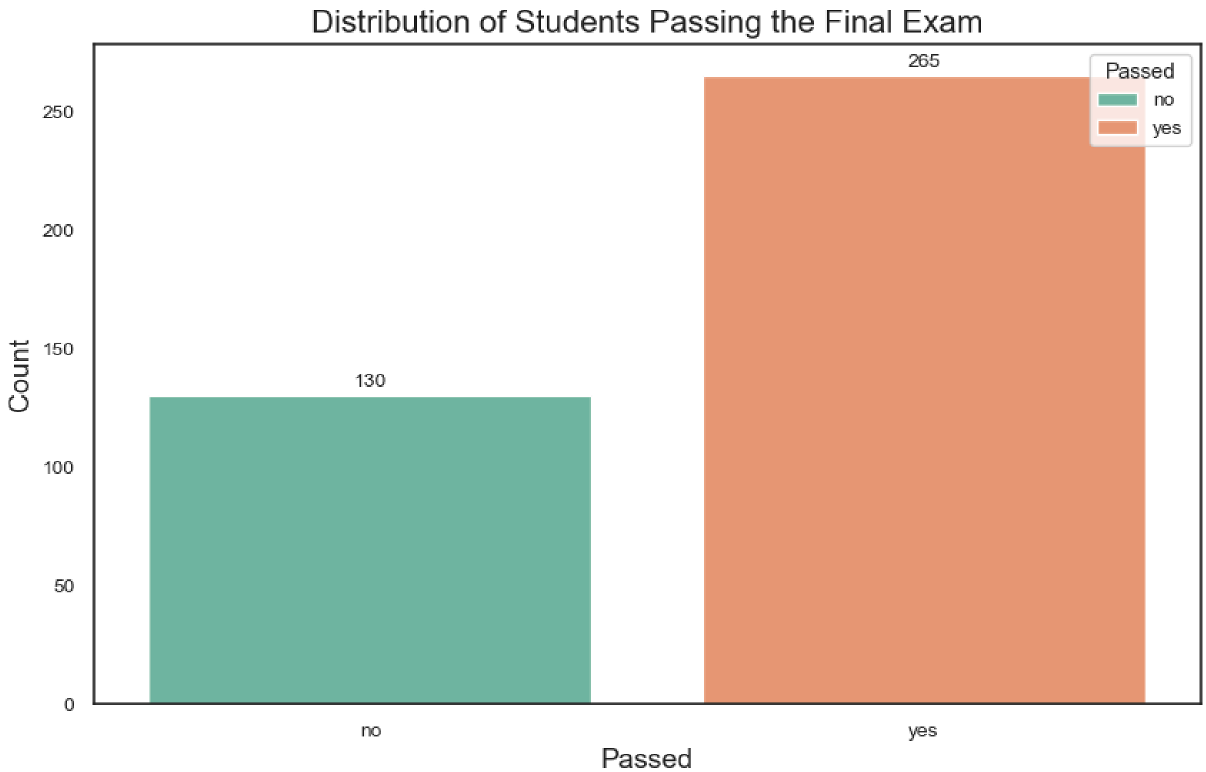
395 rows × 31 columns

In [13]:
```
print(df_copy.columns.tolist())
```

```
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'F
job', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'fam
sup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel',
'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'passed']
```

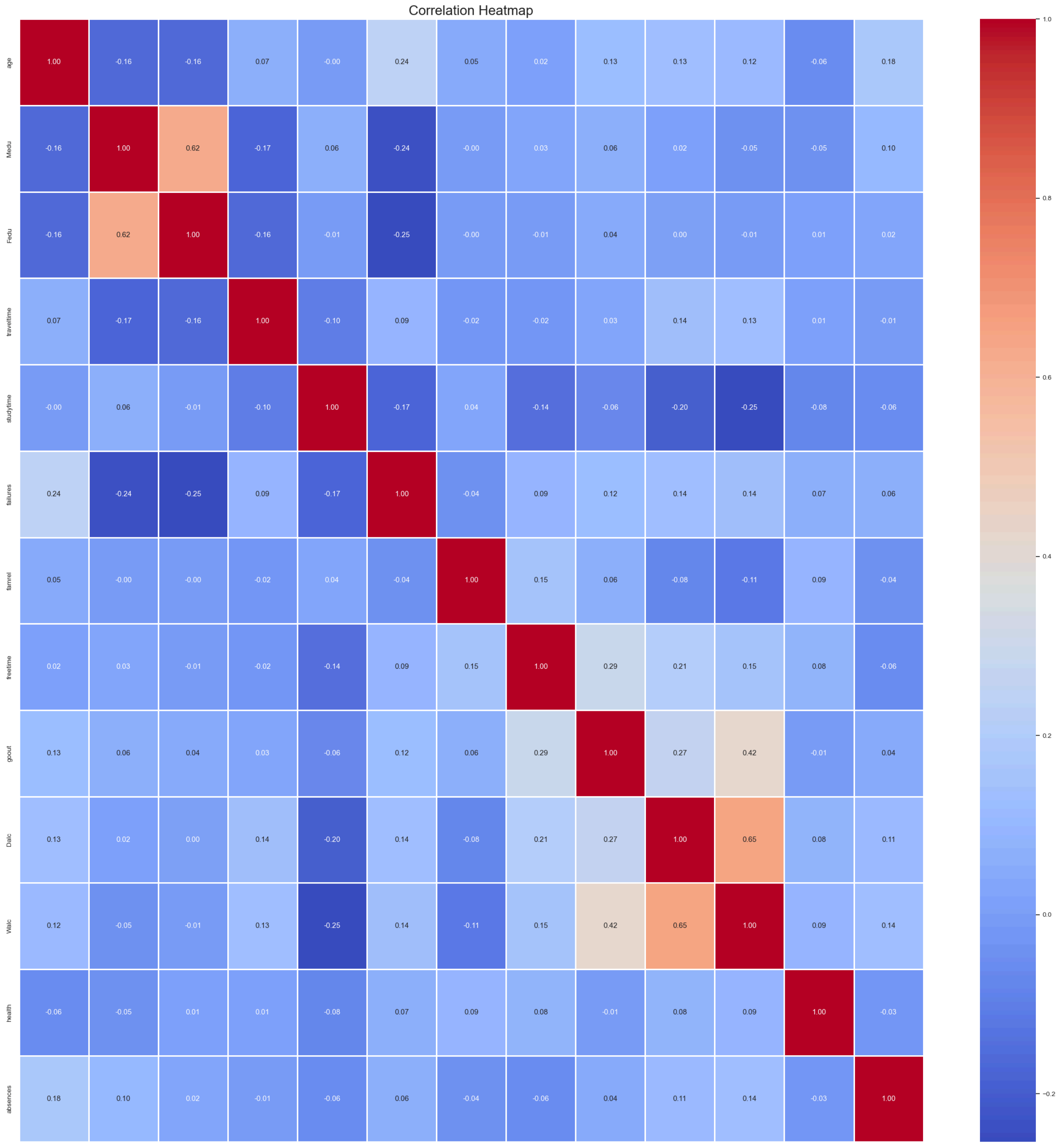
In [14]:
```
import warnings;warnings.filterwarnings('ignore')
```

In [15]:
```
# Plot the counts of each unique value in 'passed'
plt.figure(figsize=(10, 6))
sns.countplot(data=df_copy, x='passed', palette='Set2')

# Add counts on top of the bars
for p in plt.gca().patches:
    plt.text(p.get_x() + p.get_width() / 2, p.get_height() + 2,
             f'{int(p.get_height())}',
             ha='center', va='bottom', fontsize=10)

# Add labels and title
plt.xlabel('Passed', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Students Passing the Final Exam', fontsize=16)

# Add legend
plt.legend(title='Passed', labels=df_copy['passed'].unique(), loc='upper right')

plt.show();
```

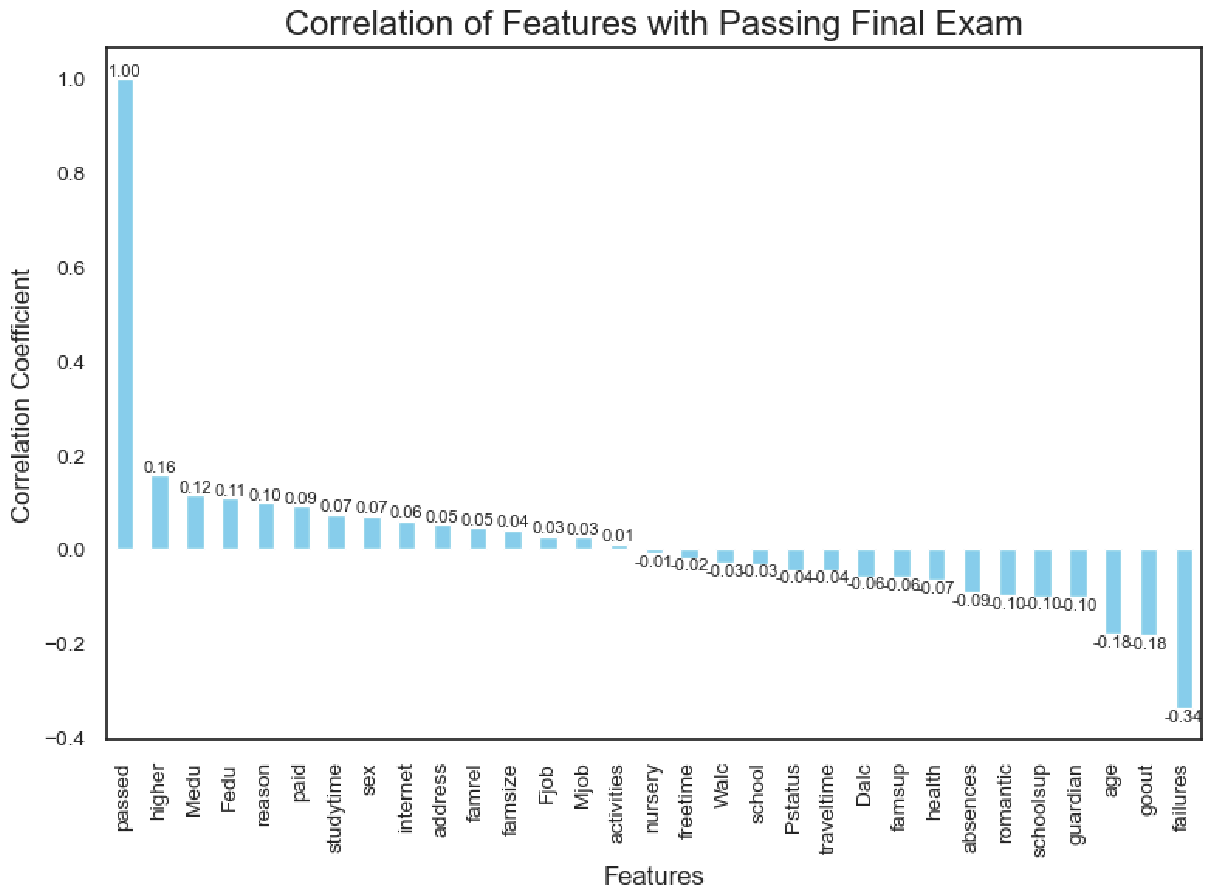


In [16]:
```
# see correlation between variables through a correlation heatmap
plt.figure(figsize=(30,30))
sns.heatmap(df_copy.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt='.2f'
plt.title('Correlation Heatmap', fontsize=20)
```

Out[16]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



In [17]:
```python
plt.figure(figsize=(8, 6))
df.corr()['passed'].sort_values(ascending=False).plot(kind='bar', color='skyblue')

plt.title('Correlation of Features with Passing Final Exam', fontsize=16)
plt.xlabel('Features', fontsize=12)
plt.ylabel('Correlation Coefficient', fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()

for index, value in enumerate(df.corr()['passed'].sort_values(ascending=False)):
    plt.text(index, value, f'{value:.2f}', ha='center', va='bottom' if value >= 0 e

plt.show();
```



Based on this heatmap we can do a quick conclusion about most impactful features on the status of students passign the final Exam:

Most Imapctful Features

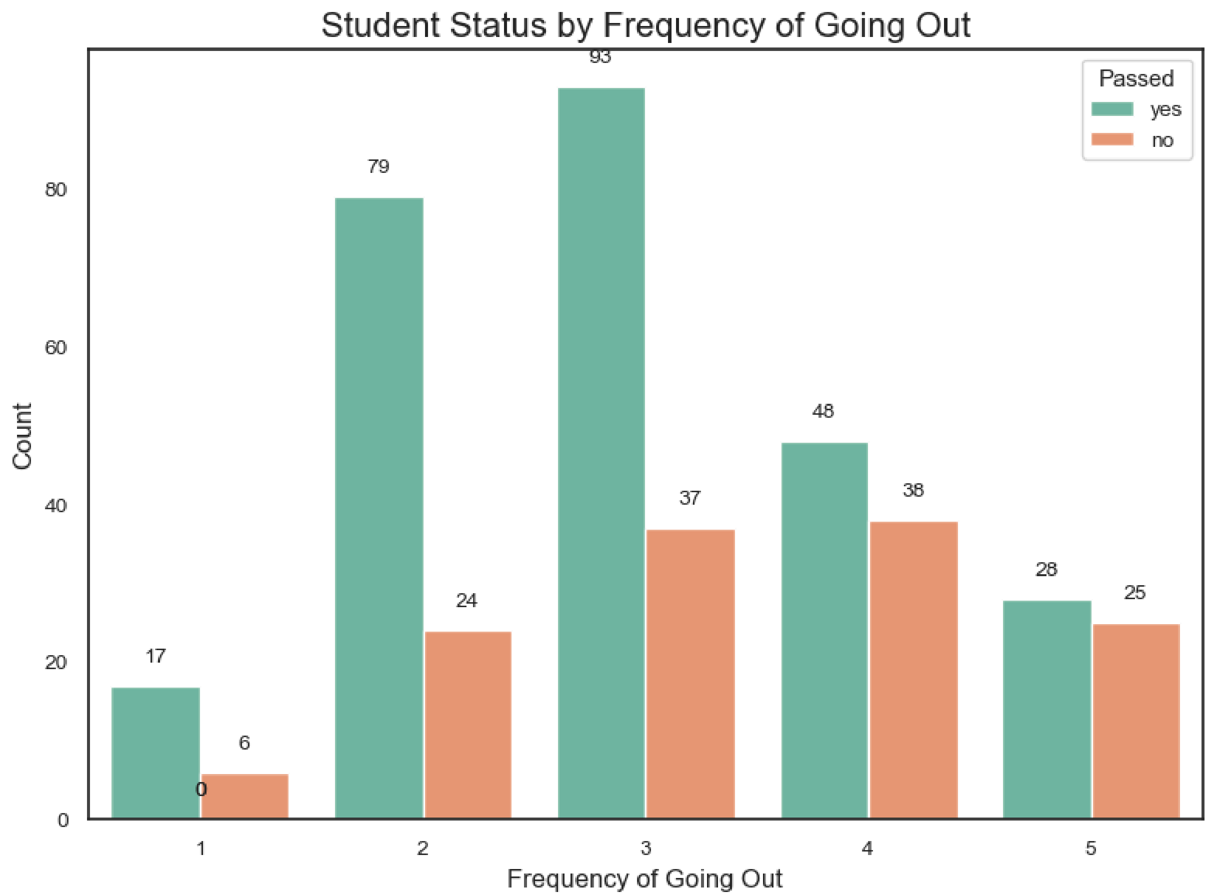
- Students hoping to take higher education
- Mother and Father's education status

Least Imapctful Features

- Number of Past Class Failures
- Going out with friends for too much hours can also impact badly
- Age of Student

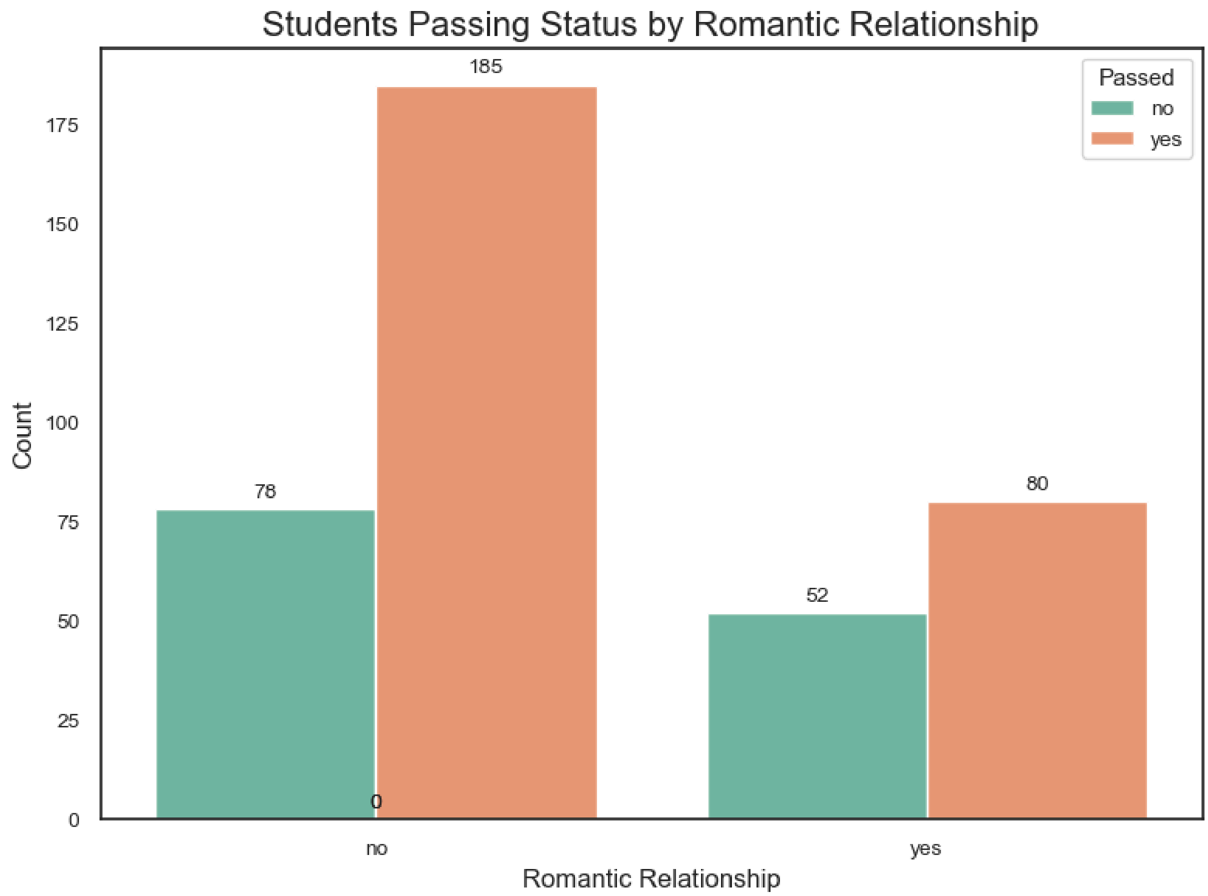
In [18]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='goout', hue='passed', data=df_copy, palette='Set2')

plt.title('Student Status by Frequency of Going Out', fontsize=16)
plt.xlabel('Frequency of Going Out', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='Passed', loc='upper right')
plt.xticks(rotation=0)
```

```
for p in plt.gca().patches:
    height = p.get_height()
    plt.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', ha='cent

plt.tight_layout()
plt.show();
```



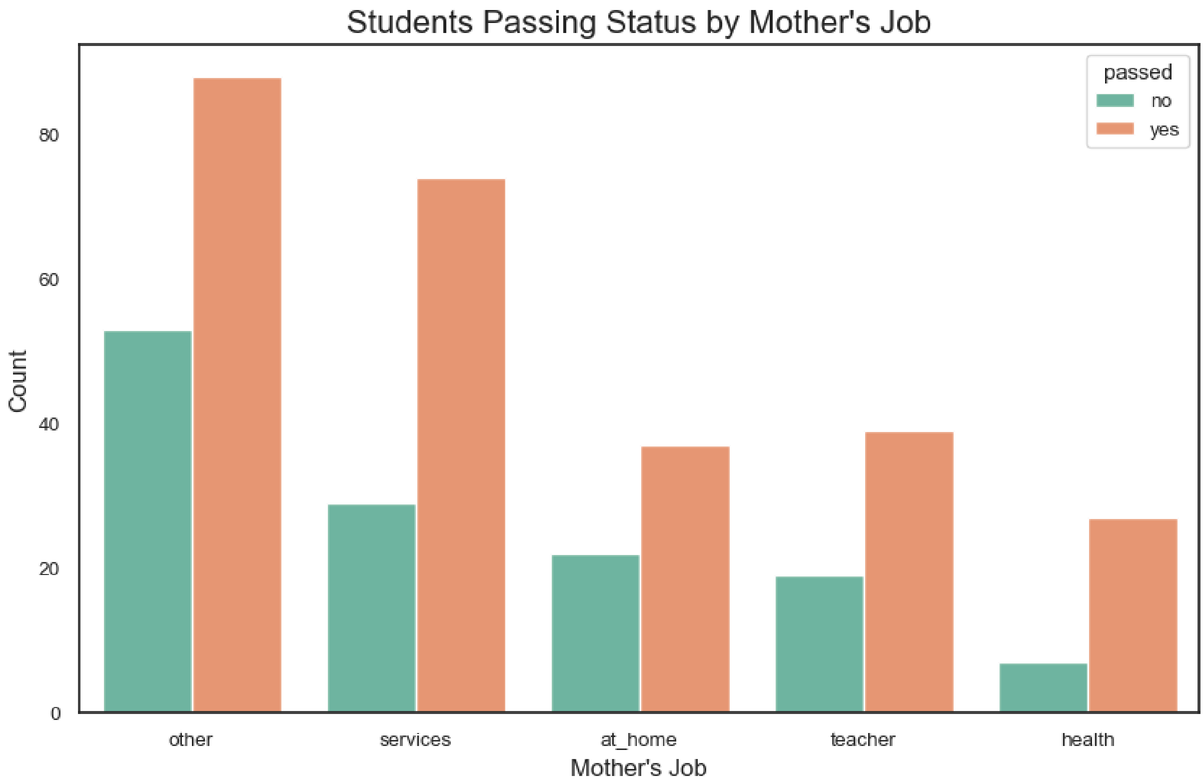
It seems that most of people who passed the exam had less hour of going out, as a conclusion we should limit the hours of going out with friends unnecessarily.

In [19]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='romantic', hue='passed', data=df_copy, palette='Set2')

plt.title('Students Passing Status by Romantic Relationship', fontsize=16)
plt.xlabel('Romantic Relationship', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='Passed', loc='upper right')
plt.xticks(rotation=0)

for p in plt.gca().patches:
    height = p.get_height()
    plt.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', ha='cent

plt.tight_layout()
plt.show();
```



Most of people who passed the exam had no romantic relation.
Romantic Relationship could be a good choice for better performance.

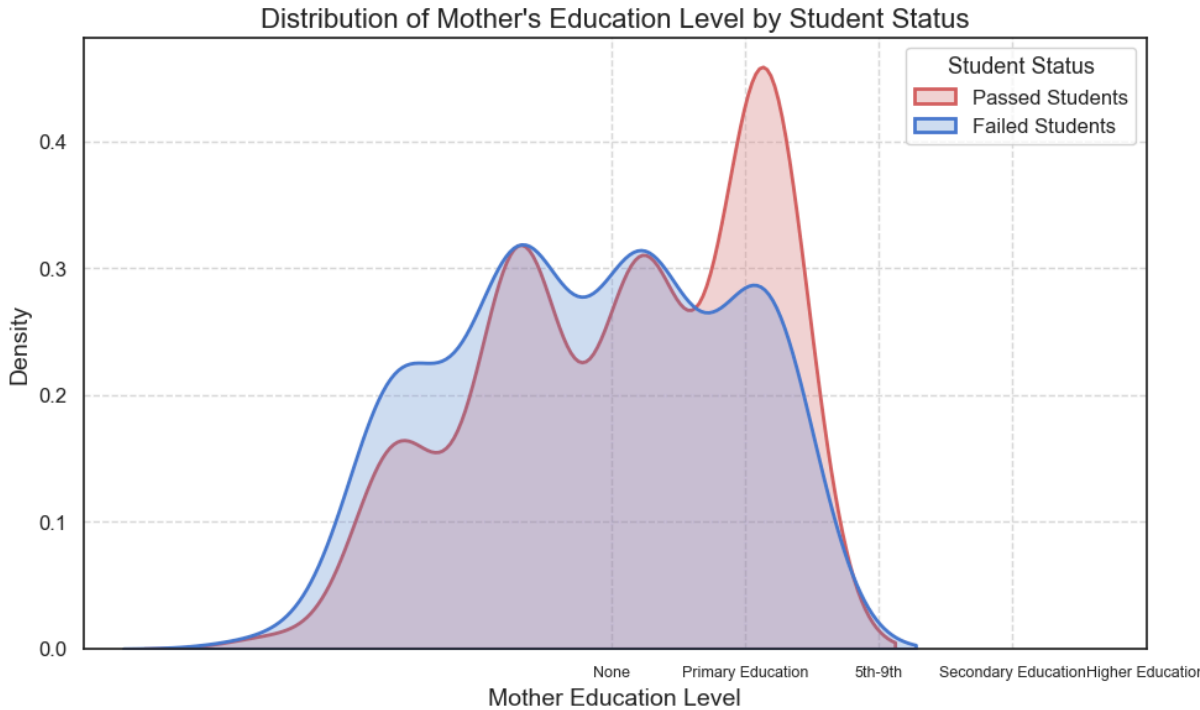
In [20]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Mjob', hue='passed', data=df_copy, palette='Set2', order=df_copy['
plt.title('Students Passing Status by Mother\'s Job', fontsize=16)
plt.xlabel('Mother\'s Job', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show();
```



Majority of the students that passed have their Mothers working other jobs.

In [21]:
```
# Split the data into groups based on 'passed'
passed_students = df.loc[df['passed'] == 1]
failed_students = df.loc[df['passed'] == 0]

# Create new columns for mother education levels
passed_students['Mother Education (Passed)'] = passed_students['Medu']
failed_students['Mother Education (Failed)'] = failed_students['Medu']

plt.figure(figsize=(10, 6))

# Plotting the KDE for both passed and failed students' mother education levels
sns.kdeplot(passed_students['Mother Education (Passed)'],
            shade=True, color="r", label='Passed Students',
            linewidth=2)

sns.kdeplot(failed_students['Mother Education (Failed)'],
            shade=True, color="b", label='Failed Students',
            linewidth=2)

# Adding labels, titles, and legends
plt.title("Distribution of Mother's Education Level by Student Status", fontsize=16
plt.xlabel('Mother Education Level', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.legend(title="Student Status", fontsize=12, title_fontsize=13)

# Customizing the ticks and layout
plt.xticks(ticks=[0, 1, 2, 3, 4], labels=['None', 'Primary Education', '5th-9th', '
plt.yticks(fontsize=12)
```

```
# Adding a grid for clarity
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show();
```

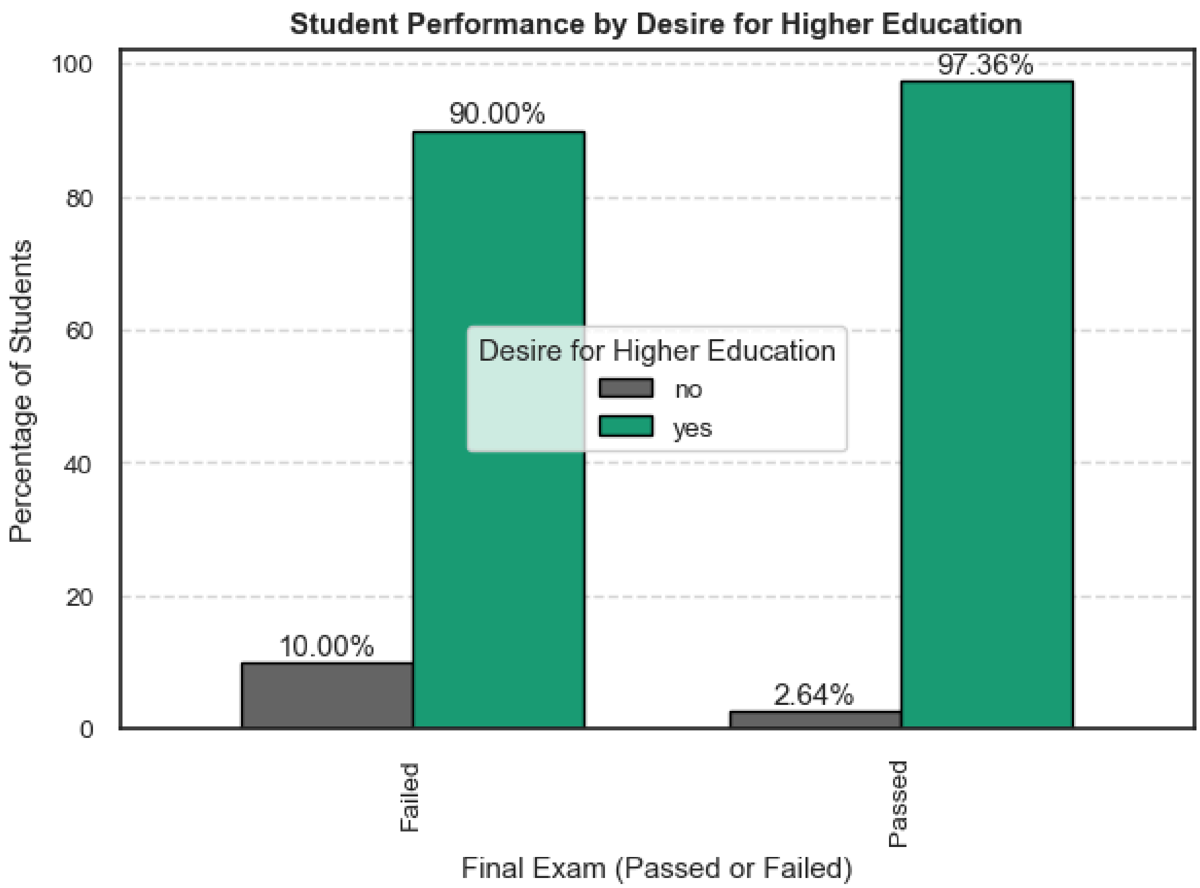


Mother Education Level had a good impact in student status of either passing or failing.

In [22]:
```
# Crosstab for 'passed' and 'higher'
higher_tab = pd.crosstab(index=df_copy['passed'], columns=df_copy['higher'])

# Calculate percentages for each category
higher_perc = higher_tab.apply(lambda x: x / sum(x) * 100, axis=1)

# Plot the bar chart
plt.figure(figsize=(14, 7))
higher_perc.plot(kind='bar', colormap="Dark2_r", edgecolor='black', width=0.7)

# Add titles and labels
plt.title('Student Performance by Desire for Higher Education', weight='bold')
plt.xlabel('Final Exam (Passed or Failed)')
plt.ylabel('Percentage of Students')

# Adjust legend
plt.legend(title='Desire for Higher Education', loc='center')

# Adding percentage labels on top of each bar
for container in plt.gca().containers:
    plt.gca().bar_label(container, fmt='%.2f%%', label_type='edge', padding=.5)

# Customize tick labels
plt.xticks(ticks=[0, 1], labels=['Failed', 'Passed'])
plt.yticks()
```

```
# Add a grid for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adjust the layout to prevent overlap
plt.tight_layout();

plt.show();
```

```
<Figure size 1400x700 with 0 Axes>
```

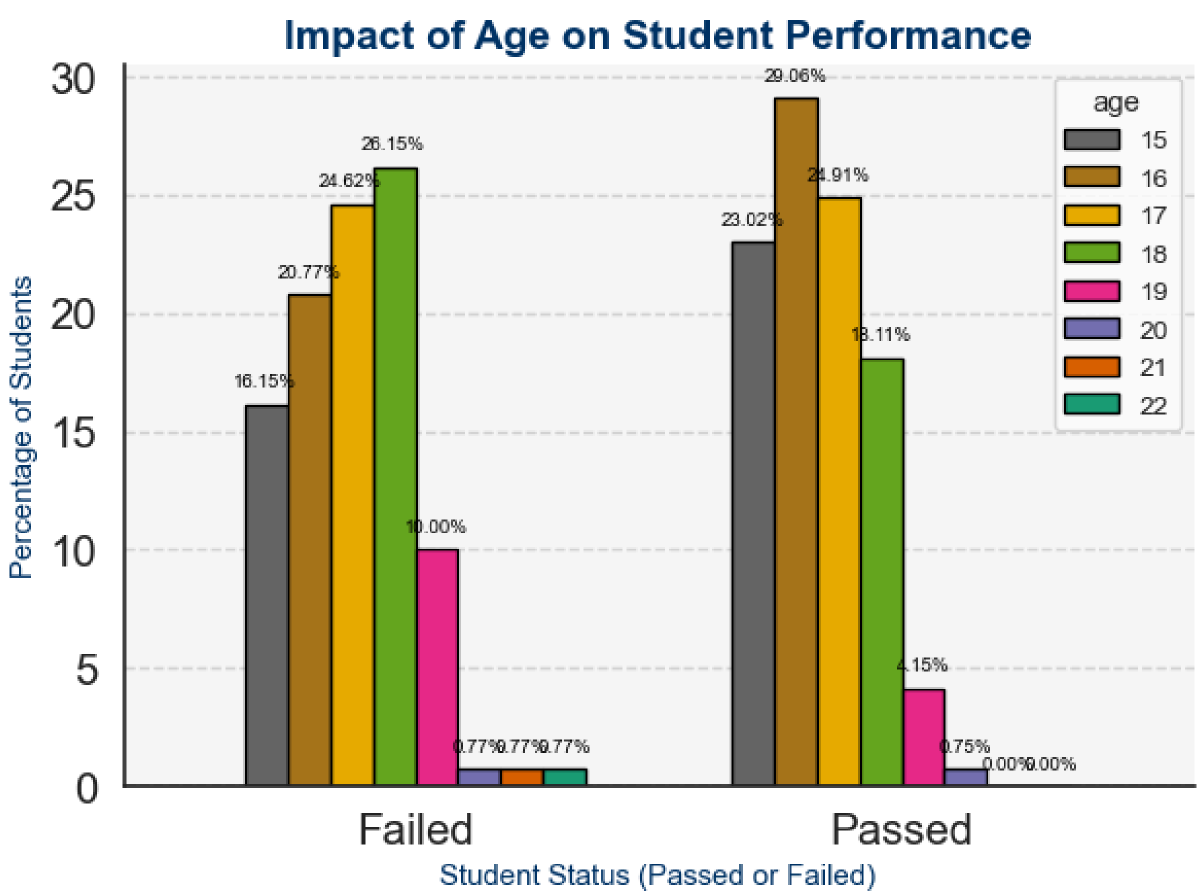


Desire to take Higher Education played a strong part in students passing the final Exam with 97.36% of students who passed the final Exam opting to take Higher Education. It could be a good idea to encourage your kids or students to take higher education.

In [32]:
```
# Crosstab for 'passed' and 'age'
age_tab = pd.crosstab(index=df_copy['passed'], columns=df_copy['age'])

# Calculate percentages for each category
age_perc = age_tab.apply(lambda x: x / sum(x) * 100, axis=1)

# Set figure size larger for better visualization
plt.figure(figsize=(18, 10))
ax = age_perc.plot(kind='bar', colormap="Dark2_r", edgecolor='black', fontsize=16,

# Add titles and labels
plt.title('Impact of Age on Student Performance', weight='bold', color='#003366', f
plt.xlabel('Student Status (Passed or Failed)', color='#003366')
```

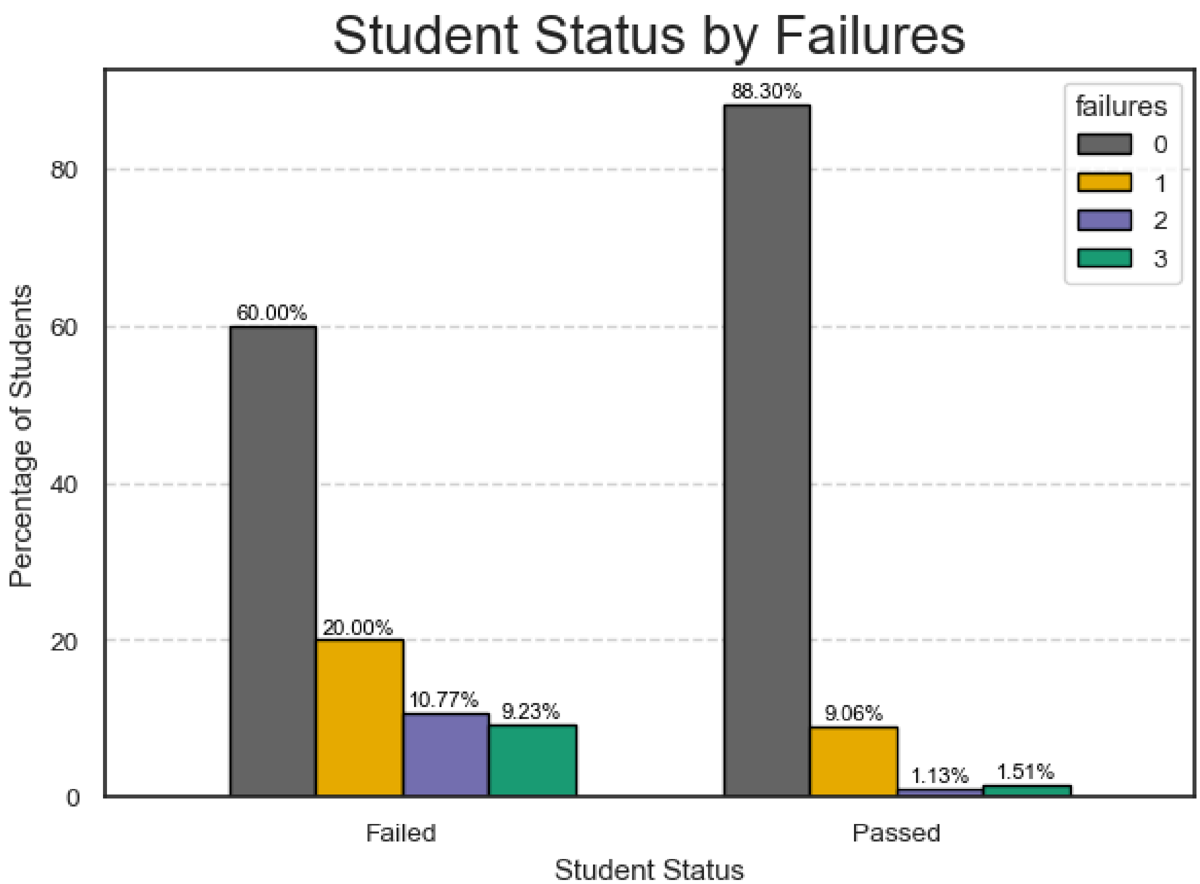
```
plt.ylabel('Percentage of Students', color='#003366')

# Adjust legend
# plt.legend(title='Age', title_fontsize=18, fontsize=16, loc='upper right', frameo

# Adding percentage labels with smaller font size
for container in ax.containers:
    ax.bar_label(container, fmt='%.2f%%', label_type='edge', fontsize=7, padding=5,

# Customize tick labels
plt.xticks(ticks=[0, 1], labels=['Failed', 'Passed'], rotation=0)
plt.yticks()

# Add gridlines for better readability and set line properties
plt.grid(axis='y', linestyle='--', alpha=0.8)

# Adjusting the background to make the plot visually appealing
ax.set_facecolor('#F5F5F5')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

# Tight layout for preventing overlap
plt.tight_layout()

plt.show();
```

```
<Figure size 1800x1000 with 0 Axes>
```



In [35]:
```
# Crosstab for 'passed' and 'failures' using df_copy
fail_tab = pd.crosstab(index=df_copy['passed'], columns=df_copy['failures'])

# Calculate percentages for each category
fail_perc = fail_tab.apply(lambda x: x / sum(x) * 100, axis=1)

# Set figure size and create bar plot
plt.figure(figsize=(16, 8))
ax = fail_perc.plot(kind='bar', colormap="Dark2_r", edgecolor='black', width=0.7)

# Add title with larger font size
plt.title('Student Status by Failures', fontsize=20)

# Add axis labels without font size modification
plt.xlabel('Student Status')
plt.ylabel('Percentage of Students')

# Adding percentage labels on bars with default font size
for container in ax.containers:
    ax.bar_label(container, fmt='%.2f%%', label_type='edge', padding=1, color='blac

# Customize tick labels and gridlines
plt.xticks(ticks=[0, 1], labels=['Failed', 'Passed'], rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.8)

# Adjusting the layout to prevent overlap
plt.tight_layout()

plt.show();
```

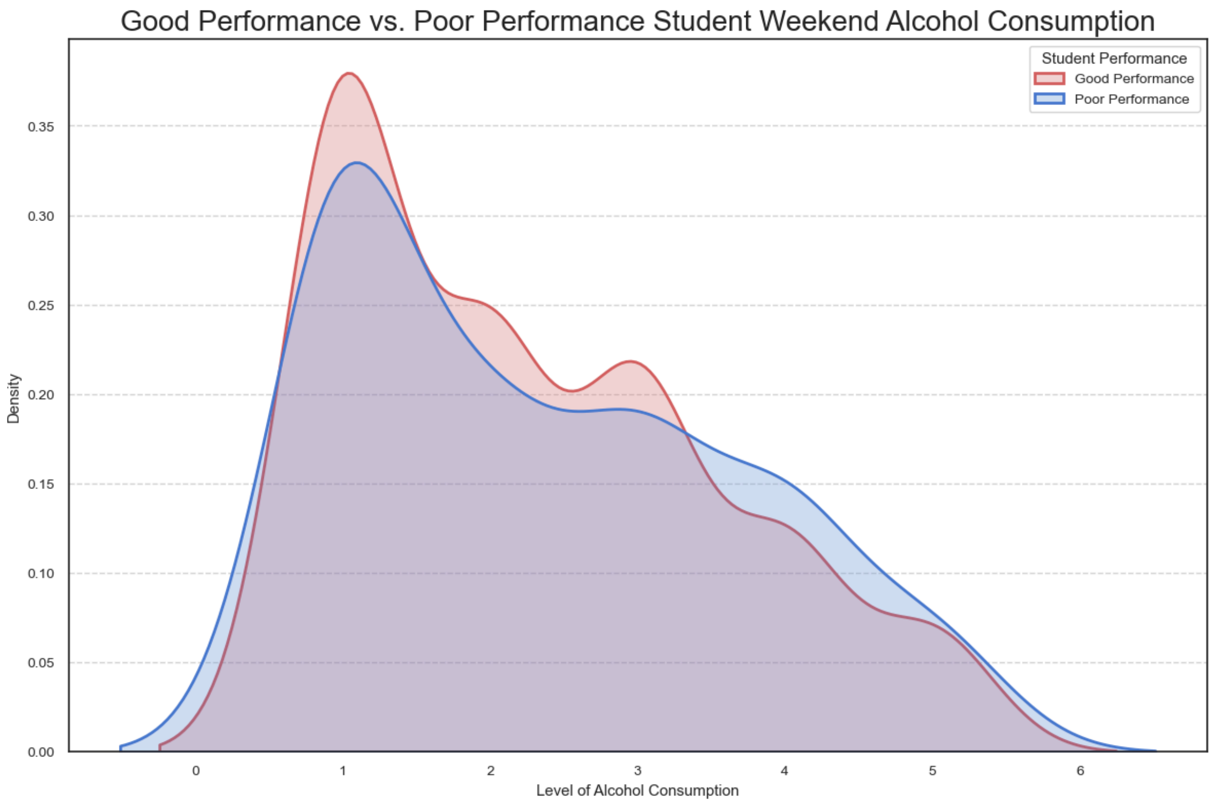
```
<Figure size 1600x800 with 0 Axes>
```



Most of the students that passed the final exam has no failures (88.30%) while the majority of those that failed the final exam had the highest number of failures

In [38]:
```
# Create good student dataframe
good = df_copy.loc[df_copy['passed'] == 'yes']
good['good_alcohol_usage'] = good['Walc']

# Create poor student dataframe
poor = df_copy.loc[df_copy['passed'] == 'no']
poor['poor_alcohol_usage'] = poor['Walc']

# Set figure size and plot KDE
plt.figure(figsize=(12, 8))
sns.kdeplot(good['good_alcohol_usage'], shade=True, color="r", label="Good Performa
sns.kdeplot(poor['poor_alcohol_usage'], shade=True, color="b", label="Poor Performa

# Add plot title with a larger font size
plt.title('Good Performance vs. Poor Performance Student Weekend Alcohol Consumptio

# Add labels to axes without font size modification
plt.ylabel('Density')
plt.xlabel('Level of Alcohol Consumption')

# Add a legend with proper placement
plt.legend(title="Student Performance", loc='upper right')

# Customize gridlines and layout for better appearance
```

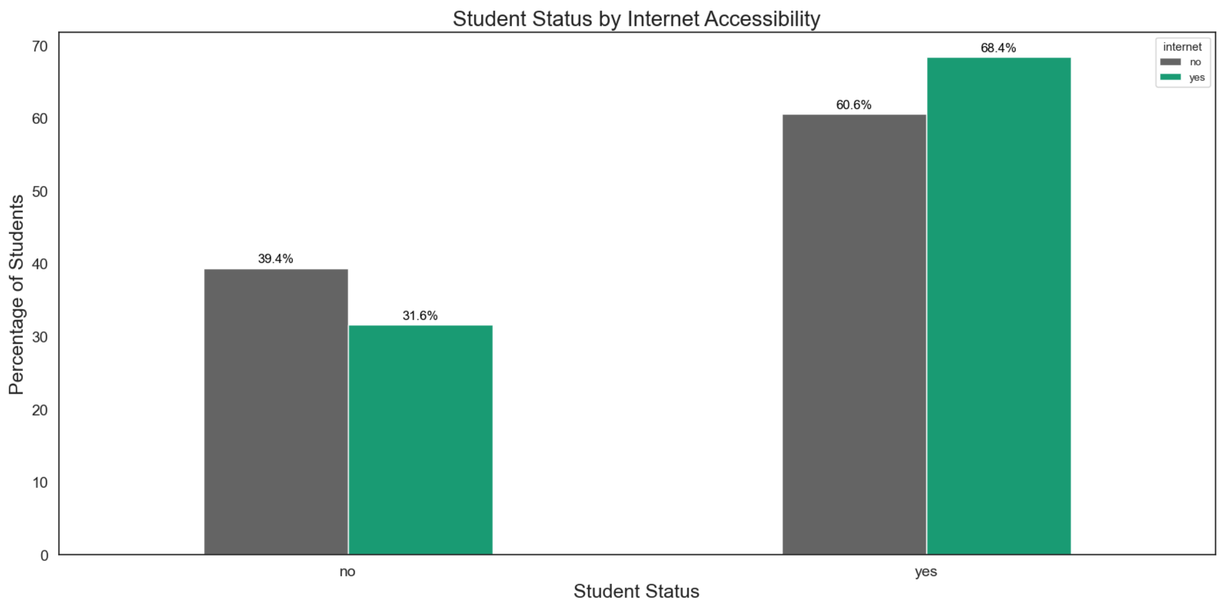
```
plt.grid(axis='y', linestyle='--', alpha=0.8)
plt.tight_layout()

plt.show();
```



For weekely alchool consumption it doesn't have an strong impact on student performance .Even people with low consumption had low grad.

In [49]:
```
# Define the perc function to calculate percentages
def perc(x):
    return x / x.sum() * 100

# Create the crosstab for student status by internet accessibility
alc_tab = pd.crosstab(index=df_copy['passed'], columns=df_copy['internet'])

# Calculate percentages
alc_perc = alc_tab.apply(perc)

# Plot the bar chart
ax = alc_perc.plot.bar(colormap="Dark2_r", figsize=(16,8))

# Add plot title
plt.title('Student Status by Internet Accessibility', fontsize=20)

# Add labels to axes
plt.xlabel('Student Status', fontsize=18)
plt.xticks(rotation=0, fontsize=14)
plt.ylabel('Percentage of Students', fontsize=18)
plt.yticks(fontsize=14)

# Add percentage values on top of the bars
```

```
for p in ax.patches:
    ax.annotate(f'{p.get_height():.1f}%', (p.get_x() + p.get_width() / 2., p.get_he
                ha='center', va='baseline', fontsize=12, color='black', xytext=(0,
                textcoords='offset points')

# Customize layout for better visual appeal
plt.tight_layout()

# Show the plot
plt.show();
```

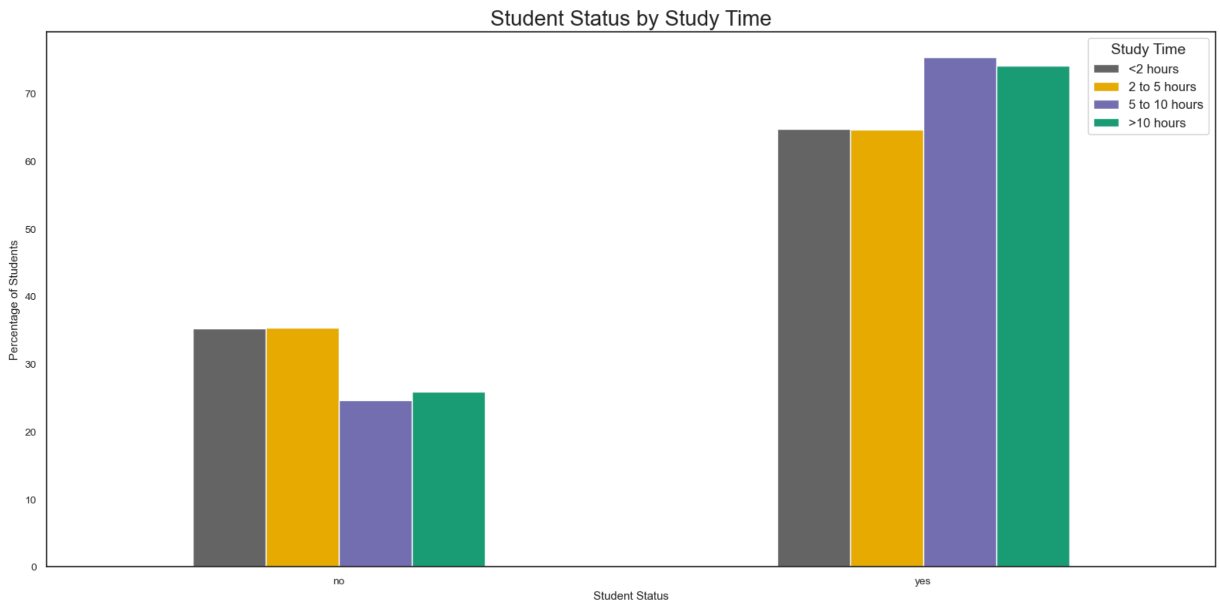


A large majority of the students that passed had Internet accesibility while the majority of the students that failed lacked internet accessibility.

In [55]:

```
# Create the crosstab for student status by study time
stu_tab = pd.crosstab(index=df_copy['passed'], columns=df_copy['studytime'])

# Calculate percentages
stu_perc = stu_tab.apply(perc)

# Plot the bar chart
ax = stu_perc.plot.bar(colormap="Dark2_r", figsize=(16,8))

# Add plot title
plt.title('Student Status by Study Time', fontsize=20)

# Add labels to axes
plt.xlabel('Student Status')
plt.xticks(rotation=0)
plt.ylabel('Percentage of Students')

# # Add percentage values on top of the bars
# for p in ax.patches:
#     ax.annotate(f'{p.get_height():.1f}%', (p.get_x() + p.get_width() / 2., p.get_
#                 ha='center', va='baseline', fontsize=12, color='black', xytext=(0
#                 textcoords='offset points')
```

```
# Customize the legend with the mapped values for study time
study_time_legend = {
    1: '<2 hours',
    2: '2 to 5 hours',
    3: '5 to 10 hours',
    4: '>10 hours'
}
handles, labels = ax.get_legend_handles_labels()
labels = [study_time_legend[int(label)] for label in labels]
ax.legend(handles, labels, title='Study Time', fontsize=12, title_fontsize=14)

# Customize layout for better visual appeal
plt.tight_layout()

# Show the plot
plt.show();
```



Most of people who passed the exam study 5-10 hours weekely

General conclusion from the EDA

Summary:

After dealing with the most relevent features ,the valedictorian of an exellents conditions for heigh academic potentials is likely to have this profile:

1.Does not go out with friend frequently

2.Is not in romantic relation

3.Parents receive higher education specialy woman

4.Have strong desire to receive higher education

5.Mother is a health care professional

6.father is a teacher

7.No absences to classes

8.have access to internet

9.study more than 10 hours a week

10.Is healthy

Machine Learning Model Development

1: Logistic Regression Model

Model Implemetation

In [61]:
```
# Initializing the scaled data to be used fr the model development
df_scaled
```

Out[61]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob
0	0	0	1.023046	1	0	0	1.143856	1.360371	0	4
1	0	0	0.238380	1	0	1	-1.600009	-1.399970	0	2
2	0	0	-1.330954	1	1	1	-1.600009	-1.399970	0	2
3	0	0	-1.330954	1	0	1	1.143856	-0.479857	1	3
4	0	0	-0.546287	1	0	1	0.229234	0.440257	2	2
...	...	...	...	...	...	...	...	...	...	...
390	1	1	2.592380	1	1	0	-0.685387	-0.479857	3	3
391	1	1	0.238380	1	1	1	0.229234	-1.399970	3	3
392	1	1	3.377047	0	0	1	-1.600009	-1.399970	2	2
393	1	1	1.023046	0	1	1	0.229234	-0.479857	3	2
394	1	1	1.807713	1	1	1	-1.600009	-1.399970	2	0

395 rows × 31 columns

In [63]:
```
# Split the data into features and target
X = df_scaled.drop('passed', axis=1)
y = df_scaled['passed']
```

```
print('The dimensions of the features are:', X.shape)
print('The dimensions of the target are:', y.shape)
```

```
The dimensions of the features are: (395, 30)
The dimensions of the target are: (395,)
```

In [64]:
```
y.value_counts()
```

Out[64]:
```
passed
1    265
0    130
Name: count, dtype: int64
```

In [65]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [66]:
```
# Initialize the Logistic Regression model
lr_model = LogisticRegression(random_state=42)
```

In [67]:
```
# Fitting the model on the training data
lr_model.fit(X_train, y_train)
```

Out[67]:
```
LogisticRegression
LogisticRegression(random_state=42)
```

In [68]:
```
# Predicting the target values
y_pred = lr_model.predict(X_test)
y_pred
```

Out[68]:
```
array([0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1])
```

Model Evaluation

In [69]:
```
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_cur
)
```

In [81]:
```
def evaluate_logistic_regression(model, X_test, y_test):
    """
    Evaluates a logistic regression model on the test data using various metrics.

    Parameters:
    - model: Trained logistic regression model (must have .predict and .predict_pro
    - X_test: Test features
    - y_test: True labels for the test set

    Returns:
    - A dictionary containing all the evaluation metrics
```

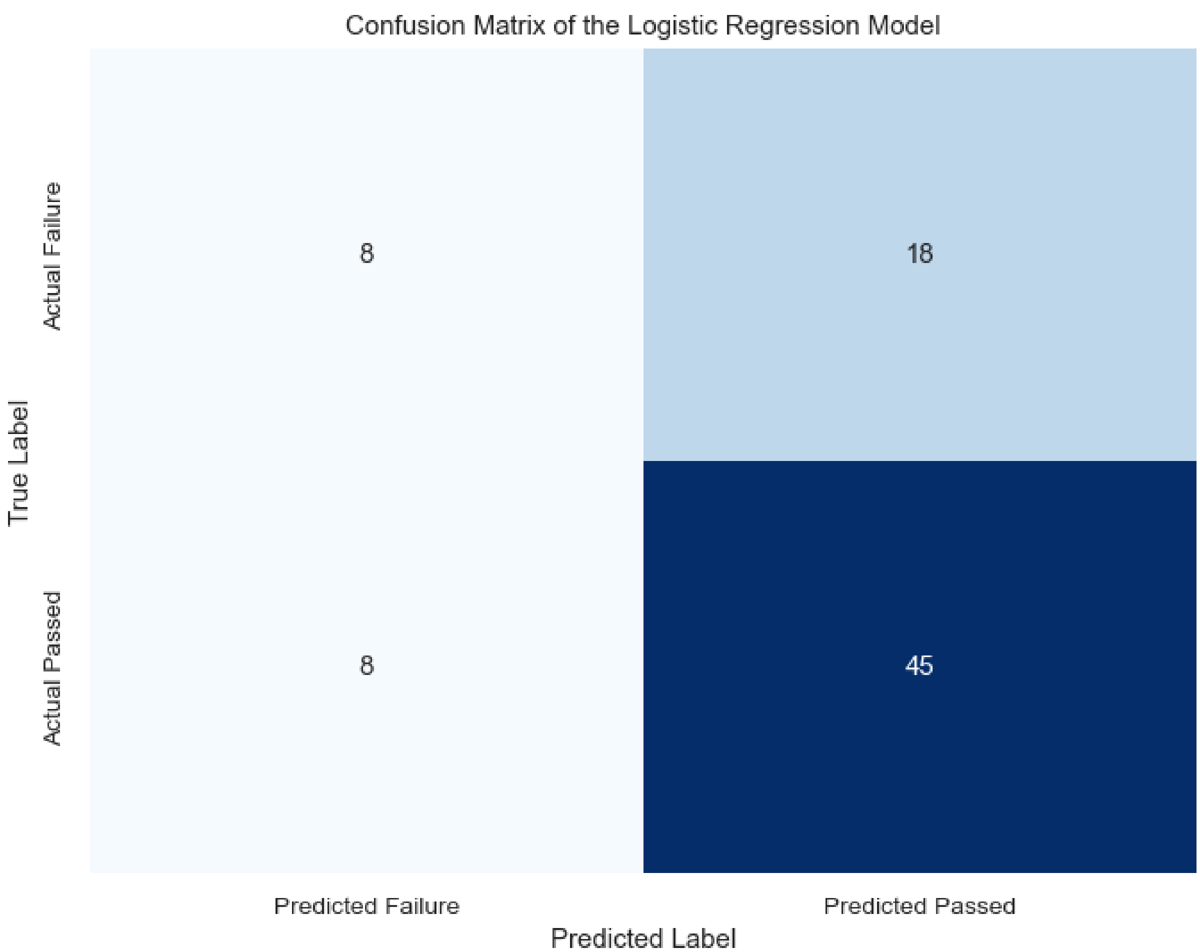
```
    """
    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Calculate metrics and convert to percentage
    accuracy = accuracy_score(y_test, y_pred) * 100
    f1 = f1_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred) * 100
    recall = recall_score(y_test, y_pred) * 100
    roc_auc = roc_auc_score(y_test, y_pred_proba) * 100

    # Print the metrics with percentage signs
    print(f"Accuracy Score: {accuracy:.2f}%")
    print(f"F1 Score: {f1:.2f}%")
    print(f"Precision Score: {precision:.2f}%")
    print(f"Recall Score: {recall:.2f}%")
    print(f"ROC-AUC Score: {roc_auc:.2f}%")

    # Print detailed classification report
    print('\n\nClassification Report:')
    print(classification_report(y_test, y_pred))


    # Display confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(
        cm,
        annot=True,
        fmt='d',
        cmap='Blues',
        cbar=False,
        xticklabels=['Predicted Failure', 'Predicted Passed'],
        yticklabels=['Actual Failure', 'Actual Passed']
    )
    plt.title('Confusion Matrix of the Logistic Regression Model')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
    plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show();
```

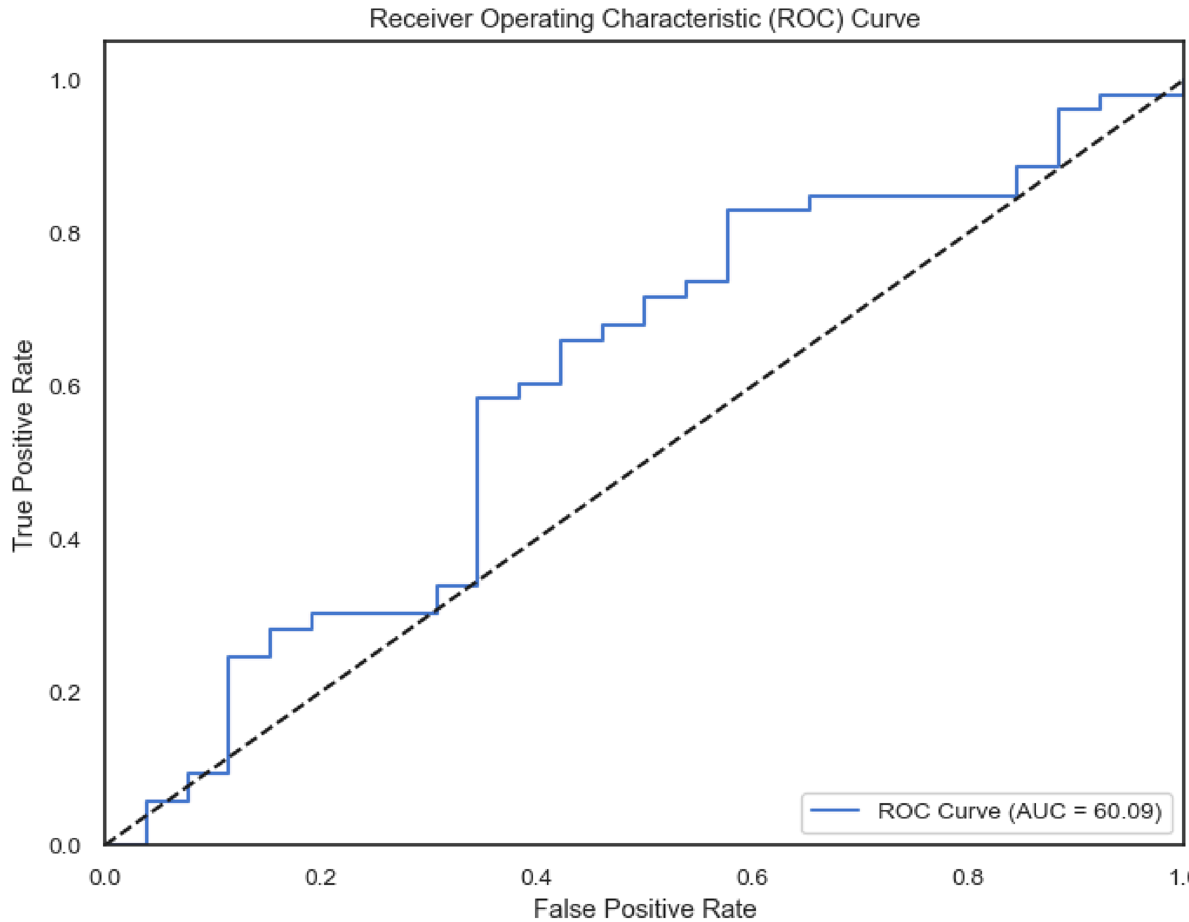
In [82]:

```
# Evaluate the model using the function
```

```
metrics = evaluate_logistic_regression(lr_model, X_test, y_test)
```

```
Accuracy Score: 67.09%
F1 Score: 77.59%
Precision Score: 71.43%
Recall Score: 84.91%
ROC-AUC Score: 60.09%


Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.31      0.38        26
           1       0.71      0.85      0.78        53

    accuracy                           0.67        79
   macro avg       0.61      0.58      0.58        79
weighted avg       0.64      0.67      0.65        79
```




Receiver Operating Characteristic (ROC) Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC Curve (AUC = 60.09)

```
In [ ]:
```