Modeling Student Success: How Lifestyle and Demographics Affect Academic Performance

Team 2: Paul Ancalima, Joseph Edwards, Erika Gallegos

Applied Artificial Intelligence, University of San Diego

AAI 500-02: Probability and Statistics for Artificial Intelligence

Prof. Leonid Shpaner

October 20, 2025

Modeling Student Success: How Lifestyle and Demographics Affect Academic Performance

Abstract: This paper examines how lifestyle habits and demographic factors influence student academic performance using a dataset of 1,000 student records, which were retrieved from Kaggle. Fourteen behavioral and demographic variables were analyzed through exploratory data analysis, linear regression, and binary logistic regression to predict both continuous exam scores and exam pass/fail outcomes. Results indicate that behavioral factors, particularly studying, attending class, sleeping, exercising, and mental health are strong positive predictors of academic success, while watching Netflix and using social media negatively impact academic performance. In contrast, demographic variables, such as age, gender, and parental education, show minimal influence. The linear regression model achieved a strong fit (adj-$R^2 = 0.90$ on test data) and similarly, the logistic model classified pass/fail outcomes with 86.8% accuracy on the unseen test data. These findings demonstrate that consistent study habits and well-being practices play a more critical role in student achievement than background characteristics, suggesting that both educational interventions and students' focus should prioritize behavioral and wellness-based strategies to improve outcomes.

Introduction

Student academic performance is a central concern across all levels of education for students, educators, and policy makers. For students, performance influences future opportunities, career readiness, and long-term socioeconomic outcomes. For educators and policy makers, student performance shapes graduation rates, institutional evaluations, and broader equity initiatives. Hence, identifying the habits and background factors that affect student performance is critical for designing effective interventions, advising students, and shaping educational policy.

Previous studies have examined how study habits influence learning outcomes. Nguyen et al. (2018) found that high-performing students tend to study proactively, while low-performing students spend more time catching up. Similarly, Bonsaksen et al. (2017) reported that students with higher GPAs engage more in self-directed study activities. However, not all studying is equally effective, and time alone reaches a point of diminishing returns. Deep, meaningful engagement with content appears to drive better academic results than surface-level review (Everaert et al., 2017). Sleep patterns have also been linked with academic performance, where students with consistent sleep schedules are associated with higher GPAs, and longer sleep duration tends to improve mood and well-being in students (Hershner, 2020). Class attendance is another well-established predictor, as consistent participation is associated with higher student grades (Tetteh, 2018).

Other lifestyle and extracurricular activities have drawn attention as well. However, research on the effects of video game use has yielded mixed findings. Some studies report that heavy gaming correlates with lower academic performance (Adzic et al., 2023; Adelantado-Renau et al., 2019), while others suggest potential benefits, such as using gaming as a self-reward system to motivate studying (Adzic et al., 2021) or supporting foreign language development (Martinez et al., 2022).

In addition to behavioral habits, demographic and background characteristics also play an important role. A study on 712 undergraduate occupation therapy students showed that older aged and female students were associated with higher GPA (Bonsaksen et al., 2017). However, age and life stage may further moderate performance, particularly when older or non-traditional students balance additional responsibilities. Additionally, parental education and socioeconomic status are long-standing predictors of achievement, likely due to differences in available resources, support systems, and academic expectations (Stull, 2013). For example, positive correlations have been identified between students' academic achievement and parents' education (Idris et al., 2020).

Although many studies have explored these predictors individually, fewer have examined them together to provide a comprehensive understanding of academic performance. Study habits are often interrelated; for example, students who attend class regularly may also study more consistently and spend less time on social media. Exploring these factors collectively can provide a more complete picture of what drives achievement. This can help students, educators, and policymakers identify which combinations of behaviors and background characteristics most strongly influence outcomes.

To this end, this paper uses a publicly available dataset of student habits and academic performance to explore these relationships in a structured way. We first provide a descriptive overview of students' background and behavioral characteristics, then analyze the associations among predictors to understand how these factors relate to one another. We subsequently use linear regression to predict continuous exam scores, and logistic regression to predict pass/fail outcomes, comparing the influence of demographic and behavioral predictors across both models. All analysis and source code for this project are publicly available on GitHub at https://github.com/USD-AAI-500-Stats-Group-2/Stats500FinalProject/.

Research Objectives

To connect these goals with our analytical plan, we define the following research questions to structure the study and focus the analyses:

- *RQ1:* What patterns of association exist among students' habits and background factors?
- *RQ2:* How do student habits and demographic variables affect exam scores?
- *RQ3:* What factors are most influential in determining whether a student achieves a passing versus failing exam grade?

Data Cleaning and Preparation

Data Overview

The dataset was obtained from Kaggle (Nath, 2025) and was most recently updated approximately six months prior to this analysis, ensuring that it represents a current dataset. Although the dataset is based on synthetic, simulated data, it was generated using realistic patterns and distributions commonly observed in student populations. The author of this dataset leveraged this simulated, yet realistic, approach to avoid privacy and ethical concerns associated with using real student data. The dataset contains 1,000 student records with 14 variables capturing a range of behavioral and wellness factors, along with final exam scores as an outcome measure. The structure of the data, with its combination of a relatively large number of observations, numerous variables, and variety of data types, makes it particularly suitable for exploratory data analysis and regression modeling.

Independent Variables

There were 14 variables considered for their effects on student performance, representing a variety of habits and demographics factors. An overview of these variables is provided in Table 1.

Table 1

Independent Variables

Variable Name	Definition	Variable Type
Age	Years old	Continuous
Gender	Male, female, or other	Nominal
Parental Education level	None, high school, bachelor, or master	Ordinal
Attendance Percentage	Class attendance from 0-100%	Continuous
Study Hours	Average daily study time	Continuous
Social Media Hours	Average daily social media time	Continuous
Netflix Hours	Average daily Netflix/binging time	Continuous
Sleep Hours	Average daily sleep	Continuous
Mental Health Rating	Rating from 1 (poor) to 10 (good)	Continuous
Part-Time Job	Has a part-time job, yes/no	Nominal
Extracurricular Participation	Participates in extracurricular activities, yes/no	Nominal
Diet Quality	Poor, fair, or good	Ordinal
Exercise Frequency	Average number of times per week	Continuous
Internet Quality	Poor, average, good	Nominal

Dependent Variables

Student performance was measured using two variables: *Exam Score,* a continuous variable representing their final exam score from 0 to 100; and *Pass/Fail,* a derived binary variable based on if their final exam score was greater than or equal to 70 (pass) or less than 70 (fail).

Data Cleaning and Quality

During data cleaning, we identified potential missing values in the Parental Education Level column; specifically, 91 instances labeled as “None.” It was unclear whether “None” meant that the parents had no formal education or if it represented missing data automatically assigned as NaN by Python. Since the dataset documentation did not confirm either case, we treated these as true missing values (NaN). To test whether this variable still held predictive value, we ran a t-test comparing exam scores of students with and without recorded parental education levels. The results showed no significant difference ($p = 0.79$), indicating that parental education had little influence on exam scores. Hence, we decided to err on the side of caution and assume these values as missing to ensure that our conclusions were not based on unwarranted assumptions.

The dataset contained six categorical variables (e.g., gender, diet quality, etc.). These variables were transformed to enable their use in the regression models. First, the levels of each ordinal variable were reordered so that the reference category corresponded to the lowest level and the remaining categories followed the correct order (e.g., poor, fair, good). This ensured that the dropped reference category in the models would consistently represent the lowest level. Then, these categorical variables were converted into numerical dummy variables (i.e., 0, 1) using pandas.

The dataset was randomly split into a training set (80%, N = 800) and a testing set (20%, N = 200). This split allowed the regression models to be trained on the majority of the data, while reserving a separate portion for evaluating performance on unseen observations. This approach helps assess how well the models generalize beyond the data they were fitted to, reducing the risk of overfitting. The random split was performed to ensure that both sets were representative of the overall dataset, preserving the distribution of key variables.

Exploratory Data Analysis

Descriptive Statistics

Table 2 presents a preview of the dataset used in the analysis. For readability, only the most relevant variables are shown, including study habits (study hours, social media hours, Netflix hours, sleep hours), background characteristics (gender, parental education, internet quality), and the outcome variable, exam score.

Table 2

Sample of Data

age	gender	study_ hours _per_day	social_ media_ hours	netflix_ hours	sleep_ hours	parental_ education_ level	internet_ quality	mental_ health_ rating	exam_ score
23	Female	0	1.2	1.1	8	Master	Average	8	56.2
20	Female	6.9	2.8	2.3	4.6	High School	Average	8	100
21	Male	1.4	3.1	1.3	8	High School	Poor	1	34.3
23	Female	1	3.9	1	9.2	Master	Good	1	26.8
19	Female	5	4.4	0.5	4.9	Master	Good	1	66.4

The mean exam score was approximately 69.6 (SD = 16.9), indicating moderate variation in student performance. The 95% confidence interval for the population mean exam score was (68.6, 70.6), suggesting high precision in this estimate due to the large sample size (n = 1000). The distribution of exam scores plot (Figure 1 – left) shows that most students scored between 60 and 80, with a roughly normal distribution and a slight left skew, indicating that a small number of students performed exceptionally well.

The study hours vs exam score scatterplot (Figure 1 – right) reveals a strong positive linear relationship, where students who dedicate more time to studying tend to achieve higher exam scores. The clustering near the upper range of study hours further reinforces the importance of consistent study habits.

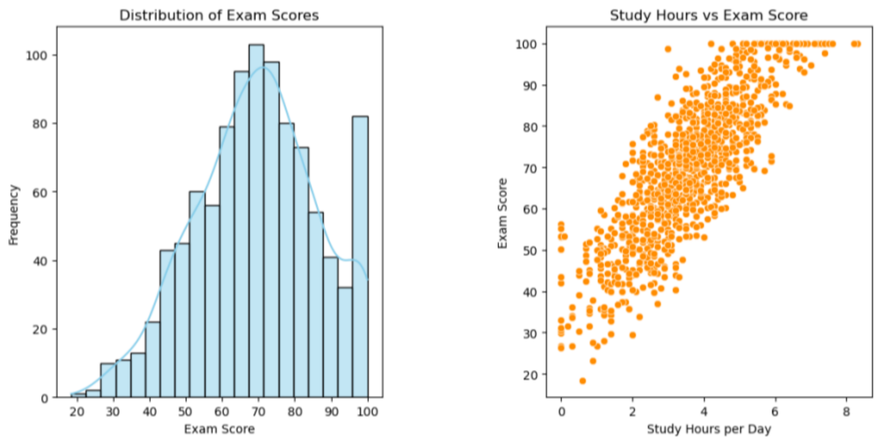


Figure 1. Distribution of exam scores (left) and relation to study hours (right).

The exam score by parental education boxplot (Figure 2 – left) suggests only small variations across education levels, implying that parental education has minimal direct effect on student exam performance in this dataset.

The exam score by gender boxplot (Figure 2 – right) shows nearly overlapping distributions among male, female, and other gender categories, suggesting no significant difference in performance between groups.

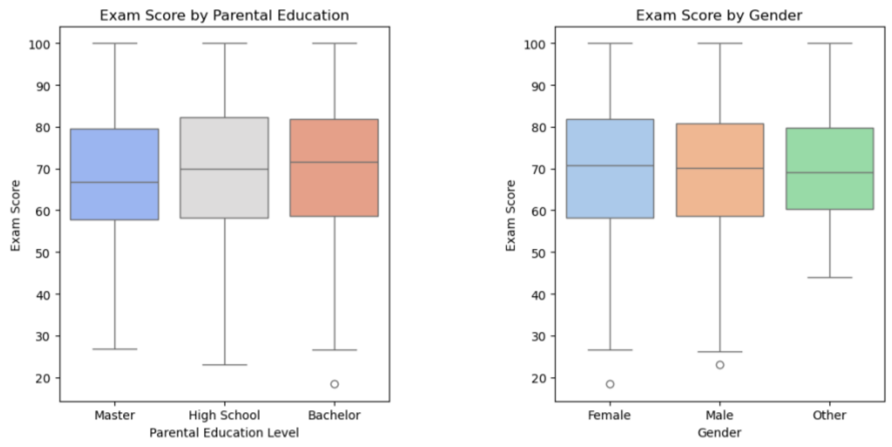


Figure 2. Exam scores by parental education (left) and gender (right).

Overall, the exploratory analysis helps to understand variable distributions and potential associations to investigate further. These findings guided the selection of variables for further regression modeling.

Correlations

A correlation analysis was performed across the continuous variables. The analysis revealed that exam scores were most strongly associated with study hours per day (r = 0.83), showing a clear and strong positive relationship, students who studied more achieved higher scores. Mental health rating also showed a moderate positive correlation with exam performance (r = 0.32), suggesting that students with better mental well-being tended to perform better academically. Smaller positive relationships were observed for exercise frequency (r = 0.16), sleep hours (r = 0.12), and attendance percentage (r = 0.09). In contrast, Netflix hours (r = –0.17) and social media hours (r = –0.17) were weakly negatively correlated with exam scores, indicating that higher leisure screen time was modestly linked to lower academic performance. Amongst the independent variables, there were not strong correlations between them, suggesting minimal multicollinearity and that each variable captures a distinct aspect of student behavior or background. The correlation heatmap (Figure 3) visually reinforced these patterns, emphasizing the importance of study habits and mental health while showing minimal relationships for age and entertainment-related features.

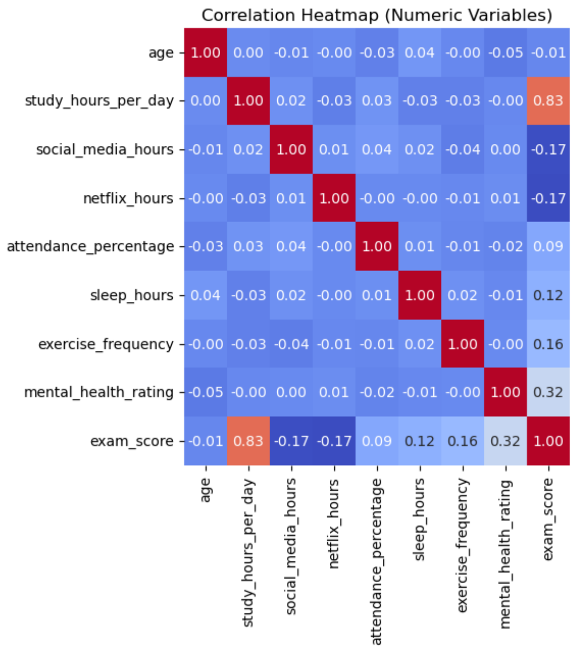


Figure 3. Correlation heatmap for continuous variables.

T-Tests

T-tests were conducted on low/high groupings of the continuous variables study hours and mental health ratings, for differences in exam scores. Study hours and mental health were each split based on median values, where students below the median were assigned to the low group, and students above the median were assigned to the high group, for the respective variable.

A t-test on study hours was conducted and showed that they positively affect exam scores. For this test, the hypotheses were as follows:

- H_0: Mean exam scores are equal between low and high study hour groups.
- H_1: Mean exam scores differ between low and high study hour groups.

The results of this t-test conclude that we can reject the null hypothesis ($p < 0.001$) and can be interpreted as study time having a large positive effect on exam score, see Figure 4 – left.

A t-test on mental health ratings was also conducted and showed that they positively affect exam scores. For this test, the hypotheses were as follows:

- H_0: Mean exam scores are equal between low and high mental health groups.
- H_1: Mean exam scores differ between low and high mental health groups.

The results of this t-test also conclude that we can reject the null hypothesis ($p < 0.001$) and can be interpreted as having a moderate positive effect on exam score, see Figure 4 – right.

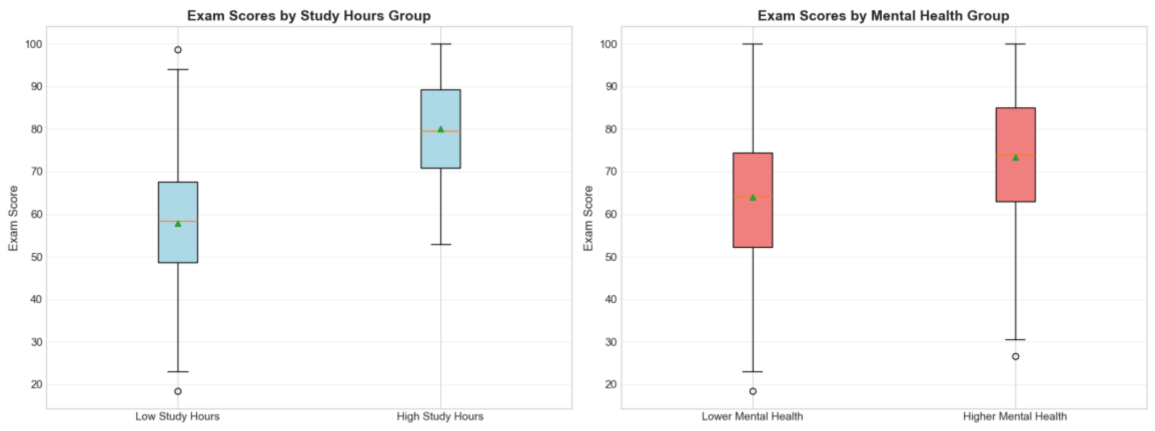


Figure 4. Exam scores by study hours (left) and mental health (right) by two groupings.

Lastly, a t-test comparing students with and without recorded parental education levels showed no significant difference in exam scores ($p = 0.79$). This suggests that missing values do not bias the linear regression model, since parental education level does not meaningfully affect continuous exam scores.

ANOVA Tests

The same variables, study hours and mental health ratings, were also analyzed through analysis of variance (ANOVA) tests. However, instead of splitting into two groups (i.e., low vs high), these were split into three groups per variable. Specifically, each variable was split into low (lower 33% quantile), medium (middle quantile), and high (above 67% quantile) groupings.

First, we examined the mental health ratings and their effect on exam scores, with the following hypotheses:

- H_0: Mean exam scores are equal between low, medium and high study hour groups.
- H_1: At least one group mean differs between low, medium and/or high study hour groups.

From the ANOVA test, we conclude that there is indeed a significant difference in means of scores by mental health rating ($p < 0.001$), showing us that the highest mental health group outperforms the others, see Figure 5 (left).

Next, we examined the study hours groups and their effect on exam scores, with the following hypotheses:

- H_0: Mean exam scores are equal between low, medium and high study hour groups.
- H_1: At least one group mean differs between low, medium and/or high study hour groups.

From the results of this test, we conclude that there is indeed a significant difference in means of scores by mental health rating ($p < 0.001$), showing us that the highest study hours group outperforms the others, see Figure 5 (right). Overall, these findings are consistent with our t-test analysis.

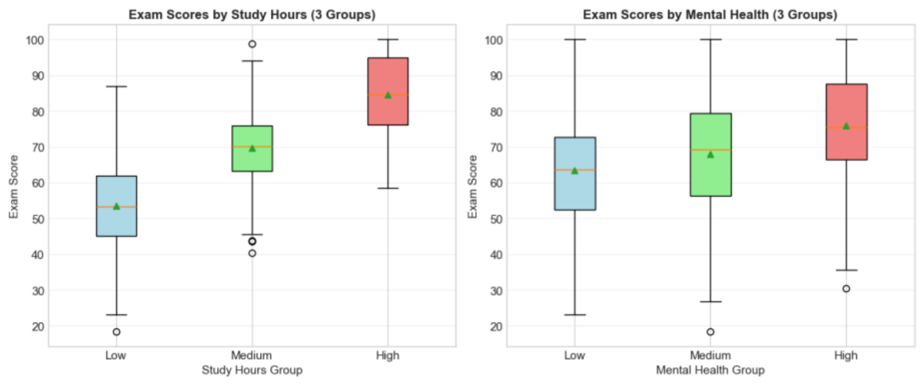


Figure 5. Exam scores by study hours (left) and mental health (right) by 3 groupings.

Chi-Square Analysis

For the chi-square analysis, we wanted to examine other variables that intuitively seemed like they could be related to exam score, but were not indicated as such through correlation analysis, thus, we selected diet quality.

To perform this test, we divided the students into three groups, but for exam score: the lower, middle, and higher third scores. Then, we examined the relationship between diet quality and exam scores, testing the following hypotheses:

- H_0: Exam Performance is independent of diet quality.
- H_1: Exam Performance is dependent on diet quality.

We conclude that we should fail to reject the null hypothesis again ($p < 0.001$), meaning that exam performance is not significantly associated with parental education level, see Figure 6.

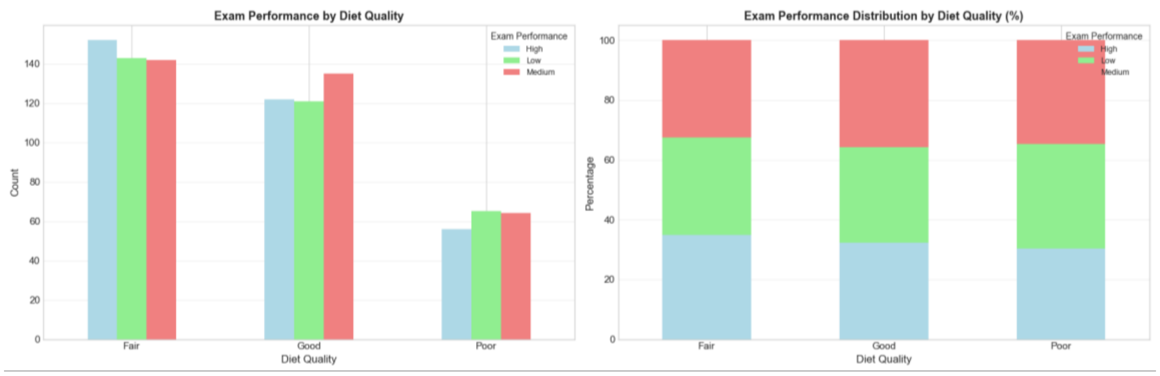


Figure 6. Exam score by diet quality.

Model Selection

Linear Regression

Since exam score is a continuous variable, we selected a linear regression model to predict students' exam scores. We fit the model on an 80% split of the data, then tested it on the remaining 20% unseen data.

After fitting the model, we assessed the output for goodness of fit and adherence to linear regression assumptions. We considered R^2 and adjusted R^2 for how well the model explained variability in the data. For goodness of fit, we compared predicted versus observed exam scores. To assess model assumptions, we performed several checks. First, we checked for linearity and homoscedasticity by plotting the predicted values versus the residuals, verifying that these assumptions were met because the residuals appeared randomly scattered around zero without clear patterns. We evaluated the normality of residuals using a histogram, a Q-Q plot, and the Shapiro-Wilk test - which yielded $p = 0.65$. All three indicated that the residuals were approximately normally distributed, supporting the validity of the linear regression model.

One interesting pattern we noticed was that mental health rating had a slightly lower coefficient than sleep hours in the linear regression model. This was not because mental health was less important, but because these two features were somewhat correlated. When predictors overlap in what they explain, something measured by the Variance Inflation Factor, it can reduce the apparent strength of each individual variable.

Binary Logistic Regression

We wanted to further investigate the data, particularly motivated by the question: what factors matter if all a student cares about is passing, rather than achieving a specific exam score. Note, this question is not motivated by our own ambitions (or lack of), but rather serves as a thought experiment to better understand the data.

To examine this, we created a new binary variable, pass (1) or fail (0), in which the resulting model would predict the likelihood of passing (1). Since this was a binary variable, we used the generalized linear model with the logit link function, specifically a binary logistic regression. Once again, using a random 80/20 split of the data for training and testing to perform this binary classification.

To assess model fit, we evaluated how well the model predicted and classified unseen data (i.e., test data). We created a confusion matrix to visualize the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). From this matrix, we computed our key metrics for classification performance: *1)* accuracy (proportion of correct predictions out of all predictions), *2)* precision (proportion of TP predictions out of all positive predictions), *3)* recall (proportion of TP predictions out of all actual positive predictions), *4)* specificity (proportion of TN predictions out of all actual negative predictions), and *5)* F1 score (harmonic mean of precision and recall). Lastly, we evaluated the Area Under the Receiver Operating Characteristic Curve (AUC), to assess the model's predictive power in distinguishing between pass/fail across different thresholds.

Model Analysis

Predicting Exam Score

This section seeks to answer research question 2: How do student habits and demographic variables affect exam scores. A linear regression model was fit on the training dataset to predict exam score using all 14 independent variables of interest, see Table 3. All variables were retained in this model, even if their individual effects on exam score were not statistically significant. This decision was made because even non-significant predictors can provide meaningful insights. For example, knowing that internet quality does not significantly affect exam performance is still useful, as it suggests that interventions targeting internet connectivity may be less impactful than those focusing on other factors.

Notably in the model, study hours, attendance percentage, sleep hours, exercise frequency, and mental health all had a positive effect on exam score. In contrast, social media hours and Netflix hours had a negative effect, indicating that increased time spent on these activities decreased exam performance. It is also noteworthy that study hours had the strongest effect, where each additional hour spent studying was associated with an average 9.50-point increase in the final exam score.

Table 3

Summary of Linear Regression Model Predicting Exam Score on Training Dataset

Variable	Coeff.	Std Error	t-statistic	p-value	95% CI [Lower, Upper]	
Intercept	8.24	2.95	2.79	0.005	2.45	14.02
Study Hours	9.50	0.13	72.71	< 0.001	9.25	9.76
Social Media Hours	-2.65	0.17	-16.02	< 0.001	-2.97	-2.32
Netflix Hours	-2.36	0.18	-13.03	< 0.001	-2.71	-2.00
Attendance Percentage	0.14	0.02	6.86	< 0.001	0.10	0.18
Sleep Hours	1.92	0.16	12.01	< 0.001	1.61	2.24
Exercise Frequency	1.49	0.09	15.76	< 0.001	1.30	1.67
Mental Health Rating	1.95	0.07	28.09	< 0.001	1.81	2.08
Age	-0.03	0.09	-0.34	0.732 (ns)	-0.20	0.14
Gender - Female	-0.26	0.40	-0.66	0.511 (ns)	-1.04	0.52
Gender - Other	0.41	0.96	0.43	0.669 (ns)	-1.48	2.30
Parent Edu. - Bachelor’s	0.05	0.43	0.11	0.911 (ns)	-0.80	0.90
Parent Edu. - Master’s	-0.21	0.55	-0.39	0.698 (ns)	-1.30	0.87
Part-Time Job - Yes	0.29	0.47	0.63	0.530 (ns)	-0.62	1.21
Extracurricular Partic. - Yes	-0.31	0.42	-0.75	0.452 (ns)	-1.13	0.50
Diet Quality - Fair	0.37	0.55	0.68	0.497 (ns)	-0.70	1.45
Diet Quality - Good	-0.24	0.56	-0.44	0.662 (ns)	-1.33	0.85
Internet Quality - Average	0.18	0.58	0.31	0.758 (ns)	-0.95	1.31
Internet Quality - Good	-0.54	0.57	-0.95	0.342 (ns)	-1.65	0.58

Model Fit: $R^2 = 0.901$; adjusted $R^2 = 0.899$; N = 800

The model fit the training data very well ($R^2 = 0.901$, adjusted $R^2 = 0.899$), and notably, performed equally well on the testing [unseen] data ($R^2 = 0.903$). Figure 7 illustrates this performance on the testing data by comparing the actual and predicted exam scores, where the red dashed line represents a 1:1 slope (i.e., perfect prediction). The close clustering of points around this line further demonstrates the model’s strong predictive accuracy on the unseen data.

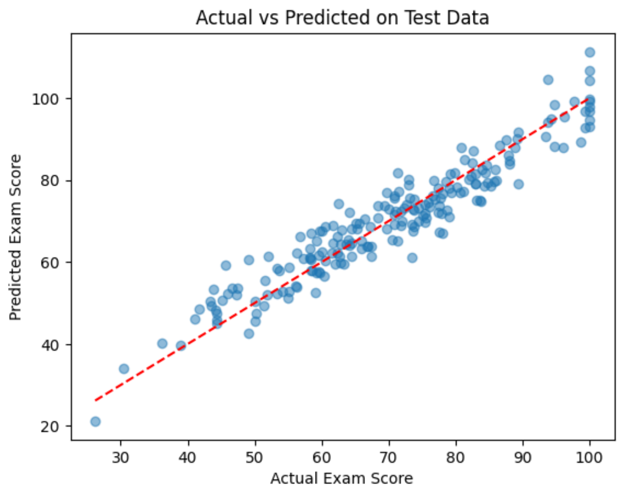


Figure 7. Goodness of fit of linear regression model on the testing data.

Predicting Pass/Fail

This section seeks to answer research question 3: What factors are most influential in determining whether a student achieves a passing versus failing exam grade. Not entirely surprising, the binary logistic regression model identified the same key predictors of exam performance as the linear regression model, see Table 4. Specifically, greater study hours, attendance percentage, sleep hours, exercise frequency, and mental health rating were all associated with a higher likelihood of passing the exam; while increased social media and Netflix hours reduced the likelihood of passing. Consistent with the linear model, study hours had the strongest effect on exam performance, where each additional hour of studying per day increased the odds of passing by 40.16 times ($e^{3.69}$), holding all other variables constant.

Table 4

Summary of Generalized Linear Model Predicting Likelihood of Passing Exam on Training Dataset

Variable	Coeff.	Std Error	z-statistic	p-value	Odds Ratio
Intercept	-23.77	2.91	-8.17	< 0.001	0.00
Study Hours	3.69	0.34	10.81	< 0.001	40.16
Social Media Hours	-1.00	0.16	-6.35	< 0.001	0.37
Netflix Hours	-0.95	0.17	-5.67	< 0.001	0.39
Attendance Percentage	0.07	0.02	4.20	< 0.001	1.07

Sleep Hours	0.79	0.13	6.02	< 0.001	2.20
Exercise Frequency	0.60	0.09	6.52	< 0.001	1.83
Mental Health Rating	0.82	0.09	9.57	< 0.001	2.27
Age	-0.08	0.07	-1.24	0.215 (ns)	0.92
Gender - Female	0.10	0.31	0.32	0.747 (ns)	1.11
Gender - Other	-1.01	0.68	-1.49	0.137 (ns)	0.37
Parent Edu. - Bachelor's	0.19	0.33	0.57	0.571 (ns)	1.21
Parent Edu. - Master's	0.11	0.44	0.24	0.807 (ns)	1.11
Part-Time Job - Yes	0.07	0.36	0.18	0.855 (ns)	1.07
Extracurricular Partic. - Yes	-0.16	0.33	-0.49	0.624 (ns)	0.85
Diet Quality - Fair	0.36	0.42	0.86	0.390 (ns)	1.44
Diet Quality - Good	-0.05	0.42	-0.11	0.914 (ns)	0.96
Internet Quality - Average	-0.56	0.44	-1.28	0.201 (ns)	0.57
Internet Quality - Good	-0.34	0.43	-0.79	0.431 (ns)	0.71
Model Fit: Pseudo R^2 = 0.62; N = 727					

Table 5 presents the confusion matrix for the model's performance on the [unseen] test data. From this, we can compute the accuracy (86.8%), precision (88.4%), recall (86.6%), specificity (87.1%), and F1 score (87.5%) of the model. All of these values indicate strong classification ability on the unseen data. Moreover, the model also achieved an AUC of 0.96 on the test data, further demonstrating that the model effectively distinguishes between students who pass and those who fail the final exam.

Table 5

Confusion Matrix of Actual vs Predicted Pass/Fail on Test Data

	Predicted Pass	**Predicted Fail**
Actual Pass	84 (TP)	13 (FN)
Actual Fail	11 (FP)	74 (TN)

Conclusion and Recommendations

This paper explored how lifestyle habits and demographic factors collectively influence student academic performance, offering a comprehensive view of success predictors in educational settings. Using both linear and logistic regression, we examined not only how these variables affect continuous exam scores but also their relationship to binary pass/fail outcomes. Across both models, consistent patterns emerged that highlight the powerful role of behavioral habits in driving academic achievement.

The analyses revealed that study hours, class attendance, sleep quality, exercise frequency, and mental health were the most influential predictors of student performance. Each of these factors demonstrated statistically significant positive effects on exam outcomes, indicating that academic success is multifaceted and rooted in consistent, healthy routines. Among these, study hours exerted the strongest influence, suggesting that structured and intentional study behaviors remain the most direct pathway to higher achievement. Conversely, increased social media use and Netflix viewing negatively affected performance, implying that digital distractions continue to pose significant challenges to effective learning.

Interestingly, demographic factors did not have a statistically significant impact on exam performance. While previous literature has suggested that demographics may play a role, our findings reinforce the notion that while social and educational background may influence outcomes, behavioral variables appear to have stronger immediate effects on students' academic success.

Interestingly, the independent variables showed little relationship with one another, indicating that each represents a distinct factor in student behavior or background. This suggests that interventions targeting one area may not necessarily influence others, highlighting the need for focused, individualized strategies to improve student performance.

Limitations of Existing Research

While this analysis provides valuable insights, several limitations must be acknowledged. The dataset used was synthetic, which, while generated based on real data, may not perfectly capture the variability of real-world student populations. Additionally, due to the data being observational rather than controlled and intervention based, we can identify associations but cannot definitively determine causality.

Directions for Future Research

Future research should expand both the scope of variables examined and the analytical methods used to model student performance. Although this paper focused on lifestyle and demographic factors, future analyses could incorporate additional variables that capture cognitive, environmental, and behavioral dimensions of learning. For instance, measures of types of studying or peer study engagement could provide richer context for understanding why some students achieve greater success despite similar habits. Incorporating institutional variables, such as class size or course modality, could also help disentangle individual versus structural influences on performance.

Analytically, future work could employ more complex statistical and machine learning techniques to uncover non-linear relationships and latent patterns not captured by traditional regression. For example, clustering algorithms could be used to identify distinct groups of students based on shared behavioral characteristics, revealing groups that respond differently to interventions. Similarly, decision trees or random forest models could help assess variable importance in a more flexible, data-driven way.

Overall Recommendations

For students, the findings suggest prioritizing consistent study routines, maintaining adequate sleep, exercising regularly, and monitoring mental health as essential strategies for academic success. Educators should consider incorporating wellness and time-management training into academic support programs, recognizing that performance is as much about health and habits as it is about intellect. Policymakers and institutions can use these insights to design evidence-based interventions that promote balanced student lifestyles, such as flexible schedules, mental health resources, and limits on excessive screen exposure, to foster environments conducive to learning and long-term success.

Use of Generative AI

Portions of this manuscript were refined using ChatGPT (GPT-5) to assist with sentence flow, structure, and consistency across sections written by the three authors. Short, human-written text segments (i.e., 1-2 sentences) were put into ChatGPT with the prompt to improve clarity, conciseness, grammar, and/or specific word choice. ChatGPT was not used to generate original content; rather, its suggestions were reviewed, edited, and integrated manually by the authors. In this use case, ChatGPT as a grammar tool was useful in ensuring a cohesive narrative across the combined work of the three authors.

References

Adelantado-Renau, M., Moliner-Urdiales, D., Cavero-Redondo, I., Beltran-Valls, M. R., Martinez-Vizcaino, V., Alvarez-Bueno, C. (2019). Association between screen media use and academic performance among children and adolescents: A systematic review and meta-analysis. *JAMA Pediatrics, 173*(11), 1058-1067. https://doi.org/10.1001/jamapediatrics.2019.3176

Adzic, S., Al-Mansour, J., Naqvi, H., & Stambolic, S. (2021). The impact of video games on students' educational outcomes. *Entertainment Computing, 38*, 100412. https://doi.org/10.1016/j.entcom.2021.100412

Adzic, S., Tot, T. S., Savic, V., Runic-Ristic, M., & Tot, V. Student achievement in relation to time spent studying and playing video games: A gender perspective. *Technology Education Management Informatics Journal, 2*, 832-839.

Bonsaksen, T., Brown, T., Lim, H. B., & Fong, K. (2017). Approaches to studying predict academic performance in undergraduate occupational therapy students: A cross-cultural study. *BMC Medical Educatio*n, *17*(76). https://doi.org/10.1186/s12909-017-0914-3

Everaert, P., Opdecam, E., & Maussen, S. (2017). The relationship between motivation, learning approaches, academic performance and time spent. *Accounting Education, 26(*1), 78-107. https://doi.org/10.1080/09639284.2016.1274911

Hershner, S. (2020). Sleep and academic performance: Measuring the impact of sleep. *Current Opinion in Behavioral Sciences, 33*, 51-56. https://doi.org/10.1016/j.cobeha.2019.11.009

Idris, M., Hussain, S., & Ahmad, N. (2020). Relationship between parents' education and their children's academic achievement. *Journal of Arts and Social Sciences, 7*(2), 82-92. https://doi.org/10.46662/jass-vol7-iss2-2020(82-92)

Martinez, L., Gimenes, M., & Lambert, E. (2022). Entertainment video games for academic learning: A systematic review. *Journal of Educational Computing Research*, 60(5). https://doi.org/10.1177/07356331211053848

Nath, J. (2025). *Student habits vs academic performance* [Dataset]. Kaggle. https://www.kaggle.com/datasets/jayaantanaath/student-habits-vs-academic-performance

Nguyen, Q., Huptych, M., & Rienties, B. (2018). Linking students' timing of engagement to learning design and academic performance. *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, 141-150. https://doi.org/10.1145/3170358.3170398

Stull, J. C. (2013). Family socioeconomic status, parent expectations, and a child’s achievement. *Research in Education, 90*(1). https://doi.org/10.7227/RIE.90.1.4

Tetteh, G. A. (2018). Effects of classroom attendance and learning strategies on the learning outcome. *Journal of the International Education in Business, 11*(2), 195-219. https://doi.org/10.1108/JIEB-01-2017-0004

Appendix:

Python Notebook

variable_analysis

October 19, 2025

1 Comprehensive Variable Analysis

1.1 Student Habits vs Academic Performance Dataset

1.1.1 Topics Explored:

- Descriptive statistics: mean, median, mode, standard deviation, min/max
- Distribution assessment: histograms, normality tests (Shapiro-Wilk), skewness, kurtosis
- Outlier detection: boxplots, z-scores (±3 SD)
- Missing value analysis
- Categorical variable frequency analysis
- Correlation analysis for relationships between variables

[60]:
```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
from scipy.stats import shapiro, normaltest, skew, kurtosis
import warnings
warnings.filterwarnings('ignore')

# Set visualization style
plt.style.use('seaborn-v0_8-darkgrid')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10
```

1.2 1. Data Loading and Initial Inspection

```
[ ]: # Load the dataset from local directory
     # Note: keep_default_na=False prevents pandas from treating "None" string as␣
      ↪missing
     # We explicitly define what should be treated as NA via na_values parameter
     df = pd.read_csv('/Users/jje/src/Stats500FinalProject/data/
      ↪student_habits_performance.csv', keep_default_na=False)

     print("Dataset Shape:", df.shape)
     df.head()
```

```
Dataset Shape: (1000, 16)

First 5 rows:
```

```
[ ]:   student_id  age  gender  study_hours_per_day  social_media_hours  \
     0      S1000   23  Female                  0.0                 1.2
     1      S1001   20  Female                  6.9                 2.8
     2      S1002   21    Male                  1.4                 3.1
     3      S1003   23  Female                  1.0                 3.9
     4      S1004   19  Female                  5.0                 4.4

        netflix_hours part_time_job  attendance_percentage  sleep_hours  \
     0            1.1            No                   85.0          8.0
     1            2.3            No                   97.3          4.6
     2            1.3            No                   94.8          8.0
     3            1.0            No                   71.0          9.2
     4            0.5            No                   90.9          4.9

       diet_quality  exercise_frequency parental_education_level internet_quality  \
     0         Fair                   6                   Master          Average
     1         Good                   6              High School          Average
     2         Poor                   1              High School             Poor
     3         Poor                   4                   Master             Good
     4         Fair                   3                   Master             Good

        mental_health_rating extracurricular_participation  exam_score
     0                     8                           Yes        56.2
     1                     8                            No       100.0
     2                     1                            No        34.3
     3                     1                           Yes        26.8
     4                     1                            No        66.4
```

```
[79]: # Display data types and basic info
      print("Dataset Information:")
      df.info()
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   student_id                     1000 non-null   object 
 1   age                            1000 non-null   int64  
 2   gender                         1000 non-null   object 
 3   study_hours_per_day            1000 non-null   float64
 4   social_media_hours             1000 non-null   float64
 5   netflix_hours                  1000 non-null   float64
 6   part_time_job                  1000 non-null   object 
 7   attendance_percentage          1000 non-null   float64
 8   sleep_hours                    1000 non-null   float64
 9   diet_quality                   1000 non-null   object 
 10  exercise_frequency             1000 non-null   int64  
 11  parental_education_level       1000 non-null   object 
 12  internet_quality               1000 non-null   object 
 13  mental_health_rating           1000 non-null   int64  
 14  extracurricular_participation  1000 non-null   object 
 15  exam_score                     1000 non-null   float64
dtypes: float64(6), int64(3), object(7)
memory usage: 125.1+ KB
```

1.3 2. Missing Value Analysis

[80]:
```
# Missing value analysis
missingData = pd.DataFrame({
    'missingCount': df.isnull().sum(),
    'missingPercentage': (df.isnull().sum() / len(df)) * 100
})
missingData = missingData[missingData['missingCount'] > 0].
 ↪sort_values('missingCount', ascending=False)

if len(missingData) > 0:
    print("Missing Values Summary:")
    print(missingData)
else:
    print("No missing values detected in the dataset.")
```

```
No missing values detected in the dataset.
```

1.4 3. Variable Classification

[81]:
```
# Classify variables by type
numericVars = df.select_dtypes(include=[np.number]).columns.tolist()
categoricalVars = df.select_dtypes(include=['object']).columns.tolist()

# Remove ID variable from analysis
```

```
if 'student_id' in numericVars:
    numericVars.remove('student_id')
if 'student_id' in categoricalVars:
    categoricalVars.remove('student_id')

print("Variable Classification:")
print(f"\nNumeric Variables ({len(numericVars)}):")
for var in numericVars:
    print(f"  - {var}")

print(f"\nCategorical Variables ({len(categoricalVars)}):")
for var in categoricalVars:
    print(f"  - {var}")
```

```
Variable Classification:

Numeric Variables (9):
  - age
  - study_hours_per_day
  - social_media_hours
  - netflix_hours
  - attendance_percentage
  - sleep_hours
  - exercise_frequency
  - mental_health_rating
  - exam_score

Categorical Variables (6):
  - gender
  - part_time_job
  - diet_quality
  - parental_education_level
  - internet_quality
  - extracurricular_participation
```

1.5 4. Descriptive Statistics for Numeric Variables

The following values for the dataset will be explored: - Central tendency: Mean, Median, Mode - Dispersion: Standard Deviation, Variance, Range, IQR - Distribution shape: Skewness, Kurtosis - Extreme values: Min, Max, Quartiles

[82]:
```
# Comprehensive descriptive statistics
descStats = df[numericVars].describe().T

# Add additional statistics
descStats['variance'] = df[numericVars].var()
descStats['skewness'] = df[numericVars].skew()
descStats['range'] = descStats['max'] - descStats['min']
```

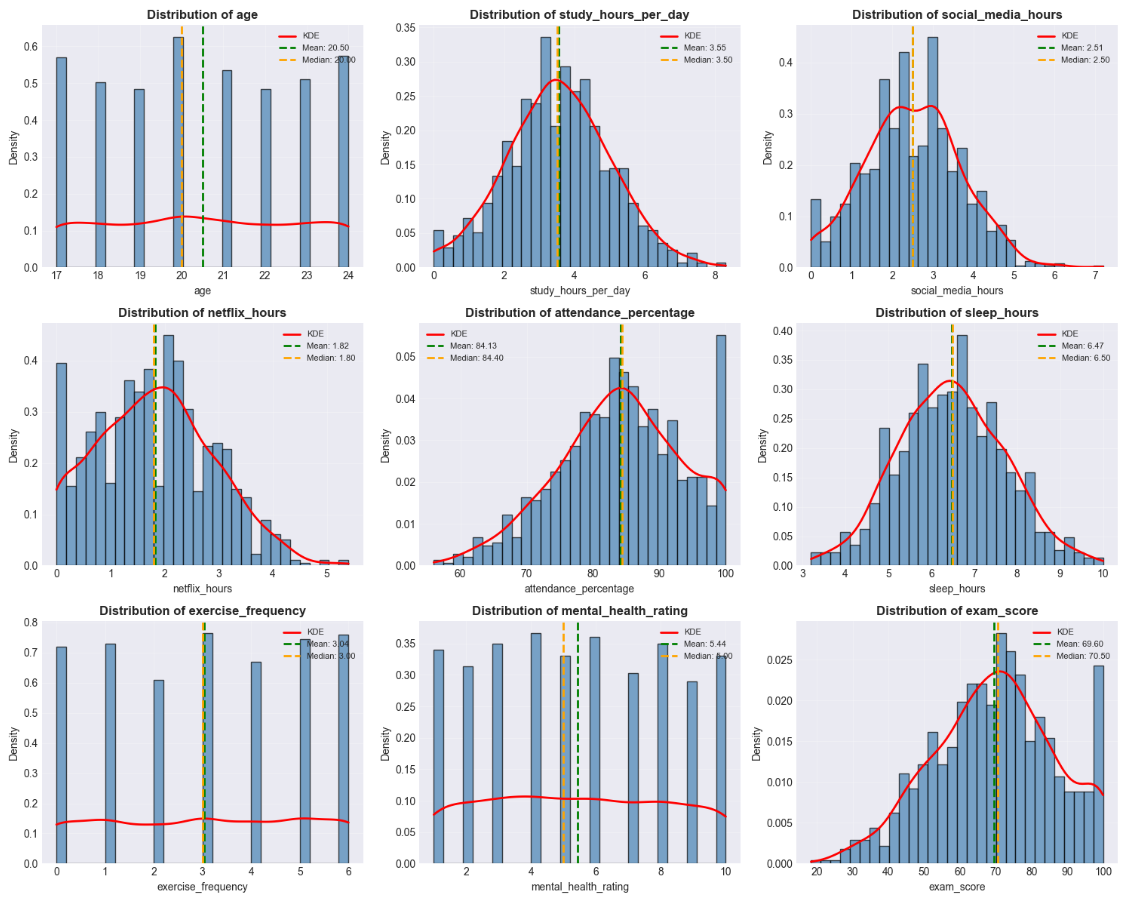
```
descStats['IQR'] = descStats['75%'] - descStats['25%']

# Reorder columns for better readability
columnOrder = ['count', 'mean', '50%', 'std', 'variance', 'min', '25%', '75%',␣
 ↪'max',
                'range', 'IQR', 'skewness']
descStats = descStats[columnOrder]
descStats.columns = ['Count', 'Mean', 'Median', 'Std Dev', 'Variance', 'Min',␣
 ↪'Q1',
                       'Q3', 'Max', 'Range', 'IQR', 'Skewness']

print("Comprehensive Descriptive Statistics:")
print(descStats.round(3))
```

```
Comprehensive Descriptive Statistics:
                        Count    Mean  Median  Std Dev  Variance   Min  \
age                    1000.0  20.498    20.0    2.308     5.327  17.0
study_hours_per_day    1000.0   3.550     3.5    1.469     2.158   0.0
social_media_hours     1000.0   2.506     2.5    1.172     1.375   0.0
netflix_hours          1000.0   1.820     1.8    1.075     1.156   0.0
attendance_percentage  1000.0  84.132    84.4    9.399    88.346  56.0
sleep_hours            1000.0   6.470     6.5    1.226     1.504   3.2
exercise_frequency     1000.0   3.042     3.0    2.025     4.102   0.0
mental_health_rating   1000.0   5.438     5.0    2.848     8.108   1.0
exam_score             1000.0  69.602    70.5   16.889   285.224  18.4

                           Q1      Q3    Max  Range     IQR  Skewness
age                    18.750  23.000   24.0    7.0   4.250     0.008
study_hours_per_day     2.600   4.500    8.3    8.3   1.900     0.054
social_media_hours      1.700   3.300    7.2    7.2   1.600     0.120
netflix_hours           1.000   2.525    5.4    5.4   1.525     0.237
attendance_percentage  78.000  91.025  100.0   44.0  13.025    -0.238
sleep_hours             5.600   7.300   10.0    6.8   1.700     0.091
exercise_frequency      1.000   5.000    6.0    6.0   4.000    -0.032
mental_health_rating    3.000   8.000   10.0    9.0   5.000     0.038
exam_score             58.475  81.325  100.0   81.6  22.850    -0.156
```

1.6 5. Distribution Analysis for Numeric Variables

[83]:
```
# Create histograms for all numeric variables
nVars = len(numericVars)
nCols = 3
nRows = (nVars + nCols - 1) // nCols

fig, axes = plt.subplots(nRows, nCols, figsize=(15, nRows * 4))
axes = axes.flatten() if nVars > 1 else [axes]
```

```python
for idx, var in enumerate(numericVars):
    ax = axes[idx]

    # Plot histogram with KDE
    data = df[var].dropna()
    ax.hist(data, bins=30, alpha=0.7, color='steelblue', edgecolor='black',␣
 ↪density=True)

    # Add KDE curve
    from scipy.stats import gaussian_kde
    kde = gaussian_kde(data)
    xRange = np.linspace(data.min(), data.max(), 100)
    ax.plot(xRange, kde(xRange), 'r-', linewidth=2, label='KDE')

    # Add mean and median lines
    meanVal = data.mean()
    medianVal = data.median()
    ax.axvline(meanVal, color='green', linestyle='--', linewidth=2,␣
 ↪label=f'Mean: {meanVal:.2f}')
    ax.axvline(medianVal, color='orange', linestyle='--', linewidth=2,␣
 ↪label=f'Median: {medianVal:.2f}')

    ax.set_title(f'Distribution of {var}', fontsize=12, fontweight='bold')
    ax.set_xlabel(var)
    ax.set_ylabel('Density')
    ax.legend(loc='best', fontsize=8)
    ax.grid(True, alpha=0.3)

# Remove extra subplots
for idx in range(nVars, len(axes)):
    fig.delaxes(axes[idx])

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



1.7 6. Normality Tests

Using Shapiro-Wilk test to assess normality: - H0: Data is normally distributed - H1: Data is not normally distributed - Significance level: = 0.05

[85]:
```
# Perform normality tests
normalityResults = []

for var in numericVars:
    data = df[var].dropna()

    # Shapiro-Wilk test
    stat, pValue = shapiro(data)

    # Interpretation
    isNormal = "Yes" if pValue > 0.05 else "No"

    normalityResults.append({
        'Variable': var,
        'Statistic': stat,
        'pValue': pValue,
        'Normal ( =0.05)': isNormal
    })

normalityDf = pd.DataFrame(normalityResults)

print("Shapiro-Wilk Normality Test Results:")
print(normalityDf.to_string(index=False))
```

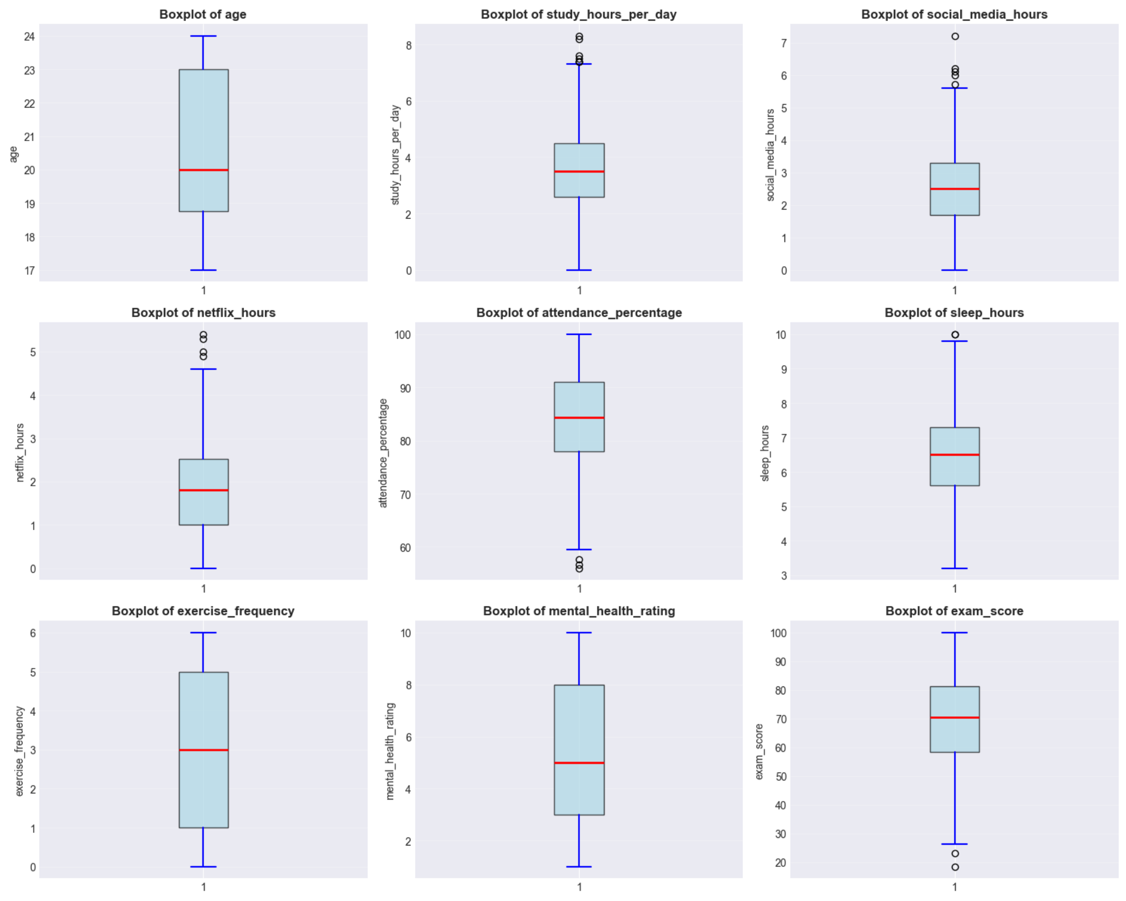
```
Shapiro-Wilk Normality Test Results:
             Variable  Statistic       pValue Normal ( =0.05)
                  age   0.924860 6.177718e-22              No
    study_hours_per_day   0.997378 1.064707e-01             Yes
    social_media_hours   0.994266 7.275089e-04              No
        netflix_hours   0.982686 1.608654e-09              No
attendance_percentage   0.982607 1.502940e-09              No
          sleep_hours   0.997267 8.877598e-02             Yes
   exercise_frequency   0.913922 2.263751e-23              No
  mental_health_rating   0.938175 5.841297e-20              No
           exam_score   0.986919 8.675028e-08              No
```

1.8 7. Outlier Detection

Using boxplots and z-score method (values beyond ±3 SD).

[86]:
```
# Create boxplots for all numeric variables
nVars = len(numericVars)
nCols = 3
nRows = (nVars + nCols - 1) // nCols

fig, axes = plt.subplots(nRows, nCols, figsize=(15, nRows * 4))
axes = axes.flatten() if nVars > 1 else [axes]

for idx, var in enumerate(numericVars):
    ax = axes[idx]

    # Create boxplot
    bp = ax.boxplot(df[var].dropna(), vert=True, patch_artist=True,
                    boxprops=dict(facecolor='lightblue', alpha=0.7),
                    medianprops=dict(color='red', linewidth=2),
                    whiskerprops=dict(color='blue', linewidth=1.5),
                    capprops=dict(color='blue', linewidth=1.5))

    ax.set_title(f'Boxplot of {var}', fontsize=12, fontweight='bold')
    ax.set_ylabel(var)
    ax.grid(True, alpha=0.3, axis='y')
```

```
# Remove extra subplots
for idx in range(nVars, len(axes)):
    fig.delaxes(axes[idx])

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



[87]:
```
# Outlier detection using z-score method (±3 SD)
outlierSummary = []

for var in numericVars:
    data = df[var].dropna()
    zScores = np.abs((data - data.mean()) / data.std())
    outliers = data[zScores > 3]

    outlierSummary.append({
        'Variable': var,
        'totalObservations': len(data),
        'outliersCount': len(outliers),
        'outliersPercentage': (len(outliers) / len(data)) * 100
    })

outlierDf = pd.DataFrame(outlierSummary)

print("Outlier Detection Summary (Z-score > 3):")
print(outlierDf.to_string(index=False))
print("\nNote: Z-scores beyond ±3 SD are considered potential outliers.")
```

```
Outlier Detection Summary (Z-score > 3):
             Variable  totalObservations  outliersCount  outliersPercentage
                  age               1000              0                 0.0
    study_hours_per_day               1000              2                 0.2
   social_media_hours               1000              3                 0.3
        netflix_hours               1000              2                 0.2
attendance_percentage               1000              0                 0.0
          sleep_hours               1000              0                 0.0
   exercise_frequency               1000              0                 0.0
  mental_health_rating               1000              0                 0.0
           exam_score               1000              1                 0.1

Note: Z-scores beyond ±3 SD are considered potential outliers.
```

1.9 8. Categorical Variables Analysis

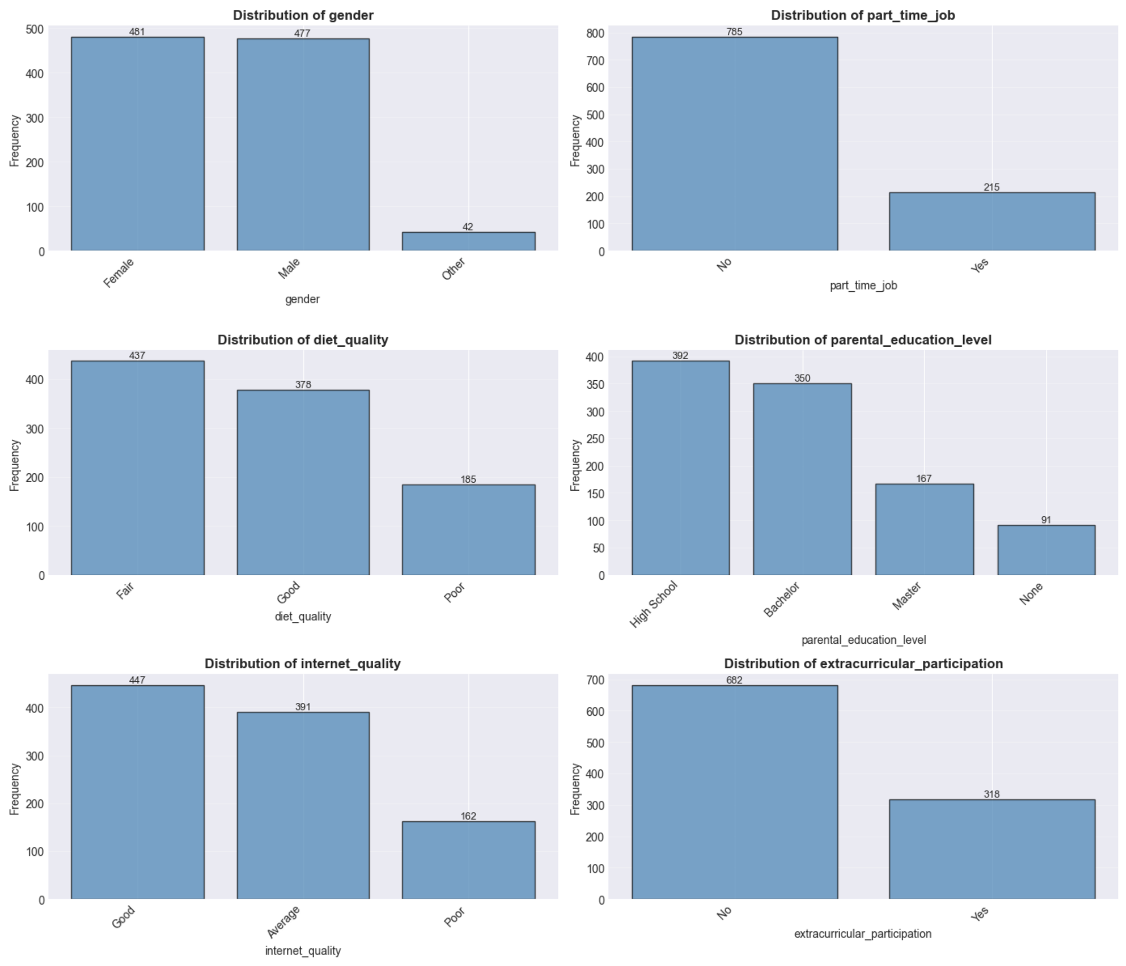
Frequency counts and percentages for categorical variables.

[88]:
```
# Frequency analysis for categorical variables
for var in categoricalVars:
    print(f"\nFrequency Distribution: {var}")

    freqTable = df[var].value_counts()
    pctTable = df[var].value_counts(normalize=True) * 100

    result = pd.DataFrame({
        'Count': freqTable,
        'Percentage': pctTable
    })

    print(result.round(2))
    print(f"Total: {freqTable.sum()}")
```

```

Frequency Distribution: gender
        Count  Percentage
gender
Female    481        48.1
Male      477        47.7
Other      42         4.2
```

```
Total: 1000

Frequency Distribution: part_time_job
               Count  Percentage
part_time_job
No               785        78.5
Yes              215        21.5
Total: 1000

Frequency Distribution: diet_quality
              Count  Percentage
diet_quality
Fair            437        43.7
Good            378        37.8
Poor            185        18.5
Total: 1000

Frequency Distribution: parental_education_level
                          Count  Percentage
parental_education_level
High School                 392        39.2
Bachelor                    350        35.0
Master                      167        16.7
None                         91         9.1
Total: 1000

Frequency Distribution: internet_quality
                  Count  Percentage
internet_quality
Good                447        44.7
Average             391        39.1
Poor                162        16.2
Total: 1000

Frequency Distribution: extracurricular_participation
                               Count  Percentage
extracurricular_participation
No                               682        68.2
Yes                              318        31.8
Total: 1000
```

[89]:
```python
# Visualize categorical variables
nVars = len(categoricalVars)
if nVars > 0:
    nCols = 2
    nRows = (nVars + nCols - 1) // nCols
    fig, axes = plt.subplots(nRows, nCols, figsize=(14, nRows * 4))
    axes = axes.flatten() if nVars > 1 else [axes]

    for idx, var in enumerate(categoricalVars):
        ax = axes[idx]

        # Count values
        valueCounts = df[var].value_counts()

        # Create bar plot
        bars = ax.bar(range(len(valueCounts)), valueCounts.values,
                      color='steelblue', alpha=0.7, edgecolor='black')
        ax.set_xticks(range(len(valueCounts)))
        ax.set_xticklabels(valueCounts.index, rotation=45, ha='right')
        ax.set_title(f'Distribution of {var}', fontsize=12, fontweight='bold')
        ax.set_xlabel(var)
        ax.set_ylabel('Frequency')
        ax.grid(True, alpha=0.3, axis='y')

        # Add count labels on bars
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                   f'{int(height)}',
                   ha='center', va='bottom', fontsize=9)

    # Remove extra subplots
    for idx in range(nVars, len(axes)):
        fig.delaxes(axes[idx])

    plt.tight_layout()
    plt.show()
else:
    print("No categorical variables to visualize.")
```



1.10 9. Correlation Analysis

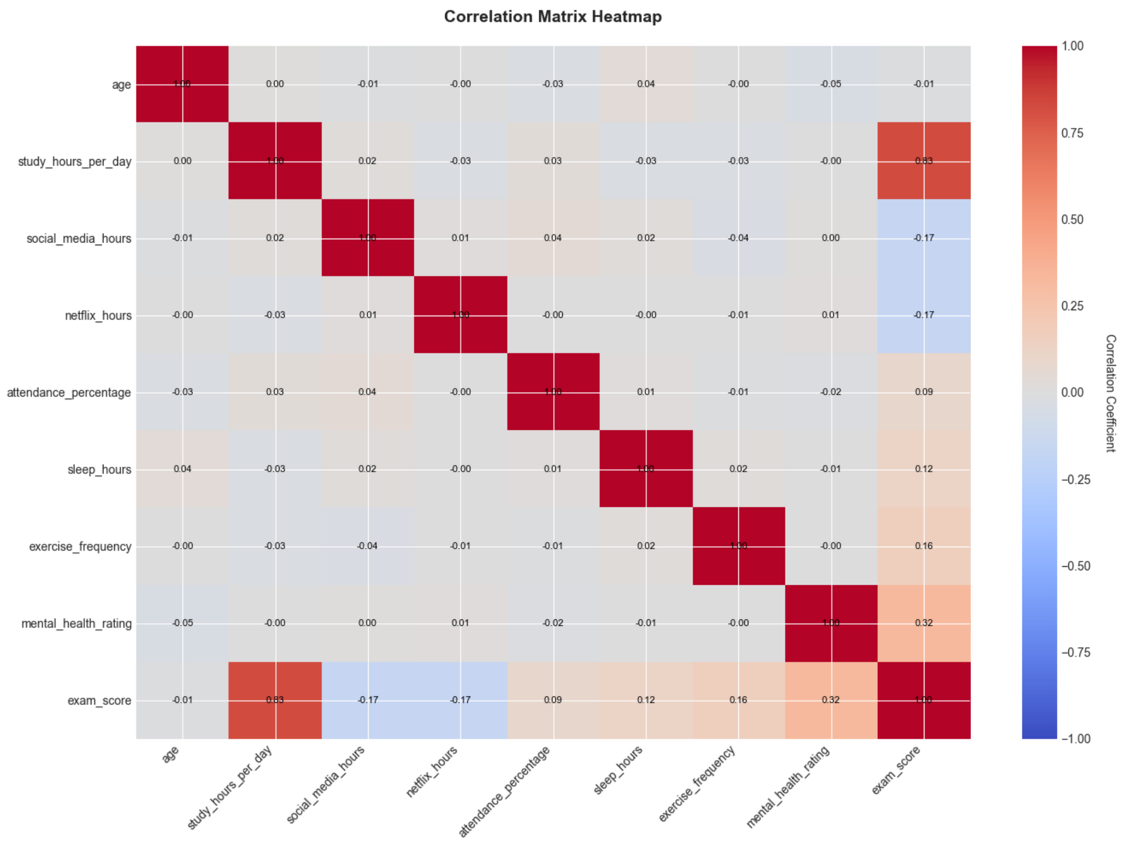
Examining relationships between numeric variables using Pearson correlation.

[90]:
```
# Calculate correlation matrix
correlationMatrix = df[numericVars].corr()

# Display correlation matrix
print("Correlation Matrix:")
print(correlationMatrix.round(3))
```

```
Correlation Matrix:
                        age  study_hours_per_day  social_media_hours  \
age                   1.000                0.004              -0.009
study_hours_per_day   0.004                1.000               0.020
social_media_hours   -0.009                0.020               1.000
netflix_hours        -0.001               -0.031               0.011
attendance_percentage -0.026               0.026               0.040
```

```
sleep_hours                0.037                -0.028                0.018
exercise_frequency        -0.004                -0.029               -0.037
mental_health_rating      -0.045                -0.004                0.001
exam_score                -0.009                 0.825               -0.167

                       netflix_hours  attendance_percentage  sleep_hours  \
age                           -0.001                 -0.026        0.037
study_hours_per_day           -0.031                  0.026       -0.028
social_media_hours             0.011                  0.040        0.018
netflix_hours                  1.000                 -0.002       -0.001
attendance_percentage         -0.002                  1.000        0.014
sleep_hours                   -0.001                  0.014        1.000
exercise_frequency            -0.006                 -0.008        0.020
mental_health_rating           0.008                 -0.019       -0.007
exam_score                    -0.172                  0.090        0.122

                       exercise_frequency  mental_health_rating  exam_score
age                                -0.004                -0.045      -0.009
study_hours_per_day                -0.029                -0.004       0.825
social_media_hours                 -0.037                 0.001      -0.167
netflix_hours                      -0.006                 0.008      -0.172
attendance_percentage              -0.008                -0.019       0.090
sleep_hours                         0.020                -0.007       0.122
exercise_frequency                  1.000                -0.000       0.160
mental_health_rating               -0.000                 1.000       0.322
exam_score                          0.160                 0.322       1.000
```

[91]:
```python
# Visualize correlation matrix as heatmap
fig, ax = plt.subplots(figsize=(14, 10))

# Create heatmap
im = ax.imshow(correlationMatrix, cmap='coolwarm', aspect='auto', vmin=-1,␣
 ↪vmax=1)

# Set ticks and labels
ax.set_xticks(range(len(correlationMatrix.columns)))
ax.set_yticks(range(len(correlationMatrix.columns)))
ax.set_xticklabels(correlationMatrix.columns, rotation=45, ha='right')
ax.set_yticklabels(correlationMatrix.columns)

# Add colorbar
cbar = plt.colorbar(im, ax=ax)
cbar.set_label('Correlation Coefficient', rotation=270, labelpad=20)

# Add correlation values as text
for i in range(len(correlationMatrix)):
    for j in range(len(correlationMatrix)):
        text = ax.text(j, i, f'{correlationMatrix.iloc[i, j]:.2f}',
                      ha='center', va='center', color='black', fontsize=8)

ax.set_title('Correlation Matrix Heatmap', fontsize=14, fontweight='bold',␣
 ↪pad=20)
plt.tight_layout()
plt.show()
```



[92]:
```
# Find strongest correlations (excluding diagonal)
# Get upper triangle of correlation matrix
corrPairs = []
for i in range(len(correlationMatrix.columns)):
    for j in range(i+1, len(correlationMatrix.columns)):
        corrPairs.append({
            'variable1': correlationMatrix.columns[i],
            'variable2': correlationMatrix.columns[j],
            'correlation': correlationMatrix.iloc[i, j]
        })

corrDf = pd.DataFrame(corrPairs)
```

```
corrDf['absCorrelation'] = corrDf['correlation'].abs()
corrDf = corrDf.sort_values('absCorrelation', ascending=False)

print("\nTop 10 Strongest Correlations:")
print(corrDf.head(10)[['variable1', 'variable2', 'correlation']].
 ↪to_string(index=False))
```

```
Top 10 Strongest Correlations:
            variable1             variable2  correlation
  study_hours_per_day            exam_score     0.825419
 mental_health_rating            exam_score     0.321523
        netflix_hours            exam_score    -0.171779
   social_media_hours            exam_score    -0.166733
   exercise_frequency            exam_score     0.160107
          sleep_hours            exam_score     0.121683
attendance_percentage            exam_score     0.089836
                  age  mental_health_rating    -0.045101
   social_media_hours attendance_percentage     0.040479
                  age           sleep_hours     0.037482
```

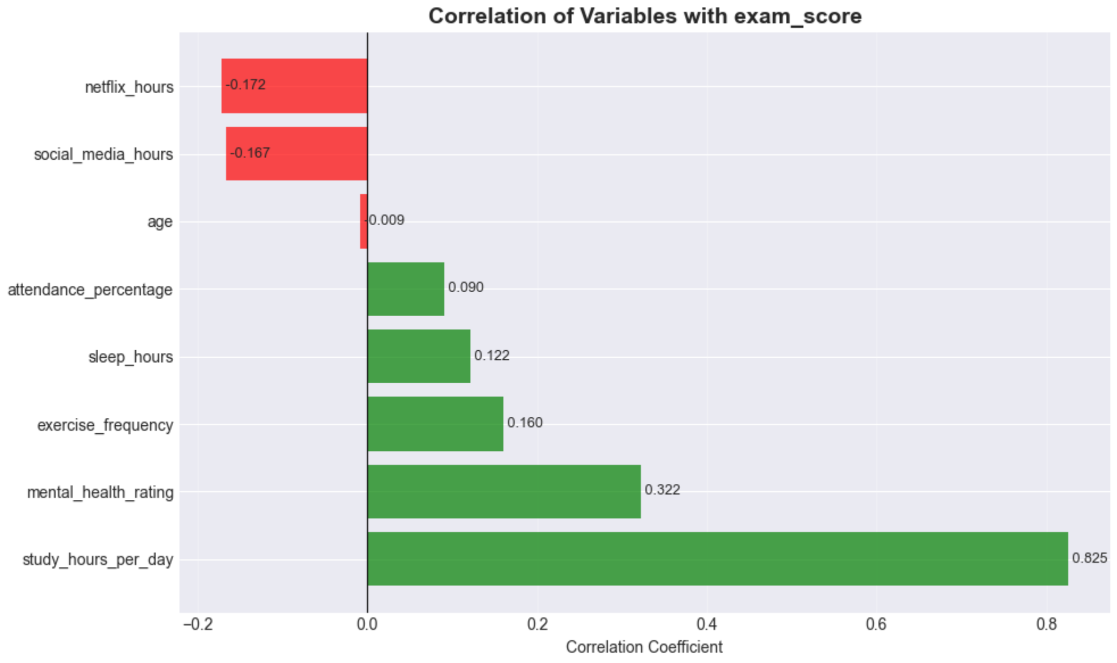
1.11 10. Variable Relationships with Target (exam_score)

Analyzing how each variable relates to the outcome variable.

[93]:
```
# Identify target variable
targetVar = 'exam_score'

if targetVar in numericVars:
    # Get correlations with target
    targetCorr = correlationMatrix[targetVar].drop(targetVar).
 ↪sort_values(ascending=False)

    print(f"Correlations with {targetVar}:")
    for var, corr in targetCorr.items():
        print(f"{var:30s}: {corr:7.3f}")

    # Visualize correlations with target
    fig, ax = plt.subplots(figsize=(10, 6))
    colors = ['green' if x > 0 else 'red' for x in targetCorr.values]
    bars = ax.barh(range(len(targetCorr)), targetCorr.values, color=colors,␣
 ↪alpha=0.7)
    ax.set_yticks(range(len(targetCorr)))
    ax.set_yticklabels(targetCorr.index)
    ax.axvline(x=0, color='black', linewidth=0.8)
    ax.set_xlabel('Correlation Coefficient')
    ax.set_title(f'Correlation of Variables with {targetVar}', fontsize=14,␣
 ↪fontweight='bold')
```

```
    ax.grid(True, alpha=0.3, axis='x')

    # Add value labels
    for i, (var, corr) in enumerate(targetCorr.items()):
        ax.text(corr, i, f' {corr:.3f}', va='center', fontsize=9)

    plt.tight_layout()
    plt.show()
else:
    print(f"Target variable '{targetVar}' not found in numeric variables.")
```

```
Correlations with exam_score:
study_hours_per_day            :   0.825
mental_health_rating           :   0.322
exercise_frequency             :   0.160
sleep_hours                    :   0.122
attendance_percentage          :   0.090
age                            :  -0.009
social_media_hours             :  -0.167
netflix_hours                  :  -0.172
```



variable_comparison_analysis

October 19, 2025

1 Group Comparison Analysis: Study Habits and Exam Performance

1.1 T-tests, ANOVA, and Chi-Squared Tests

This notebook explores exam score differences across groups using: - **T-tests**: Comparing exam scores between two groups - **ANOVA**: Comparing exam scores across three or more groups - **Chi-squared tests**: Examining associations between categorical variables

1.1.1 Research Questions:

1. Do students with high study hours score higher on exams than students with low study hours?
2. Do students with good mental health score higher on exams than students with poor mental health?
3. Does exam performance differ across multiple study hour categories?
4. Does exam performance differ across multiple mental health categories?
5. Does exam performance differ across exercise frequency levels?
6. Does exam performance differ across netflix viewing time categories?
7. Does exam performance differ across diet quality levels?
8. Is exam performance associated with parental education level?
9. Is exam performance associated with diet quality?

1.1.2 Statistical Methods:

- Independent samples t-test
- One-way ANOVA
- Post-hoc tests (Tukey HSD)
- Effect size measures (Cohen's d, eta-squared)
- Chi-squared test of independence
- Cramér's V effect size

[387]:
```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import ttest_ind, f_oneway, shapiro, levene
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import warnings
```

```
warnings.filterwarnings('ignore')

# Set visualization style
plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11
```

1.2 1. Data Loading and Preparation

[388]:
```
# Load the dataset
df = pd.read_csv('/Users/jje/src/Stats500FinalProject/data/
 ↪student_habits_performance.csv', keep_default_na=False)

print(f"Dataset shape: {df.shape}")
print(f"Variables of interest:")
print(df[['study_hours_per_day', 'mental_health_rating', 'exam_score']].
 ↪describe())
```

```
Dataset shape: (1000, 16)
Variables of interest:
       study_hours_per_day  mental_health_rating   exam_score
count           1000.00000           1000.000000  1000.000000
mean               3.55010              5.438000    69.601500
std                1.46889              2.847501    16.888564
min                0.00000              1.000000    18.400000
25%                2.60000              3.000000    58.475000
50%                3.50000              5.000000    70.500000
75%                4.50000              8.000000    81.325000
max                8.30000             10.000000   100.000000
```

1.3 2. Creating Group Categories

We'll create categorical groups based on continuous variables for group comparisons.

[389]:
```
# Create study hours groups (2 groups for t-test)
studyMedian = df['study_hours_per_day'].median()
df['studyGroup2'] = df['study_hours_per_day'].apply(
    lambda x: 'Low Study Hours' if x < studyMedian else 'High Study Hours'
)

# Create study hours groups (3 groups for ANOVA)
studyQ1 = df['study_hours_per_day'].quantile(0.33)
studyQ2 = df['study_hours_per_day'].quantile(0.67)
df['studyGroup3'] = df['study_hours_per_day'].apply(
    lambda x: 'Low' if x < studyQ1 else ('Medium' if x < studyQ2 else 'High')
)

# Create mental health groups (2 groups for t-test)
```

```
mentalMedian = df['mental_health_rating'].median()
df['mentalGroup2'] = df['mental_health_rating'].apply(
    lambda x: 'Lower Mental Health' if x < mentalMedian else 'Higher Mental␣
  ↪Health'
)

# Create mental health groups (3 groups for ANOVA)
mentalQ1 = df['mental_health_rating'].quantile(0.33)
mentalQ2 = df['mental_health_rating'].quantile(0.67)
df['mentalGroup3'] = df['mental_health_rating'].apply(
    lambda x: 'Low' if x < mentalQ1 else ('Medium' if x < mentalQ2 else 'High')
)

# Create exercise frequency groups (3 groups for ANOVA)
exerciseQ1 = df['exercise_frequency'].quantile(0.33)
exerciseQ2 = df['exercise_frequency'].quantile(0.67)
df['exerciseGroup3'] = df['exercise_frequency'].apply(
    lambda x: 'Low' if x < exerciseQ1 else ('Medium' if x < exerciseQ2 else␣
  ↪'High')
)

# Create netflix hours groups (3 groups for ANOVA)
netflixQ1 = df['netflix_hours'].quantile(0.33)
netflixQ2 = df['netflix_hours'].quantile(0.67)
df['netflixGroup3'] = df['netflix_hours'].apply(
    lambda x: 'Low' if x < netflixQ1 else ('Medium' if x < netflixQ2 else␣
  ↪'High')
)

# diet_quality is already categorical, so no grouping needed

print(f"Group Distribution - Study Hours (2 groups): {df['studyGroup2'].
  ↪value_counts()}")
print(f"Group Distribution - Study Hours (3 groups): {df['studyGroup3'].
  ↪value_counts()}")
print(f"Group Distribution - Mental Health (2 groups): {df['mentalGroup2'].
  ↪value_counts()}")
print(f"Group Distribution - Mental Health (3 groups): {df['mentalGroup3'].
  ↪value_counts()}")
print(f"Group Distribution - Exercise Frequency (3 groups):␣
  ↪{df['exerciseGroup3'].value_counts()}")
print(f"Group Distribution - Netflix Hours (3 groups): {df['netflixGroup3'].
  ↪value_counts()}")
print(f"Group Distribution - Diet Quality (categorical): {df['diet_quality'].
  ↪value_counts()}")
```

```
Group Distribution - Study Hours (2 groups): studyGroup2
High Study Hours    529
Low Study Hours     471
Name: count, dtype: int64
Group Distribution - Study Hours (3 groups): studyGroup3
Medium    347
High      339
Low       314
Name: count, dtype: int64
Group Distribution - Mental Health (2 groups): mentalGroup2
Higher Mental Health    589
Lower Mental Health     411
Name: count, dtype: int64
Group Distribution - Mental Health (3 groups): mentalGroup3
High      382
Medium    317
Low       301
Name: count, dtype: int64
Group Distribution - Exercise Frequency (3 groups): exerciseGroup3
High      435
Low       290
Medium    275
Name: count, dtype: int64
Group Distribution - Netflix Hours (3 groups): netflixGroup3
High      342
Medium    339
Low       319
Name: count, dtype: int64
Group Distribution - Diet Quality (categorical): diet_quality
Fair    437
Good    378
Poor    185
Name: count, dtype: int64
```

1.4 3. T-Test Analysis (2 Groups)

1.4.1 3.1 Study Hours Groups: Independent Samples T-Test

Hypotheses: - H : Mean exam scores are equal between low and high study hour groups - H : Mean exam scores differ between low and high study hour groups

[390]:
```python
# Separate groups
lowStudyScores = df[df['studyGroup2'] == 'Low Study Hours']['exam_score']
highStudyScores = df[df['studyGroup2'] == 'High Study Hours']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Study Hours Group:")
print(f"Low Study Hours (n={len(lowStudyScores)}):")
print(f"Mean: {lowStudyScores.mean():.2f}")
print(f"SD: {lowStudyScores.std():.2f}")
print(f"Median: {lowStudyScores.median():.2f}")

print(f"High Study Hours (n={len(highStudyScores)}):")
print(f"Mean: {highStudyScores.mean():.2f}")
print(f"SD: {highStudyScores.std():.2f}")
print(f"Median: {highStudyScores.median():.2f}")

print(f"Mean Difference: {highStudyScores.mean() - lowStudyScores.mean():.2f}␣
↪points")
```

```
Descriptive Statistics by Study Hours Group:
Low Study Hours (n=471):
Mean: 57.81
SD: 13.28
Median: 58.40
High Study Hours (n=529):
Mean: 80.10
SD: 12.17
Median: 79.60
Mean Difference: 22.30 points
```

[391]:
```
# 1. Normality test
statLow, pLow = shapiro(lowStudyScores)
statHigh, pHigh = shapiro(highStudyScores)

print("1. Normality (Shapiro-Wilk test):")
print(f"Low Study Hours: p = {pLow:.4f} {'(Normal)' if pLow > 0.05 else '(Not␣
↪Normal)'}")
print(f"High Study Hours: p = {pHigh:.4f} {'(Normal)' if pHigh > 0.05 else␣
↪'(Not Normal)'}")

# 2. Homogeneity of variance
statLevene, pLevene = levene(lowStudyScores, highStudyScores)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
↪variances)'}")
```

```
1. Normality (Shapiro-Wilk test):
Low Study Hours: p = 0.1422 (Normal)
High Study Hours: p = 0.0000 (Not Normal)
2. Homogeneity of Variance (Levene's test):
p = 0.1378 (Equal variances)
```

[392]:
```
# Perform t-test
tStat, pValue = ttest_ind(highStudyScores, lowStudyScores, equal_var=(pLevene >␣
↪0.05))
```

```
print("Independent Samples T-Test Results:")
print(f"t-statistic: {tStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
  ↪at   = 0.05")

# Calculate Cohen's d (effect size)
pooledStd = np.sqrt(((len(lowStudyScores)-1)*lowStudyScores.std()**2 +
                      (len(highStudyScores)-1)*highStudyScores.std()**2) /
                     (len(lowStudyScores) + len(highStudyScores) - 2))
cohensD = (highStudyScores.mean() - lowStudyScores.mean()) / pooledStd

print(f"Effect Size (Cohen's d): {cohensD:.4f}")
if abs(cohensD) < 0.2:
    effect = "negligible"
elif abs(cohensD) < 0.5:
    effect = "small"
elif abs(cohensD) < 0.8:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
Independent Samples T-Test Results:
t-statistic: 27.7087
p-value: 8.4759e-126
Significance: Significant at   = 0.05
Effect Size (Cohen's d): 1.7554
Interpretation: Large effect
```

1.4.2 3.2 Mental Health Groups: Independent Samples T-Test

Hypotheses: - H : Mean exam scores are equal between lower and higher mental health groups - H : Mean exam scores differ between lower and higher mental health groups

[393]:
```
# Separate groups
lowerMentalScores = df[df['mentalGroup2'] == 'Lower Mental␣
  ↪Health']['exam_score']
higherMentalScores = df[df['mentalGroup2'] == 'Higher Mental␣
  ↪Health']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Mental Health Group:")
print(f"Lower Mental Health (n={len(lowerMentalScores)}):")
print(f"Mean: {lowerMentalScores.mean():.2f}")
print(f"SD: {lowerMentalScores.std():.2f}")
print(f"Median: {lowerMentalScores.median():.2f}")
print(f"Higher Mental Health (n={len(higherMentalScores)}):")
print(f"Mean: {higherMentalScores.mean():.2f}")
print(f"SD: {higherMentalScores.std():.2f}")
print(f"Median: {higherMentalScores.median():.2f}")

print(f"Mean Difference: {higherMentalScores.mean() - lowerMentalScores.mean():.
 ↪2f} points")
```

```
Descriptive Statistics by Mental Health Group:
Lower Mental Health (n=411):
Mean: 64.03
SD: 16.39
Median: 64.20
Higher Mental Health (n=589):
Mean: 73.49
SD: 16.14
Median: 74.00
Mean Difference: 9.45 points
```

[394]:
```
# Check assumptions

# 1. Normality test
statLow, pLow = shapiro(lowerMentalScores)
statHigh, pHigh = shapiro(higherMentalScores)

print("1. Normality (Shapiro-Wilk test):")
print(f"Lower Mental Health: p = {pLow:.4f} {'(Normal)' if pLow > 0.05 else␣
 ↪'(Not Normal)'}")
print(f"Higher Mental Health: p = {pHigh:.4f} {'(Normal)' if pHigh > 0.05 else␣
 ↪'(Not Normal)'}")

# 2. Homogeneity of variance
statLevene, pLevene = levene(lowerMentalScores, higherMentalScores)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
 ↪variances)'}")
```

```
1. Normality (Shapiro-Wilk test):
Lower Mental Health: p = 0.0834 (Normal)
Higher Mental Health: p = 0.0000 (Not Normal)
2. Homogeneity of Variance (Levene's test):
p = 0.9070 (Equal variances)
```

[395]:
```
# Perform t-test
tStat, pValue = ttest_ind(higherMentalScores, lowerMentalScores,␣
 ↪equal_var=(pLevene > 0.05))
```

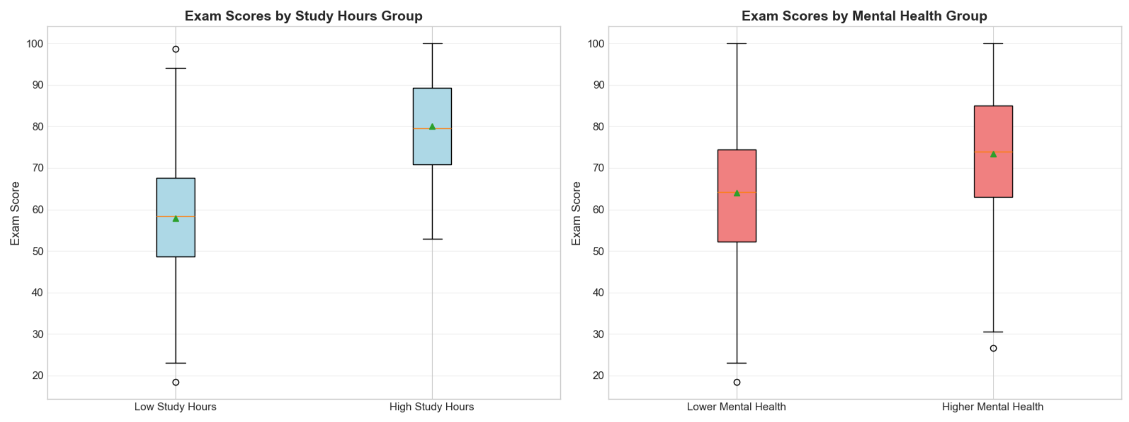
```python
print("Independent Samples T-Test Results:")
print(f"t-statistic: {tStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
 ↪at   = 0.05")

# Calculate Cohen's d (effect size)
pooledStd = np.sqrt(((len(lowerMentalScores)-1)*lowerMentalScores.std()**2 +
                      (len(higherMentalScores)-1)*higherMentalScores.std()**2) /
                     (len(lowerMentalScores) + len(higherMentalScores) - 2))
cohensD = (higherMentalScores.mean() - lowerMentalScores.mean()) / pooledStd

print(f"Effect Size (Cohen's d): {cohensD:.4f}")
if abs(cohensD) < 0.2:
    effect = "negligible"
elif abs(cohensD) < 0.5:
    effect = "small"
elif abs(cohensD) < 0.8:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
Independent Samples T-Test Results:
t-statistic: 9.0529
p-value: 7.1393e-19
Significance: Significant at   = 0.05
Effect Size (Cohen's d): 0.5818
Interpretation: Medium effect
```

1.5 4. Visualization of T-Test Results

[396]:
```python
# Create comparison plots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Study hours comparison
boxData1 = [lowStudyScores, highStudyScores]
bp1 = axes[0].boxplot(boxData1, labels=['Low Study Hours', 'High Study Hours'],
                       patch_artist=True, showmeans=True)
for patch in bp1['boxes']:
    patch.set_facecolor('lightblue')
axes[0].set_ylabel('Exam Score', fontsize=12)
axes[0].set_title('Exam Scores by Study Hours Group', fontsize=14,␣
 ↪fontweight='bold')
axes[0].grid(True, alpha=0.3, axis='y')

# Mental health comparison
```

```
boxData2 = [lowerMentalScores, higherMentalScores]
bp2 = axes[1].boxplot(boxData2, labels=['Lower Mental Health', 'Higher Mental␣
↪Health'],
                       patch_artist=True, showmeans=True)
for patch in bp2['boxes']:
    patch.set_facecolor('lightcoral')
axes[1].set_ylabel('Exam Score', fontsize=12)
axes[1].set_title('Exam Scores by Mental Health Group', fontsize=14,␣
↪fontweight='bold')
axes[1].grid(True, alpha=0.3, axis='y')

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



1.6 5. One-Way ANOVA (3+ Groups)

1.6.1 5.1 Study Hours Groups: One-Way ANOVA

Hypotheses: - H : Mean exam scores are equal across all study hour groups (Low, Medium, High) - H : At least one group mean differs from the others

[397]:
```
# Separate groups
lowStudy = df[df['studyGroup3'] == 'Low']['exam_score']
medStudy = df[df['studyGroup3'] == 'Medium']['exam_score']
highStudy = df[df['studyGroup3'] == 'High']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Study Hours Group (3 levels):")
print(f"Low (n={len(lowStudy)}):")
print(f"Mean: {lowStudy.mean():.2f}, SD: {lowStudy.std():.2f}")
print(f"Medium (n={len(medStudy)}):")
print(f"Mean: {medStudy.mean():.2f}, SD: {medStudy.std():.2f}")
print(f"High (n={len(highStudy)}):")
print(f"Mean: {highStudy.mean():.2f}, SD: {highStudy.std():.2f}")
```

```
Descriptive Statistics by Study Hours Group (3 levels):
Low (n=314):
Mean: 53.45, SD: 12.40
Medium (n=347):
Mean: 69.64, SD: 10.15
High (n=339):
Mean: 84.52, SD: 11.38
```

[398]:
```
# Check assumptions for ANOVA

# 1. Normality
_, pLow = shapiro(lowStudy)
_, pMed = shapiro(medStudy)
_, pHigh = shapiro(highStudy)
print("1. Normality (Shapiro-Wilk):")
print(f"Low: p = {pLow:.4f}")
print(f"Medium: p = {pMed:.4f}")
print(f"High: p = {pHigh:.4f}")

# 2. Homogeneity of variance
statLevene, pLevene = levene(lowStudy, medStudy, highStudy)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
  ↪variances)'}")
```

```
1. Normality (Shapiro-Wilk):
Low: p = 0.6678
Medium: p = 0.7494
High: p = 0.0000
2. Homogeneity of Variance (Levene's test):
p = 0.0003 (Unequal variances)
```

[399]:
```
# Perform ANOVA
fStat, pValue = f_oneway(lowStudy, medStudy, highStudy)

print("One-Way ANOVA Results:")
print(f"F-statistic: {fStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
  ↪at   = 0.05")

# Calculate eta-squared (effect size)
allScores = pd.concat([lowStudy, medStudy, highStudy])
ssTotal = np.sum((allScores - allScores.mean())**2)
ssBetween = (len(lowStudy) * (lowStudy.mean() - allScores.mean())**2 +
             len(medStudy) * (medStudy.mean() - allScores.mean())**2 +
             len(highStudy) * (highStudy.mean() - allScores.mean())**2)
etaSquared = ssBetween / ssTotal

print(f"Effect Size ( ²): {etaSquared:.4f}")
if etaSquared < 0.01:
    effect = "negligible"
elif etaSquared < 0.06:
    effect = "small"
elif etaSquared < 0.14:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
One-Way ANOVA Results:
F-statistic: 614.6671
p-value: 1.1905e-174
Significance: Significant at   = 0.05
Effect Size ( ²): 0.5522
Interpretation: Large effect
```

[400]:
```
# If ANOVA is Significant)
if pValue < 0.05:
    print("Post-Hoc Analysis (Tukey HSD):")

    # Prepare data for Tukey HSD
    studyData = df[['exam_score', 'studyGroup3']].copy()
    tukey = pairwise_tukeyhsd(endog=studyData['exam_score'],
                              groups=studyData['studyGroup3'],
                              alpha=0.05)

    print(tukey)

    print("Conclusion:")
    print("Exam scores differ Significantly across study hour groups.")
    print("See post-hoc results above for pairwise comparisons.")
else:
    print("Conclusion:")
    print("No Significant differences in exam scores across study hour groups.")
```

```
Post-Hoc Analysis (Tukey HSD):
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
  High    Low -31.0675   0.0 -33.1473 -28.9877   True
  High Medium -14.8809   0.0 -16.9087  -12.853   True
   Low Medium  16.1866   0.0  14.1184  18.2549   True
```

```
-----------------------------------------------------
Conclusion:
Exam scores differ Significantly across study hour groups.
See post-hoc results above for pairwise comparisons.
```

1.6.2 5.2 Mental Health Groups: One-Way ANOVA

Hypotheses: - H : Mean exam scores are equal across all mental health groups (Low, Medium, High) - H : At least one group mean differs from the others

[401]:
```python
# Separate groups
lowMental = df[df['mentalGroup3'] == 'Low']['exam_score']
medMental = df[df['mentalGroup3'] == 'Medium']['exam_score']
highMental = df[df['mentalGroup3'] == 'High']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Mental Health Group (3 levels):")
print(f"Low (n={len(lowMental)}):")
print(f"Mean: {lowMental.mean():.2f}, SD: {lowMental.std():.2f}")
print(f"Medium (n={len(medMental)}):")
print(f"Mean: {medMental.mean():.2f}, SD: {medMental.std():.2f}")
print(f"High (n={len(highMental)}):")
print(f"Mean: {highMental.mean():.2f}, SD: {highMental.std():.2f}")
```

```
Descriptive Statistics by Mental Health Group (3 levels):
Low (n=301):
Mean: 63.43, SD: 15.94
Medium (n=317):
Mean: 67.87, SD: 16.85
High (n=382):
Mean: 75.90, SD: 15.49
```

[402]:
```python
# Check assumptions
print("Assumption Checks for ANOVA:")

# 1. Normality
_, pLow = shapiro(lowMental)
_, pMed = shapiro(medMental)
_, pHigh = shapiro(highMental)
print("Normality (Shapiro-Wilk):")
print(f"Low: p = {pLow:.4f}")
print(f"Medium: p = {pMed:.4f}")
print(f"High: p = {pHigh:.4f}")

# 2. Homogeneity of variance
statLevene, pLevene = levene(lowMental, medMental, highMental)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
  ↪variances)'}")
```

```
Assumption Checks for ANOVA:
Normality (Shapiro-Wilk):
Low: p = 0.0848
Medium: p = 0.0105
High: p = 0.0000
2. Homogeneity of Variance (Levene's test):
p = 0.2683 (Equal variances)
```

[403]:
```
# Perform ANOVA
fStat, pValue = f_oneway(lowMental, medMental, highMental)

print("One-Way ANOVA Results:")
print(f"F-statistic: {fStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
  ↪at   = 0.05")

# Calculate eta-squared (effect size)
allScores = pd.concat([lowMental, medMental, highMental])
ssTotal = np.sum((allScores - allScores.mean())**2)
ssBetween = (len(lowMental) * (lowMental.mean() - allScores.mean())**2 +
             len(medMental) * (medMental.mean() - allScores.mean())**2 +
             len(highMental) * (highMental.mean() - allScores.mean())**2)
etaSquared = ssBetween / ssTotal

print(f"\nEffect Size ( ²): {etaSquared:.4f}")
if etaSquared < 0.01:
    effect = "negligible"
elif etaSquared < 0.06:
    effect = "small"
elif etaSquared < 0.14:
    effect = "medium"
else:
    effect = "large"
print(f"  Interpretation: {effect.capitalize()} effect")
```

```
One-Way ANOVA Results:
F-statistic: 53.4275
p-value: 9.0758e-23
Significance: Significant at   = 0.05

Effect Size ( ²): 0.0968
  Interpretation: Medium effect
```

[404]:
```
# Post-hoc test (if ANOVA is Significant)
if pValue < 0.05:
    print("\nPost-Hoc Analysis (Tukey HSD):")

    # Prepare data for Tukey HSD
    mentalData = df[['exam_score', 'mentalGroup3']].copy()
    tukey = pairwise_tukeyhsd(endog=mentalData['exam_score'],
                              groups=mentalData['mentalGroup3'],
                              alpha=0.05)

    print(tukey)

    print("Conclusion:")
    print("Exam scores differ Significantly across mental health groups.")
    print("See post-hoc results above for pairwise comparisons.")
else:
    print("Conclusion:")
    print("No Significant differences in exam scores across mental health␣
 ↪groups.")
```

```
Post-Hoc Analysis (Tukey HSD):
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper  reject
-----------------------------------------------------
  High    Low -12.4716    0.0 -15.3781 -9.5651   True
  High Medium  -8.0369    0.0 -10.9021 -5.1717   True
   Low Medium   4.4347 0.0018   1.3997  7.4696   True
-----------------------------------------------------
Conclusion:
Exam scores differ Significantly across mental health groups.
See post-hoc results above for pairwise comparisons.
```

1.6.3 5.3 Exercise Frequency Groups: One-Way ANOVA

Hypotheses: - H : Mean exam scores are equal across all exercise frequency groups (Low, Medium, High) - H : At least one group mean differs from the others

[405]:
```
# Separate groups
lowExercise = df[df['exerciseGroup3'] == 'Low']['exam_score']
medExercise = df[df['exerciseGroup3'] == 'Medium']['exam_score']
highExercise = df[df['exerciseGroup3'] == 'High']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Exercise Frequency Group (3 levels):")
print(f"Low (n={len(lowExercise)}):")
print(f"Mean: {lowExercise.mean():.2f}, SD: {lowExercise.std():.2f}")
print(f"Medium (n={len(medExercise)}):")
print(f"Mean: {medExercise.mean():.2f}, SD: {medExercise.std():.2f}")
print(f"High (n={len(highExercise)}):")
print(f"Mean: {highExercise.mean():.2f}, SD: {highExercise.std():.2f}")
```

```
Descriptive Statistics by Exercise Frequency Group (3 levels):
Low (n=290):
Mean: 66.09, SD: 16.97
Medium (n=275):
Mean: 69.30, SD: 15.91
High (n=435):
Mean: 72.13, SD: 17.04
```

[406]:
```
# Check assumptions
print("Assumption Checks for ANOVA:")

# 1. Normality
_, pLow = shapiro(lowExercise)
_, pMed = shapiro(medExercise)
_, pHigh = shapiro(highExercise)
print("Normality (Shapiro-Wilk):")
print(f"Low: p = {pLow:.4f}")
print(f"Medium: p = {pMed:.4f}")
print(f"High: p = {pHigh:.4f}")

# 2. Homogeneity of variance
statLevene, pLevene = levene(lowExercise, medExercise, highExercise)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
 ↪variances)'}")
```

```
Assumption Checks for ANOVA:
Normality (Shapiro-Wilk):
Low: p = 0.0264
Medium: p = 0.0301
High: p = 0.0000
2. Homogeneity of Variance (Levene's test):
p = 0.2589 (Equal variances)
```

[407]:
```
# Perform ANOVA
fStat, pValue = f_oneway(lowExercise, medExercise, highExercise)

print("One-Way ANOVA Results:")
print(f"F-statistic: {fStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
 ↪at   = 0.05")
```

```
# Calculate eta-squared (effect size)
allScores = pd.concat([lowExercise, medExercise, highExercise])
ssTotal = np.sum((allScores - allScores.mean())**2)
ssBetween = (len(lowExercise) * (lowExercise.mean() - allScores.mean())**2 +
             len(medExercise) * (medExercise.mean() - allScores.mean())**2 +
             len(highExercise) * (highExercise.mean() - allScores.mean())**2)
etaSquared = ssBetween / ssTotal

print(f"Effect Size ( ²): {etaSquared:.4f}")
if etaSquared < 0.01:
    effect = "negligible"
elif etaSquared < 0.06:
    effect = "small"
elif etaSquared < 0.14:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
One-Way ANOVA Results:
F-statistic: 11.4395
p-value: 1.2248e-05
Significance: Significant at   = 0.05
Effect Size ( ²): 0.0224
Interpretation: Small effect
```

[408]:
```
# Post-hoc test (if ANOVA is Significant)
if pValue < 0.05:
    print("Post-Hoc Analysis (Tukey HSD):")

    # Prepare data for Tukey HSD
    exerciseData = df[['exam_score', 'exerciseGroup3']].copy()
    tukey = pairwise_tukeyhsd(endog=exerciseData['exam_score'],
                              groups=exerciseData['exerciseGroup3'],
                              alpha=0.05)

    print(tukey)

    print("Conclusion:")
    print("Exam scores differ Significantly across exercise frequency groups.")
else:
    print("Conclusion:")
    print("No Significant differences in exam scores across exercise frequency␣
 ↪groups.")
```

```
Post-Hoc Analysis (Tukey HSD):
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower   upper  reject
----------------------------------------------------
  High    Low  -6.0448    0.0 -9.0191 -3.0706   True
  High Medium  -2.8313 0.0719 -5.8538  0.1913  False
   Low Medium   3.2135 0.0585 -0.0887  6.5158  False
----------------------------------------------------
Conclusion:
Exam scores differ Significantly across exercise frequency groups.
```

1.6.4 5.4 Netflix Hours Groups: One-Way ANOVA

Hypotheses: - H : Mean exam scores are equal across all netflix hours groups (Low, Medium, High) - H : At least one group mean differs from the others

[409]:
```python
# Separate groups
lowNetflix = df[df['netflixGroup3'] == 'Low']['exam_score']
medNetflix = df[df['netflixGroup3'] == 'Medium']['exam_score']
highNetflix = df[df['netflixGroup3'] == 'High']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Netflix Hours Group (3 levels):")
print(f"Low (n={len(lowNetflix)}):")
print(f"Mean: {lowNetflix.mean():.2f}, SD: {lowNetflix.std():.2f}")
print(f"Medium (n={len(medNetflix)}):")
print(f"Mean: {medNetflix.mean():.2f}, SD: {medNetflix.std():.2f}")
print(f"High (n={len(highNetflix)}):")
print(f"Mean: {highNetflix.mean():.2f}, SD: {highNetflix.std():.2f}")
```

```
Descriptive Statistics by Netflix Hours Group (3 levels):
Low (n=319):
Mean: 73.14, SD: 15.93
Medium (n=339):
Mean: 69.40, SD: 16.61
High (n=342):
Mean: 66.51, SD: 17.45
```

[410]:
```python
# Check assumptions
print("Assumption Checks for ANOVA:")

# 1. Normality
_, pLow = shapiro(lowNetflix)
_, pMed = shapiro(medNetflix)
_, pHigh = shapiro(highNetflix)
print("Normality (Shapiro-Wilk):")
print(f"Low: p = {pLow:.4f}")
print(f"Medium: p = {pMed:.4f}")
print(f"High: p = {pHigh:.4f}")
```

```
# 2. Homogeneity of variance
statLevene, pLevene = levene(lowNetflix, medNetflix, highNetflix)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
↪variances)'}")
```

```
Assumption Checks for ANOVA:
Normality (Shapiro-Wilk):
Low: p = 0.0002
Medium: p = 0.0008
High: p = 0.0149
2. Homogeneity of Variance (Levene's test):
p = 0.1869 (Equal variances)
```

[411]:
```
# Perform ANOVA
fStat, pValue = f_oneway(lowNetflix, medNetflix, highNetflix)

print("One-Way ANOVA Results:")
print(f"F-statistic: {fStat:.4f}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
↪at   = 0.05")

# Calculate eta-squared (effect size)
allScores = pd.concat([lowNetflix, medNetflix, highNetflix])
ssTotal = np.sum((allScores - allScores.mean())**2)
ssBetween = (len(lowNetflix) * (lowNetflix.mean() - allScores.mean())**2 +
             len(medNetflix) * (medNetflix.mean() - allScores.mean())**2 +
             len(highNetflix) * (highNetflix.mean() - allScores.mean())**2)
etaSquared = ssBetween / ssTotal

print(f"Effect Size ( ²): {etaSquared:.4f}")
if etaSquared < 0.01:
    effect = "negligible"
elif etaSquared < 0.06:
    effect = "small"
elif etaSquared < 0.14:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
One-Way ANOVA Results:
F-statistic: 13.0730
p-value: 2.4867e-06
Significance: Significant at   = 0.05
Effect Size ( ²): 0.0256
Interpretation: Small effect
```

[412]:
```python
# Post-hoc test (if ANOVA is Significant)
if pValue < 0.05:
    print("\nPost-Hoc Analysis (Tukey HSD):")

    # Prepare data for Tukey HSD
    netflixData = df[['exam_score', 'netflixGroup3']].copy()
    tukey = pairwise_tukeyhsd(endog=netflixData['exam_score'],
                              groups=netflixData['netflixGroup3'],
                              alpha=0.05)

    print(tukey)

    print("Conclusion:")
    print("Exam scores differ Significantly across netflix hours groups.")
else:
    print("Conclusion:")
    print("No Significant differences in exam scores across netflix hours␣
 ↪groups.")
```

```
Post-Hoc Analysis (Tukey HSD):
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower   upper  reject
----------------------------------------------------
  High    Low   6.6321    0.0  3.5832  9.6811   True
  High Medium   2.8891 0.0622 -0.1129  5.8912  False
   Low Medium   -3.743 0.0115 -6.7984 -0.6875   True
----------------------------------------------------
Conclusion:
Exam scores differ Significantly across netflix hours groups.
```

1.6.5 5.5 Diet Quality Groups: One-Way ANOVA

Hypotheses: - H : Mean exam scores are equal across all diet quality groups (Poor, Fair, Good) - H : At least one group mean differs from the others

[413]:
```python
# Separate groups (diet_quality is already categorical)
poorDiet = df[df['diet_quality'] == 'Poor']['exam_score']
fairDiet = df[df['diet_quality'] == 'Fair']['exam_score']
goodDiet = df[df['diet_quality'] == 'Good']['exam_score']

# Descriptive statistics
print("Descriptive Statistics by Diet Quality Group:")
print(f"Poor (n={len(poorDiet)}):")
print(f"Mean: {poorDiet.mean():.2f}, SD: {poorDiet.std():.2f}")
print(f"Fair (n={len(fairDiet)}):")
print(f"Mean: {fairDiet.mean():.2f}, SD: {fairDiet.std():.2f}")
print(f"Good (n={len(goodDiet)}):")
print(f"Mean: {goodDiet.mean():.2f}, SD: {goodDiet.std():.2f}")
```

```
Descriptive Statistics by Diet Quality Group:
Poor (n=185):
Mean: 68.13, SD: 17.06
Fair (n=437):
Mean: 70.43, SD: 16.65
Good (n=378):
Mean: 69.37, SD: 17.07
```

[414]:
```python
# Check assumptions
print("Assumption Checks for ANOVA:")

# 1. Normality
_, pPoor = shapiro(poorDiet)
_, pFair = shapiro(fairDiet)
_, pGood = shapiro(goodDiet)
print("Normality (Shapiro-Wilk):")
print(f"Poor: p = {pPoor:.4f}")
print(f"Fair: p = {pFair:.4f}")
print(f"Good: p = {pGood:.4f}\n")

# 2. Homogeneity of variance
statLevene, pLevene = levene(poorDiet, fairDiet, goodDiet)
print(f"2. Homogeneity of Variance (Levene's test):")
print(f"p = {pLevene:.4f} {'(Equal variances)' if pLevene > 0.05 else '(Unequal␣
↪variances)'}")
```

```
Assumption Checks for ANOVA:
Normality (Shapiro-Wilk):
Poor: p = 0.0332
Fair: p = 0.0002
Good: p = 0.0012

2. Homogeneity of Variance (Levene's test):
p = 0.8866 (Equal variances)
```

[415]:
```python
# Perform ANOVA
fStat, pValue = f_oneway(poorDiet, fairDiet, goodDiet)

print("One-Way ANOVA Results:")
print(f"  F-statistic: {fStat:.4f}")
print(f"  p-value: {pValue:.4e}")
print(f"  Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
↪at   = 0.05\n")

# Calculate eta-squared (effect size)
```

```
allScores = pd.concat([poorDiet, fairDiet, goodDiet])
ssTotal = np.sum((allScores - allScores.mean())**2)
ssBetween = (len(poorDiet) * (poorDiet.mean() - allScores.mean())**2 +
             len(fairDiet) * (fairDiet.mean() - allScores.mean())**2 +
             len(goodDiet) * (goodDiet.mean() - allScores.mean())**2)
etaSquared = ssBetween / ssTotal

print(f"Effect Size ( ²): {etaSquared:.4f}")
if etaSquared < 0.01:
    effect = "negligible"
elif etaSquared < 0.06:
    effect = "small"
elif etaSquared < 0.14:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
One-Way ANOVA Results:
  F-statistic: 1.2662
  p-value: 2.8235e-01
  Significance: Not Significant at   = 0.05

Effect Size ( ²): 0.0025
Interpretation: Negligible effect
```

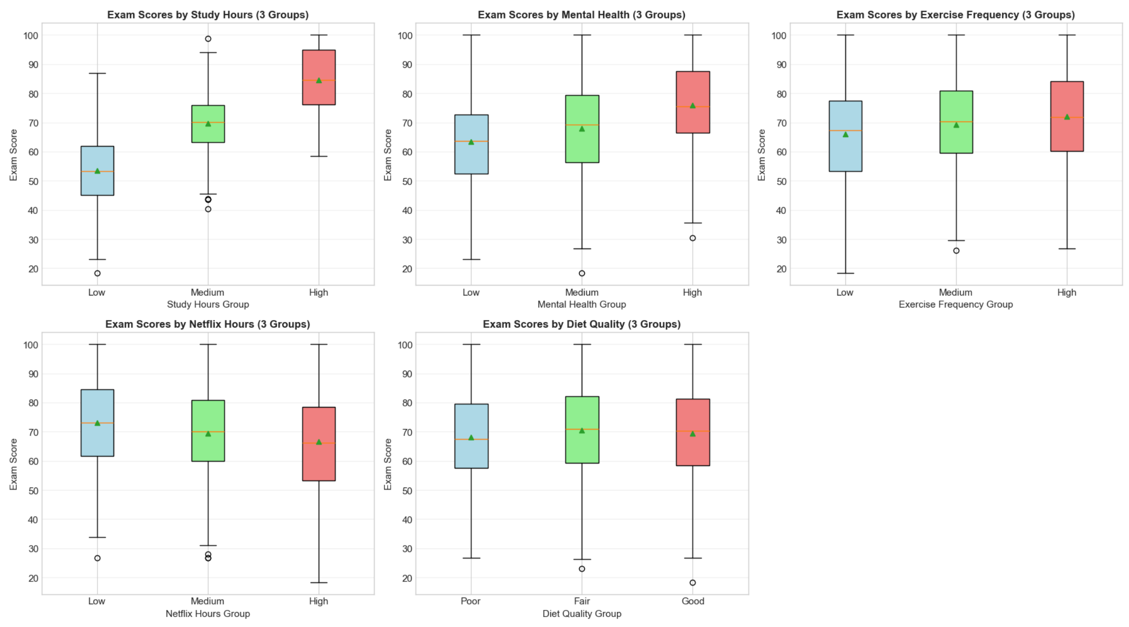
[416]:
```
# Post-hoc test (if ANOVA is Significant)
if pValue < 0.05:
    print("Post-Hoc Analysis (Tukey HSD):")

    # Prepare data for Tukey HSD
    dietData = df[['exam_score', 'diet_quality']].copy()
    tukey = pairwise_tukeyhsd(endog=dietData['exam_score'],
                              groups=dietData['diet_quality'],
                              alpha=0.05)

    print(tukey)

    print("Conclusion:")
    print("Exam scores differ Significantly across diet quality groups.")
else:
    print("Conclusion:")
    print("No Significant differences in exam scores across diet quality groups.
 ↪")
```

```
Conclusion:
No Significant differences in exam scores across diet quality groups.
```

1.6.6 5.6 Visualization of ANOVA Results

```
[417]: # Create comparison plots for all ANOVA analyses
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

colors = ['lightblue', 'lightgreen', 'lightcoral']

# Study hours ANOVA visualization
boxData1 = [lowStudy, medStudy, highStudy]
bp1 = axes[0, 0].boxplot(boxData1, labels=['Low', 'Medium', 'High'],
                       patch_artist=True, showmeans=True)
for patch, color in zip(bp1['boxes'], colors):
    patch.set_facecolor(color)
axes[0, 0].set_xlabel('Study Hours Group', fontsize=11)
axes[0, 0].set_ylabel('Exam Score', fontsize=11)
axes[0, 0].set_title('Exam Scores by Study Hours (3 Groups)', fontsize=12,␣
 ↪fontweight='bold')
axes[0, 0].grid(True, alpha=0.3, axis='y')

# Mental health ANOVA visualization
boxData2 = [lowMental, medMental, highMental]
bp2 = axes[0, 1].boxplot(boxData2, labels=['Low', 'Medium', 'High'],
                       patch_artist=True, showmeans=True)
for patch, color in zip(bp2['boxes'], colors):
    patch.set_facecolor(color)
axes[0, 1].set_xlabel('Mental Health Group', fontsize=11)
axes[0, 1].set_ylabel('Exam Score', fontsize=11)
axes[0, 1].set_title('Exam Scores by Mental Health (3 Groups)', fontsize=12,␣
 ↪fontweight='bold')
axes[0, 1].grid(True, alpha=0.3, axis='y')

# Exercise frequency ANOVA visualization
boxData3 = [lowExercise, medExercise, highExercise]
bp3 = axes[0, 2].boxplot(boxData3, labels=['Low', 'Medium', 'High'],
                       patch_artist=True, showmeans=True)
for patch, color in zip(bp3['boxes'], colors):
    patch.set_facecolor(color)
axes[0, 2].set_xlabel('Exercise Frequency Group', fontsize=11)
axes[0, 2].set_ylabel('Exam Score', fontsize=11)
axes[0, 2].set_title('Exam Scores by Exercise Frequency (3 Groups)',␣
 ↪fontsize=12, fontweight='bold')
axes[0, 2].grid(True, alpha=0.3, axis='y')

# Netflix hours ANOVA visualization
boxData4 = [lowNetflix, medNetflix, highNetflix]
bp4 = axes[1, 0].boxplot(boxData4, labels=['Low', 'Medium', 'High'],
                       patch_artist=True, showmeans=True)
```

```
for patch, color in zip(bp4['boxes'], colors):
    patch.set_facecolor(color)
axes[1, 0].set_xlabel('Netflix Hours Group', fontsize=11)
axes[1, 0].set_ylabel('Exam Score', fontsize=11)
axes[1, 0].set_title('Exam Scores by Netflix Hours (3 Groups)', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 0].grid(True, alpha=0.3, axis='y')

# Diet quality ANOVA visualization
boxData5 = [poorDiet, fairDiet, goodDiet]
bp5 = axes[1, 1].boxplot(boxData5, labels=['Poor', 'Fair', 'Good'],
                       patch_artist=True, showmeans=True)
for patch, color in zip(bp5['boxes'], colors):
    patch.set_facecolor(color)
axes[1, 1].set_xlabel('Diet Quality Group', fontsize=11)
axes[1, 1].set_ylabel('Exam Score', fontsize=11)
axes[1, 1].set_title('Exam Scores by Diet Quality (3 Groups)', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 1].grid(True, alpha=0.3, axis='y')

# Remove the extra subplot
fig.delaxes(axes[1, 2])

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



1.7 5.7 Chi-Squared Tests for Categorical Associations

Examining relationships between exam performance categories and categorical variables.

[430]:
```
# Create exam performance categories (tertiles to match ANOVA grouping)
examQ1 = df['exam_score'].quantile(0.33)
examQ2 = df['exam_score'].quantile(0.67)
df['examPerformance'] = df['exam_score'].apply(
    lambda x: 'Low' if x < examQ1 else ('Medium' if x < examQ2 else 'High')
)

print("Exam Performance Category Distribution:")
print(df['examPerformance'].value_counts().sort_index())
print(f"\nCutoff scores:")
print(f"Low: < {examQ1:.2f}")
print(f"Medium: {examQ1:.2f} - {examQ2:.2f}")
print(f"High: > {examQ2:.2f}")
```

```
Exam Performance Category Distribution:
examPerformance
High      330
Low       329
Medium    341
Name: count, dtype: int64

Cutoff scores:
Low: < 62.70
Medium: 62.70 - 77.03
High: > 77.03
```

1.7.1 5.7.1 Exam Performance × Parental Education Level

Hypotheses: - H : Exam performance is independent of parental education level - H : Exam performance is associated with parental education level

[419]:
```
# Create contingency table
contingencyTable1 = pd.crosstab(df['examPerformance'],␣
  ↪df['parental_education_level'])

print("Contingency Table: Exam Performance × Parental Education Level")
print(contingencyTable1)
print(f"\nTotal observations: {contingencyTable1.sum().sum()}")
```

```
Contingency Table: Exam Performance × Parental Education Level
parental_education_level  Bachelor  High School  Master  None
examPerformance
High                           120          134      47    29
Low                            106          134      60    29
Medium                         124          124      60    33
```

```
Total observations: 1000
```

[420]:
```
from scipy.stats import chi2_contingency

# Perform Chi-squared test
chi2Stat, pValue, dof, expectedFreq = chi2_contingency(contingencyTable1)

print("Chi-Squared Test Results:")
print(f"Chi-squared statistic: {chi2Stat:.4f}")
print(f"Degrees of freedom: {dof}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
↪at   = 0.05")

# Calculate Cramér's V (effect size)
n = contingencyTable1.sum().sum()
minDim = min(contingencyTable1.shape[0] - 1, contingencyTable1.shape[1] - 1)
cramersV = np.sqrt(chi2Stat / (n * minDim))

print(f"\nEffect Size (Cramér's V): {cramersV:.4f}")
if cramersV < 0.1:
    effect = "negligible"
elif cramersV < 0.3:
    effect = "small"
elif cramersV < 0.5:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

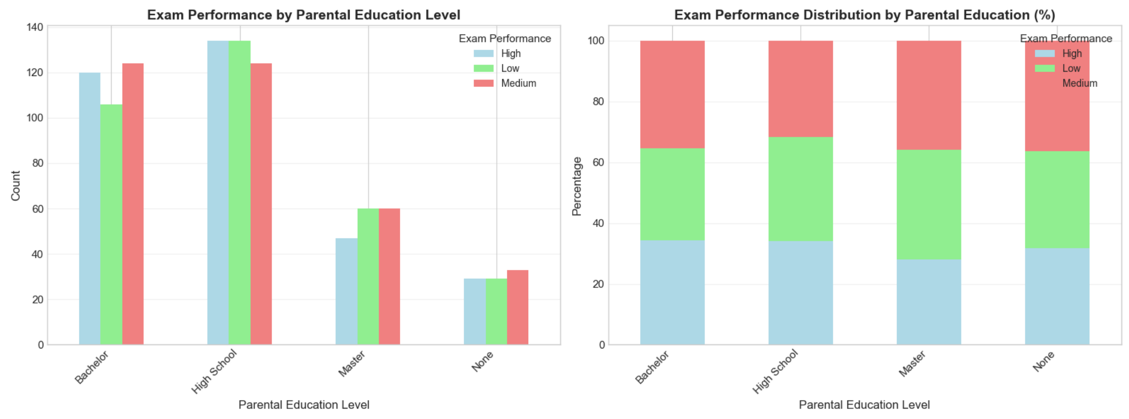
```
Chi-Squared Test Results:
Chi-squared statistic: 4.1585
Degrees of freedom: 6
p-value: 6.5524e-01
Significance: Not Significant at   = 0.05

Effect Size (Cramér's V): 0.0456
Interpretation: Negligible effect
```

[421]:
```
# Display expected frequencies
print("\nExpected Frequencies:")
expectedDf = pd.DataFrame(expectedFreq,
                          index=contingencyTable1.index,
                          columns=contingencyTable1.columns)
print(expectedDf.round(2))

# Check assumption: all expected frequencies >= 5
```

```
minExpected = expectedFreq.min()
print(f"\nMinimum expected frequency: {minExpected:.2f}")
if minExpected >= 5:
    print("Assumption satisfied: All expected frequencies   5")
else:
    print("Warning: Some expected frequencies < 5; results may be unreliable")
```

```
Expected Frequencies:
parental_education_level  Bachelor  High School  Master   None
examPerformance
High                        115.50       129.36   55.11  30.03
Low                         115.15       128.97   54.94  29.94
Medium                      119.35       133.67   56.95  31.03

Minimum expected frequency: 29.94
Assumption satisfied: All expected frequencies   5
```

[422]:
```
# Visualize with grouped bar chart
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Grouped bar chart
contingencyTable1.T.plot(kind='bar', ax=axes[0], color=['lightblue',␣
 ↪'lightgreen', 'lightcoral'])
axes[0].set_xlabel('Parental Education Level', fontsize=12)
axes[0].set_ylabel('Count', fontsize=12)
axes[0].set_title('Exam Performance by Parental Education Level', fontsize=14,␣
 ↪fontweight='bold')
axes[0].legend(title='Exam Performance', title_fontsize=11, fontsize=10)
axes[0].grid(True, alpha=0.3, axis='y')
plt.setp(axes[0].xaxis.get_majorticklabels(), rotation=45, ha='right')

# Stacked percentage bar chart
contingencyPct1 = contingencyTable1.T.div(contingencyTable1.T.sum(axis=1),␣
 ↪axis=0) * 100
contingencyPct1.plot(kind='bar', stacked=True, ax=axes[1],
                      color=['lightblue', 'lightgreen', 'lightcoral'])
axes[1].set_xlabel('Parental Education Level', fontsize=12)
axes[1].set_ylabel('Percentage', fontsize=12)
axes[1].set_title('Exam Performance Distribution by Parental Education (%)',␣
 ↪fontsize=14, fontweight='bold')
axes[1].legend(title='Exam Performance', title_fontsize=11, fontsize=10)
axes[1].grid(True, alpha=0.3, axis='y')
plt.setp(axes[1].xaxis.get_majorticklabels(), rotation=45, ha='right')

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



1.7.2 5.7.2 Exam Performance × Diet Quality

Hypotheses: - H : Exam performance is independent of diet quality - H : Exam performance is associated with diet quality

[423]:
```
# Create contingency table
contingencyTable2 = pd.crosstab(df['examPerformance'], df['diet_quality'])

print("Contingency Table: Exam Performance × Diet Quality")
print(contingencyTable2)
print(f"\nTotal observations: {contingencyTable2.sum().sum()}")
```

```
Contingency Table: Exam Performance × Diet Quality
diet_quality     Fair  Good  Poor
examPerformance
High              152   122    56
Low               143   121    65
Medium            142   135    64

Total observations: 1000
```

[424]:
```
# Perform Chi-squared test
chi2Stat, pValue, dof, expectedFreq = chi2_contingency(contingencyTable2)

print("Chi-Squared Test Results:")
print(f"Chi-squared statistic: {chi2Stat:.4f}")
print(f"Degrees of freedom: {dof}")
print(f"p-value: {pValue:.4e}")
print(f"Significance: {'Significant' if pValue < 0.05 else 'Not Significant'}␣
  ↪at   = 0.05")

# Calculate Cramér's V (effect size)
n = contingencyTable2.sum().sum()
```

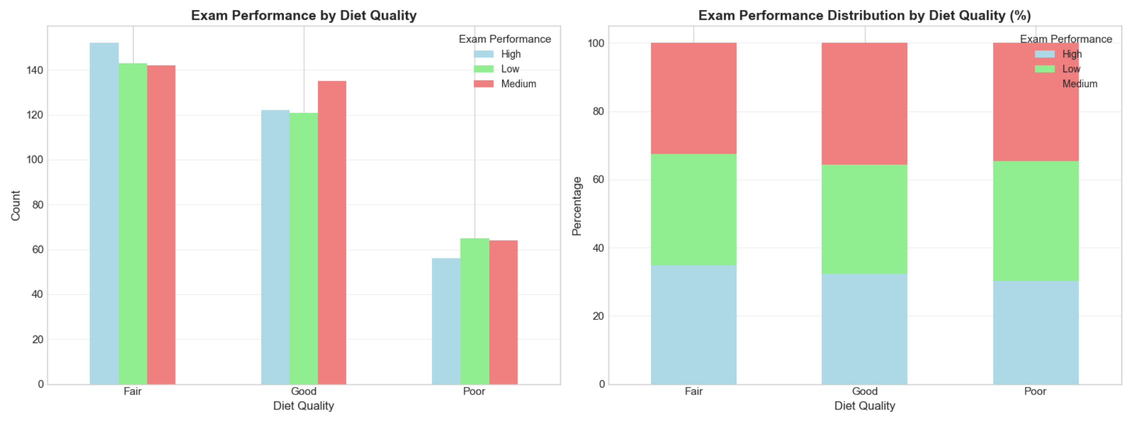
```
minDim = min(contingencyTable2.shape[0] - 1, contingencyTable2.shape[1] - 1)
cramersV = np.sqrt(chi2Stat / (n * minDim))

print(f"\nEffect Size (Cramér's V): {cramersV:.4f}")
if cramersV < 0.1:
    effect = "negligible"
elif cramersV < 0.3:
    effect = "small"
elif cramersV < 0.5:
    effect = "medium"
else:
    effect = "large"
print(f"Interpretation: {effect.capitalize()} effect")
```

```
Chi-Squared Test Results:
Chi-squared statistic: 1.9072
Degrees of freedom: 4
p-value: 7.5281e-01
Significance: Not Significant at   = 0.05

Effect Size (Cramér's V): 0.0309
Interpretation: Negligible effect
```

[425]:
```
# Display expected frequencies
print("\nExpected Frequencies:")
expectedDf = pd.DataFrame(expectedFreq,
                          index=contingencyTable2.index,
                          columns=contingencyTable2.columns)
print(expectedDf.round(2))

# Check assumption: all expected frequencies >= 5
minExpected = expectedFreq.min()
print(f"\nMinimum expected frequency: {minExpected:.2f}")
if minExpected >= 5:
    print("Assumption satisfied: All expected frequencies   5")
else:
    print("Warning: Some expected frequencies < 5; results may be unreliable")
```

```
Expected Frequencies:
diet_quality       Fair    Good   Poor
examPerformance
High             144.21  124.74  61.05
Low              143.77  124.36  60.86
Medium           149.02  128.90  63.08

Minimum expected frequency: 60.87
Assumption satisfied: All expected frequencies   5
```

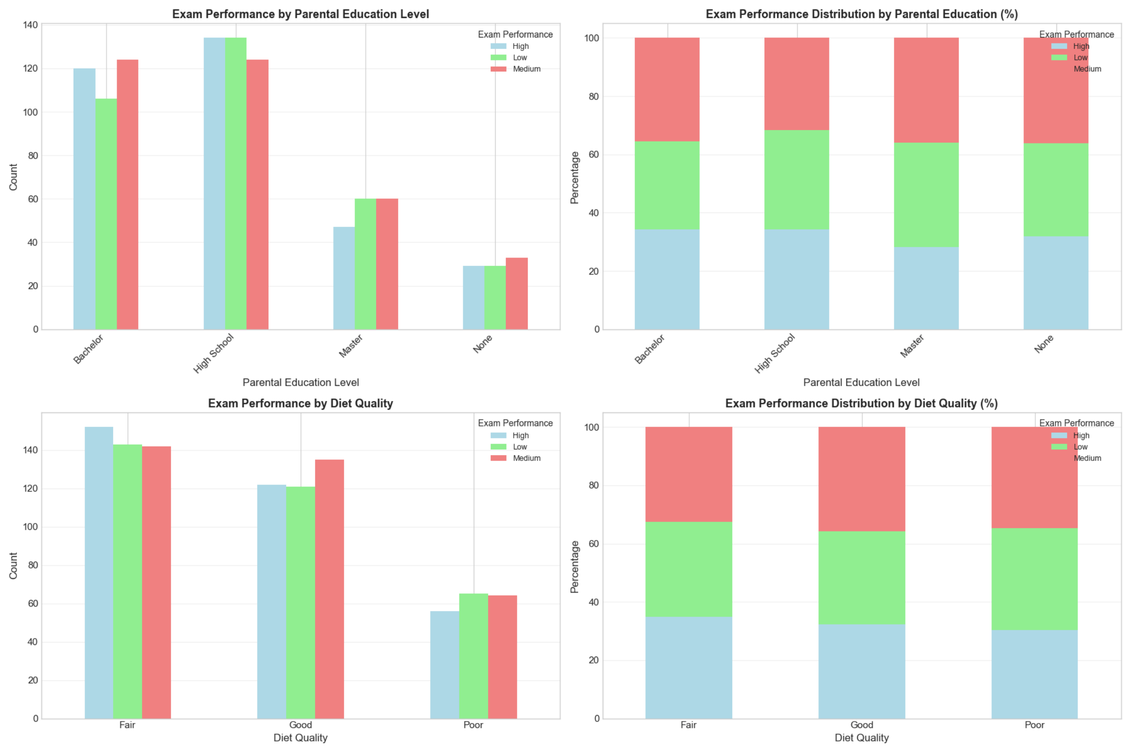
```
[426]: # Visualize with grouped bar chart
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Grouped bar chart
contingencyTable2.T.plot(kind='bar', ax=axes[0], color=['lightblue',␣
 ↪'lightgreen', 'lightcoral'])
axes[0].set_xlabel('Diet Quality', fontsize=12)
axes[0].set_ylabel('Count', fontsize=12)
axes[0].set_title('Exam Performance by Diet Quality', fontsize=14,␣
 ↪fontweight='bold')
axes[0].legend(title='Exam Performance', title_fontsize=11, fontsize=10)
axes[0].grid(True, alpha=0.3, axis='y')
plt.setp(axes[0].xaxis.get_majorticklabels(), rotation=0)

# Stacked percentage bar chart
contingencyPct2 = contingencyTable2.T.div(contingencyTable2.T.sum(axis=1),␣
 ↪axis=0) * 100
contingencyPct2.plot(kind='bar', stacked=True, ax=axes[1],
                      color=['lightblue', 'lightgreen', 'lightcoral'])
axes[1].set_xlabel('Diet Quality', fontsize=12)
axes[1].set_ylabel('Percentage', fontsize=12)
axes[1].set_title('Exam Performance Distribution by Diet Quality (%)',␣
 ↪fontsize=14, fontweight='bold')
axes[1].legend(title='Exam Performance', title_fontsize=11, fontsize=10)
axes[1].grid(True, alpha=0.3, axis='y')
plt.setp(axes[1].xaxis.get_majorticklabels(), rotation=0)

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



1.7.3 5.8 Visualization of Chi-Squared Test Results

[427]:
```
# Create unified visualization for both chi-squared tests
fig, axes = plt.subplots(2, 2, figsize=(18, 12))

# Recreate contingency tables for visualization
contingencyTable1 = pd.crosstab(df['examPerformance'],␣
 ↪df['parental_education_level'])
contingencyTable2 = pd.crosstab(df['examPerformance'], df['diet_quality'])

# Calculate percentages
contingencyPct1 = contingencyTable1.T.div(contingencyTable1.T.sum(axis=1),␣
 ↪axis=0) * 100
contingencyPct2 = contingencyTable2.T.div(contingencyTable2.T.sum(axis=1),␣
 ↪axis=0) * 100

# Row 1: Parental Education Level
# Grouped bar chart
contingencyTable1.T.plot(kind='bar', ax=axes[0, 0],
                          color=['lightblue', 'lightgreen', 'lightcoral'])
axes[0, 0].set_xlabel('Parental Education Level', fontsize=12)
axes[0, 0].set_ylabel('Count', fontsize=12)
axes[0, 0].set_title('Exam Performance by Parental Education Level',
                      fontsize=13, fontweight='bold')
axes[0, 0].legend(title='Exam Performance', title_fontsize=10, fontsize=9)
axes[0, 0].grid(True, alpha=0.3, axis='y')
plt.setp(axes[0, 0].xaxis.get_majorticklabels(), rotation=45, ha='right')

# Stacked percentage bar chart
contingencyPct1.plot(kind='bar', stacked=True, ax=axes[0, 1],
                      color=['lightblue', 'lightgreen', 'lightcoral'])
axes[0, 1].set_xlabel('Parental Education Level', fontsize=12)
axes[0, 1].set_ylabel('Percentage', fontsize=12)
axes[0, 1].set_title('Exam Performance Distribution by Parental Education (%)',
                      fontsize=13, fontweight='bold')
axes[0, 1].legend(title='Exam Performance', title_fontsize=10, fontsize=9)
axes[0, 1].grid(True, alpha=0.3, axis='y')
plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45, ha='right')

# Row 2: Diet Quality
# Grouped bar chart
contingencyTable2.T.plot(kind='bar', ax=axes[1, 0],
                          color=['lightblue', 'lightgreen', 'lightcoral'])
axes[1, 0].set_xlabel('Diet Quality', fontsize=12)
axes[1, 0].set_ylabel('Count', fontsize=12)
axes[1, 0].set_title('Exam Performance by Diet Quality',
                      fontsize=13, fontweight='bold')
```

```python
axes[1, 0].legend(title='Exam Performance', title_fontsize=10, fontsize=9)
axes[1, 0].grid(True, alpha=0.3, axis='y')
plt.setp(axes[1, 0].xaxis.get_majorticklabels(), rotation=0)

# Stacked percentage bar chart
contingencyPct2.plot(kind='bar', stacked=True, ax=axes[1, 1],
                     color=['lightblue', 'lightgreen', 'lightcoral'])
axes[1, 1].set_xlabel('Diet Quality', fontsize=12)
axes[1, 1].set_ylabel('Percentage', fontsize=12)
axes[1, 1].set_title('Exam Performance Distribution by Diet Quality (%)',
                     fontsize=13, fontweight='bold')
axes[1, 1].legend(title='Exam Performance', title_fontsize=10, fontsize=9)
axes[1, 1].grid(True, alpha=0.3, axis='y')
plt.setp(axes[1, 1].xaxis.get_majorticklabels(), rotation=0)

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



1.8 6. Summary and Conclusions

[428]:
```
# Recalculate for summary
tStat1, pValue1 = ttest_ind(highStudyScores, lowStudyScores)
tStat2, pValue2 = ttest_ind(higherMentalScores, lowerMentalScores)

print("GROUP COMPARISON ANALYSIS SUMMARY\n")

print("T-TEST RESULTS (2 Groups)")

print("1. Study Hours Groups:")
print(f"   Low Study Hours: M = {lowStudyScores.mean():.2f}, SD =␣
 ↪{lowStudyScores.std():.2f}")
print(f"   High Study Hours: M = {highStudyScores.mean():.2f}, SD =␣
 ↪{highStudyScores.std():.2f}")
print(f"   t = {tStat1:.2f}, p = {pValue1:.4f}")
print(f"   Result: {'Significant' if pValue1 < 0.05 else 'Not Significant'}\n")

print("2. Mental Health Groups:")
print(f"   Lower Mental Health: M = {lowerMentalScores.mean():.2f}, SD =␣
 ↪{lowerMentalScores.std():.2f}")
print(f"   Higher Mental Health: M = {higherMentalScores.mean():.2f}, SD =␣
 ↪{higherMentalScores.std():.2f}")
print(f"   t = {tStat2:.2f}, p = {pValue2:.4f}")
print(f"   Result: {'Significant' if pValue2 < 0.05 else 'Not Significant'}\n")

print("ANOVA RESULTS (3 Groups)")

# Recalculate for summary
fStat1, pValue1 = f_oneway(lowStudy, medStudy, highStudy)
fStat2, pValue2 = f_oneway(lowMental, medMental, highMental)
fStat3, pValue3 = f_oneway(lowExercise, medExercise, highExercise)
fStat4, pValue4 = f_oneway(lowNetflix, medNetflix, highNetflix)
fStat5, pValue5 = f_oneway(poorDiet, fairDiet, goodDiet)

print("1. Study Hours Groups:")
print(f"   Low: M = {lowStudy.mean():.2f}, SD = {lowStudy.std():.2f}")
print(f"   Medium: M = {medStudy.mean():.2f}, SD = {medStudy.std():.2f}")
print(f"   High: M = {highStudy.mean():.2f}, SD = {highStudy.std():.2f}")
print(f"   F = {fStat1:.2f}, p = {pValue1:.4f}")
print(f"   Result: {'Significant' if pValue1 < 0.05 else 'Not Significant'}\n")

print("2. Mental Health Groups:")
print(f"   Low: M = {lowMental.mean():.2f}, SD = {lowMental.std():.2f}")
print(f"   Medium: M = {medMental.mean():.2f}, SD = {medMental.std():.2f}")
print(f"   High: M = {highMental.mean():.2f}, SD = {highMental.std():.2f}")
print(f"   F = {fStat2:.2f}, p = {pValue2:.4f}")
print(f"   Result: {'Significant' if pValue2 < 0.05 else 'Not Significant'}\n")

print("3. Exercise Frequency Groups:")
print(f"   Low: M = {lowExercise.mean():.2f}, SD = {lowExercise.std():.2f}")
print(f"   Medium: M = {medExercise.mean():.2f}, SD = {medExercise.std():.2f}")
print(f"   High: M = {highExercise.mean():.2f}, SD = {highExercise.std():.2f}")
print(f"   F = {fStat3:.2f}, p = {pValue3:.4f}")
print(f"   Result: {'Significant' if pValue3 < 0.05 else 'Not Significant'}\n")

print("4. Netflix Hours Groups:")
print(f"   Low: M = {lowNetflix.mean():.2f}, SD = {lowNetflix.std():.2f}")
print(f"   Medium: M = {medNetflix.mean():.2f}, SD = {medNetflix.std():.2f}")
print(f"   High: M = {highNetflix.mean():.2f}, SD = {highNetflix.std():.2f}")
print(f"   F = {fStat4:.2f}, p = {pValue4:.4f}")
print(f"   Result: {'Significant' if pValue4 < 0.05 else 'Not Significant'}\n")

print("5. Diet Quality Groups:")
print(f"   Poor: M = {poorDiet.mean():.2f}, SD = {poorDiet.std():.2f}")
print(f"   Fair: M = {fairDiet.mean():.2f}, SD = {fairDiet.std():.2f}")
print(f"   Good: M = {goodDiet.mean():.2f}, SD = {goodDiet.std():.2f}")
print(f"   F = {fStat5:.2f}, p = {pValue5:.4f}")
print(f"   Result: {'Significant' if pValue5 < 0.05 else 'Not Significant'}\n")

print("CHI-SQUARED TEST RESULTS (Categorical Associations)")

# Recalculate Chi-squared tests for summary
from scipy.stats import chi2_contingency

contingencyTable1 = pd.crosstab(df['examPerformance'],␣
 ↪df['parental_education_level'])
chi2Stat1, pValue1, dof1, _ = chi2_contingency(contingencyTable1)
n1 = contingencyTable1.sum().sum()
minDim1 = min(contingencyTable1.shape[0] - 1, contingencyTable1.shape[1] - 1)
cramersV1 = np.sqrt(chi2Stat1 / (n1 * minDim1))

contingencyTable2 = pd.crosstab(df['examPerformance'], df['diet_quality'])
chi2Stat2, pValue2, dof2, _ = chi2_contingency(contingencyTable2)
n2 = contingencyTable2.sum().sum()
minDim2 = min(contingencyTable2.shape[0] - 1, contingencyTable2.shape[1] - 1)
cramersV2 = np.sqrt(chi2Stat2 / (n2 * minDim2))

print("\n1. Exam Performance × Parental Education Level:")
print(f"    ² = {chi2Stat1:.2f}, df = {dof1}, p = {pValue1:.4f}")
print(f"   Cramér's V = {cramersV1:.4f}")
print(f"   Result: {'Significant' if pValue1 < 0.05 else 'Not Significant'}")

print("\n2. Exam Performance × Diet Quality:")
print(f"    ² = {chi2Stat2:.2f}, df = {dof2}, p = {pValue2:.4f}")
print(f"   Cramér's V = {cramersV2:.4f}")
print(f"   Result: {'Significant' if pValue2 < 0.05 else 'Not Significant'}")
```

```
GROUP COMPARISON ANALYSIS SUMMARY

T-TEST RESULTS (2 Groups)
1. Study Hours Groups:
   Low Study Hours: M = 57.81, SD = 13.28
   High Study Hours: M = 80.10, SD = 12.17
   t = 27.71, p = 0.0000
   Result: Significant

2. Mental Health Groups:
   Lower Mental Health: M = 64.03, SD = 16.39
   Higher Mental Health: M = 73.49, SD = 16.14
   t = 9.05, p = 0.0000
   Result: Significant

ANOVA RESULTS (3 Groups)
1. Study Hours Groups:
   Low: M = 53.45, SD = 12.40
   Medium: M = 69.64, SD = 10.15
   High: M = 84.52, SD = 11.38
   F = 614.67, p = 0.0000
   Result: Significant

2. Mental Health Groups:
   Low: M = 63.43, SD = 15.94
   Medium: M = 67.87, SD = 16.85
   High: M = 75.90, SD = 15.49
   F = 53.43, p = 0.0000
   Result: Significant

3. Exercise Frequency Groups:
   Low: M = 66.09, SD = 16.97
   Medium: M = 69.30, SD = 15.91
   High: M = 72.13, SD = 17.04
   F = 11.44, p = 0.0000
   Result: Significant

4. Netflix Hours Groups:
   Low: M = 73.14, SD = 15.93
   Medium: M = 69.40, SD = 16.61
   High: M = 66.51, SD = 17.45
   F = 13.07, p = 0.0000
   Result: Significant
```

```
5. Diet Quality Groups:
   Poor: M = 68.13, SD = 17.06
   Fair: M = 70.43, SD = 16.65
   Good: M = 69.37, SD = 17.07
   F = 1.27, p = 0.2824
   Result: Not Significant

CHI-SQUARED TEST RESULTS (Categorical Associations)

1. Exam Performance × Parental Education Level:
    ² = 4.16, df = 6, p = 0.6552
   Cramér's V = 0.0456
   Result: Not Significant

2. Exam Performance × Diet Quality:
    ² = 1.91, df = 4, p = 0.7528
   Cramér's V = 0.0309
   Result: Not Significant
```

erika_analysis

October 19, 2025

1 Regression Analyses

This section performs the linear regression on exam score and binary logistic regression on exam pass/fail.

[41]:
```
## libraries
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, confusion_matrix,␣
↪accuracy_score, roc_auc_score, precision_score, recall_score, f1_score
```

1.1 Data Cleaning

[42]:
```
## read in data
url = 'https://raw.githubusercontent.com/USD-AAI-500-Stats-Group-2/
↪Stats500FinalProject/main/data/student_habits_performance.csv'

student_habits = pd.read_csv(url)
student_habits.head(15)
```

[42]:
```
   student_id  age  gender  study_hours_per_day  social_media_hours  \
0       S1000   23  Female                  0.0                 1.2
1       S1001   20  Female                  6.9                 2.8
2       S1002   21    Male                  1.4                 3.1
3       S1003   23  Female                  1.0                 3.9
4       S1004   19  Female                  5.0                 4.4
5       S1005   24    Male                  7.2                 1.3
6       S1006   21  Female                  5.6                 1.5
7       S1007   21  Female                  4.3                 1.0
8       S1008   23  Female                  4.4                 2.2
9       S1009   18  Female                  4.8                 3.1
10      S1010   19  Female                  4.6                 3.7
```

```
11      S1011   23    Male                  3.9                2.4
12      S1012   19  Female                  3.7                2.1
13      S1013   19  Female                  3.4                2.7
14      S1014   24    Male                  2.4                1.5

    netflix_hours part_time_job  attendance_percentage  sleep_hours  \
0             1.1            No                   85.0          8.0
1             2.3            No                   97.3          4.6
2             1.3            No                   94.8          8.0
3             1.0            No                   71.0          9.2
4             0.5            No                   90.9          4.9
5             0.0            No                   82.9          7.4
6             1.4           Yes                   85.8          6.5
7             2.0           Yes                   77.7          4.6
8             1.7            No                  100.0          7.1
9             1.3            No                   95.4          7.5
10            0.8            No                   77.6          5.8
11            2.5            No                   71.7          7.9
12            0.4           Yes                   81.1          4.5
13            2.7            No                   89.3          4.7
14            0.7            No                   87.4          6.7

   diet_quality  exercise_frequency parental_education_level internet_quality  \
0          Fair                   6                   Master          Average
1          Good                   6              High School          Average
2          Poor                   1              High School             Poor
3          Poor                   4                   Master             Good
4          Fair                   3                   Master             Good
5          Fair                   1                   Master          Average
6          Good                   2                   Master             Poor
7          Fair                   0                 Bachelor          Average
8          Good                   3                 Bachelor             Good
9          Good                   5                 Bachelor             Good
10         Fair                   1                      NaN             Good
11         Fair                   2                 Bachelor          Average
12         Fair                   1                 Bachelor             Good
13         Fair                   4                 Bachelor             Good
14         Poor                   6                 Bachelor          Average

    mental_health_rating extracurricular_participation  exam_score
0                      8                           Yes        56.2
1                      8                            No       100.0
2                      1                            No        34.3
3                      1                           Yes        26.8
4                      1                            No        66.4
5                      4                            No       100.0
6                      4                            No        89.8
```

```
7                        8                             No        72.6
8                        1                             No        78.9
9                       10                            Yes       100.0
10                       3                             No        63.3
11                       1                             No        74.4
12                       9                             No        76.9
13                      10                             No        75.8
14                       9                             No        78.9
```

[43]:
```
## identify "None" coded as Nulls
nulls = pd.DataFrame(student_habits.isnull().sum(), columns=['Nulls'])
nulls = nulls[nulls['Nulls']>0]
nulls

## replace with 'None' - decided not to do this, and keep as NA
#student_habits['parental_education_level'] =␣
 ↪student_habits['parental_education_level'].fillna('None')
```

[43]:
```
                          Nulls
parental_education_level     91
```

[44]:
```
## assign ordered levels for ordinal variables

# gender
student_habits['gender'].value_counts()
student_habits['gender'] = pd.Categorical(student_habits['gender'],
                                          categories = ['Male', 'Female',␣
 ↪'Other'],
                                          ordered = False)

# part time job
student_habits['part_time_job'].value_counts()
student_habits['part_time_job'] = pd.
 ↪Categorical(student_habits['part_time_job'],
                                          categories = ['No', 'Yes'],
                                          ordered = False)

# diet quality
student_habits['diet_quality'].value_counts()
student_habits['diet_quality'] = pd.Categorical(student_habits['diet_quality'],
                                          categories = ['Poor', 'Fair', 'Good'],
                                          ordered = False)

# parental education level
student_habits['parental_education_level'].value_counts()
# "None" not considered
```

```
#student_habits['parental_education_level'] = pd.
 ↪Categorical(student_habits['parental_education_level'],
#                                         categories = ['None', 'High School',␣
 ↪'Bachelor', 'Master'],
#                                         ordered = False)
student_habits['parental_education_level'] = pd.
 ↪Categorical(student_habits['parental_education_level'],
                                          categories = ['High School',␣
 ↪'Bachelor', 'Master'],
                                          ordered = False)

# internet quality
student_habits['internet_quality'].value_counts()
student_habits['internet_quality'] = pd.
 ↪Categorical(student_habits['internet_quality'],
                                          categories = ['Poor', 'Average',␣
 ↪'Good'],
                                          ordered = False)

# extracurricular participation
student_habits['extracurricular_participation'].value_counts()
student_habits['extracurricular_participation'] = pd.
 ↪Categorical(student_habits['extracurricular_participation'],
                                          categories = ['No', 'Yes'],
                                          ordered = False)
```

1.2 Linear Regression Analysis

Predicting exam score

1.2.1 Train Model

[45]:
```
#### linear regression with all variables (except student_id)

## identify independent (x) and dependent (y) variables
x = student_habits[['age', 'gender', 'study_hours_per_day',␣
 ↪'social_media_hours', 'netflix_hours',
                    'part_time_job', 'attendance_percentage', 'sleep_hours',␣
 ↪'diet_quality', 'exercise_frequency',
                    'parental_education_level', 'internet_quality',␣
 ↪'mental_health_rating', 'extracurricular_participation']]
y = student_habits['exam_score']

## create dummy variables for categorical data
x = pd.get_dummies(x, drop_first = True)

## split into train/test sets
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
 ↪random_state = 33)

## add intercept
x_train = sm.add_constant(x_train)

## convert to float variable types
x_train = x_train.astype(float)
y_train = y_train.astype(float)

## train model
model_linear = sm.OLS(y_train, x_train).fit()

## view model
print(model_linear.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             exam_score   R-squared:                       0.901
Model:                            OLS   Adj. R-squared:                  0.899
Method:                 Least Squares   F-statistic:                     395.0
Date:                Thu, 16 Oct 2025   Prob (F-statistic):               0.00
Time:                        01:04:40   Log-Likelihood:                -2483.1
No. Observations:                 800   AIC:                             5004.
Df Residuals:                     781   BIC:                             5093.
Df Model:                          18
Covariance Type:            nonrobust
================================================================================
=====================
                                        coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
---------------------
const                                 8.2371      2.948      2.794      0.005
2.450      14.024
age                                  -0.0290      0.085     -0.343      0.732
-0.195       0.137
study_hours_per_day                   9.5015      0.131     72.709      0.000
9.245       9.758
social_media_hours                   -2.6451      0.165    -16.021      0.000
-2.969      -2.321
netflix_hours                        -2.3575      0.181    -13.030      0.000
-2.713      -2.002
attendance_percentage                 0.1414      0.021      6.859      0.000
0.101       0.182
sleep_hours                           1.9210      0.160     12.013      0.000
1.607       2.235
exercise_frequency                    1.4888      0.094     15.759      0.000
```

```
1.303       1.674
mental_health_rating                     1.9480      0.069     28.094      0.000
1.812       2.084
gender_Female                           -0.2616      0.398     -0.657      0.511
-1.043       0.520
gender_Other                             0.4110      0.961      0.428      0.669
-1.475       2.297
part_time_job_Yes                        0.2932      0.467      0.628      0.530
-0.623       1.209
diet_quality_Fair                        0.3715      0.547      0.680      0.497
-0.702       1.445
diet_quality_Good                       -0.2427      0.555     -0.438      0.662
-1.332       0.846
parental_education_level_Bachelor        0.0485      0.432      0.112      0.911
-0.799       0.896
parental_education_level_Master         -0.2143      0.552     -0.388      0.698
-1.297       0.869
internet_quality_Average                 0.1779      0.577      0.308      0.758
-0.954       1.310
internet_quality_Good                   -0.5399      0.568     -0.951      0.342
-1.654       0.575
extracurricular_participation_Yes       -0.3121      0.415     -0.752      0.452
-1.127       0.503
==============================================================================
Omnibus:                       11.149   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.004   Jarque-Bera (JB):               14.337
Skew:                          -0.163   Prob(JB):                     0.000770
Kurtosis:                       3.570   Cond. No.                     1.34e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.34e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1.2.2 Evaluate Linear Regression Goodness of Fit

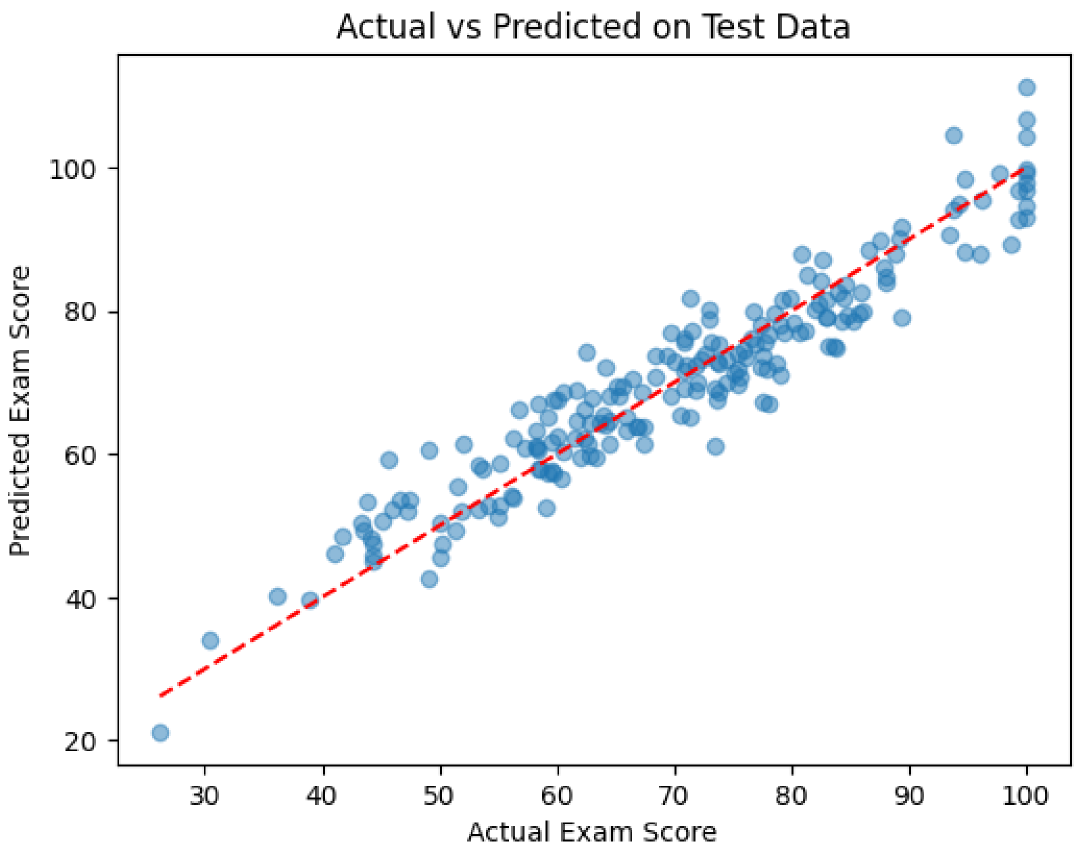
Use test (unseen) data

[46]:
```
### evaluate linear regression

## add constant column to test set
x_test = sm.add_constant(x_test)

## predict y based on x's
y_test_pred = model_linear.predict(x_test)
```

```
## view metrics
print('R^2 on test: %.4f' % r2_score(y_test, y_test_pred))
```

```
R^2 on test: 0.9032
```

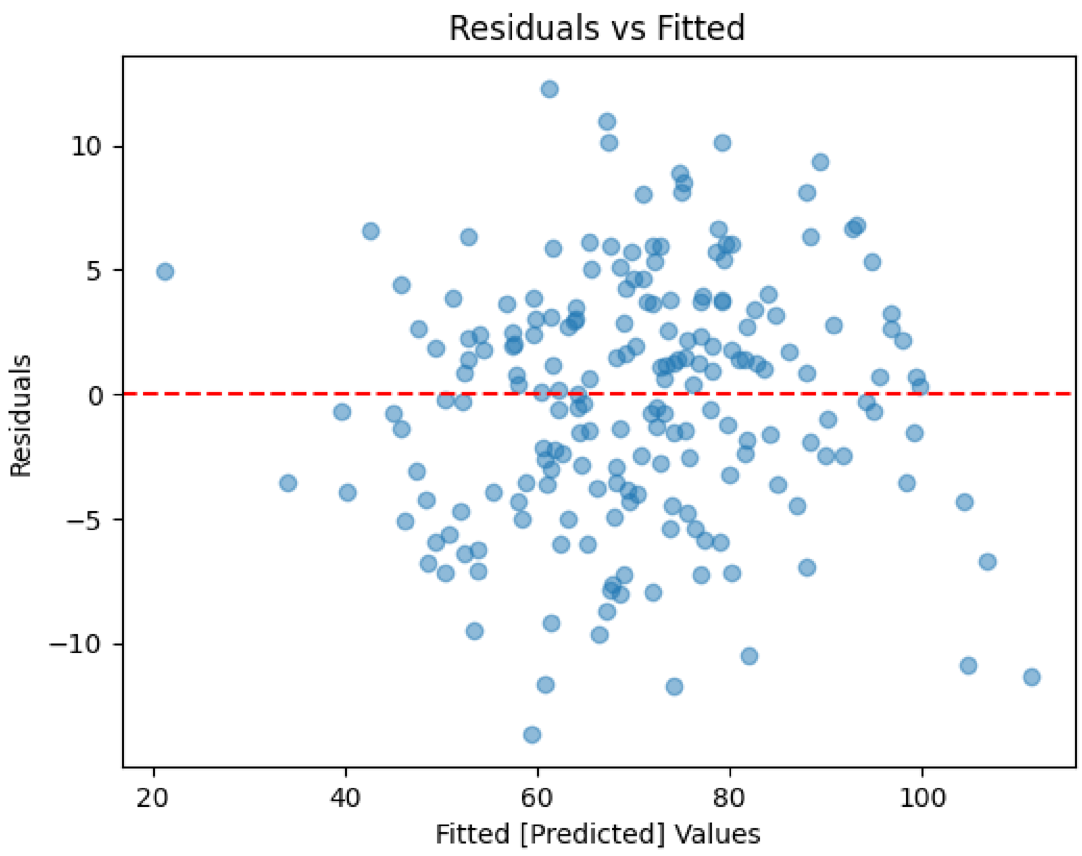
[47]:
```
## goodness of fit
plt.scatter(y_test, y_test_pred, alpha = 0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Exam Score')
plt.ylabel('Predicted Exam Score')
plt.title('Actual vs Predicted on Test Data')
plt.show()

# interpretation: points are close to red dashed line, which indicates good␣
↪predictions
```



1.2.3 Evaluate Linear Regression Model Assumptions

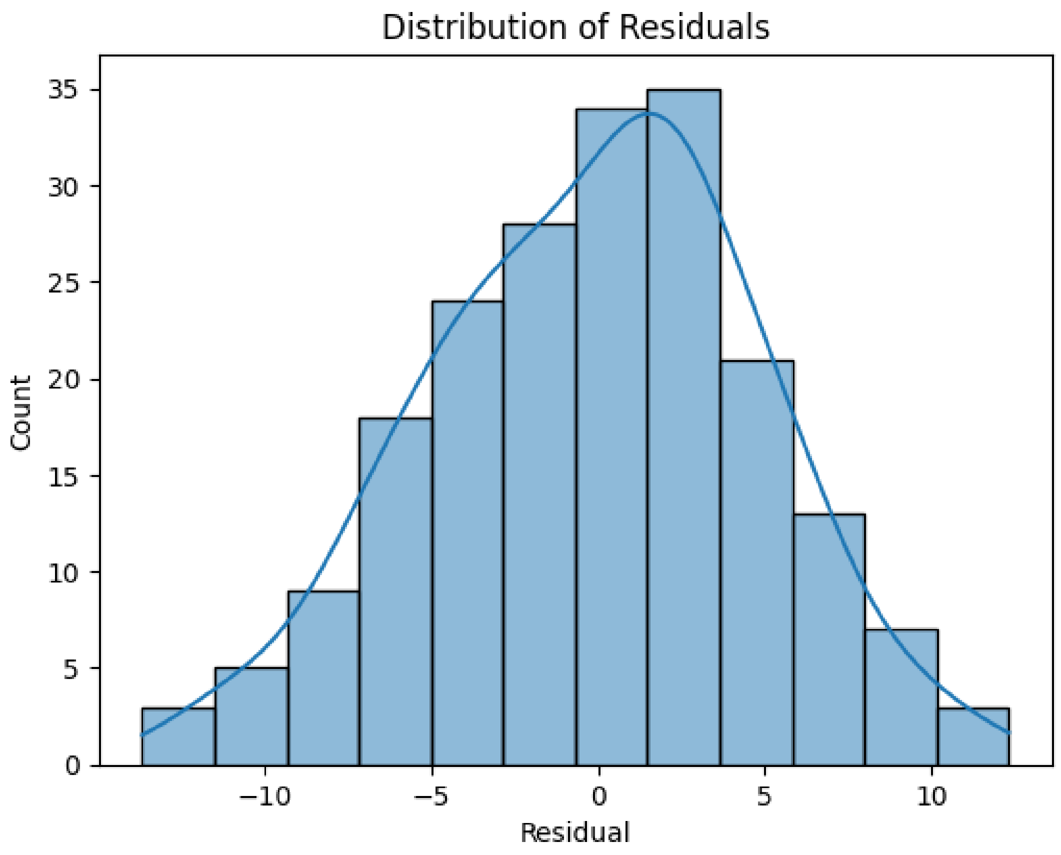
Use test (unseen) data

[48]:
```
## check for linearity and homoscedasticity
residuals = y_test - y_test_pred

plt.scatter(y_test_pred, residuals, alpha = 0.5)
plt.axhline(0, color = 'red', linestyle = '--')
plt.xlabel('Fitted [Predicted] Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted')
plt.show()

# interpretation: points appear to have random scatter around 0, no concerning␣
↪patterns
```



[49]:
```
## check for normality of residuals

sns.histplot(residuals, kde = True)
plt.title('Distribution of Residuals')
plt.xlabel('Residual')
plt.show()
```

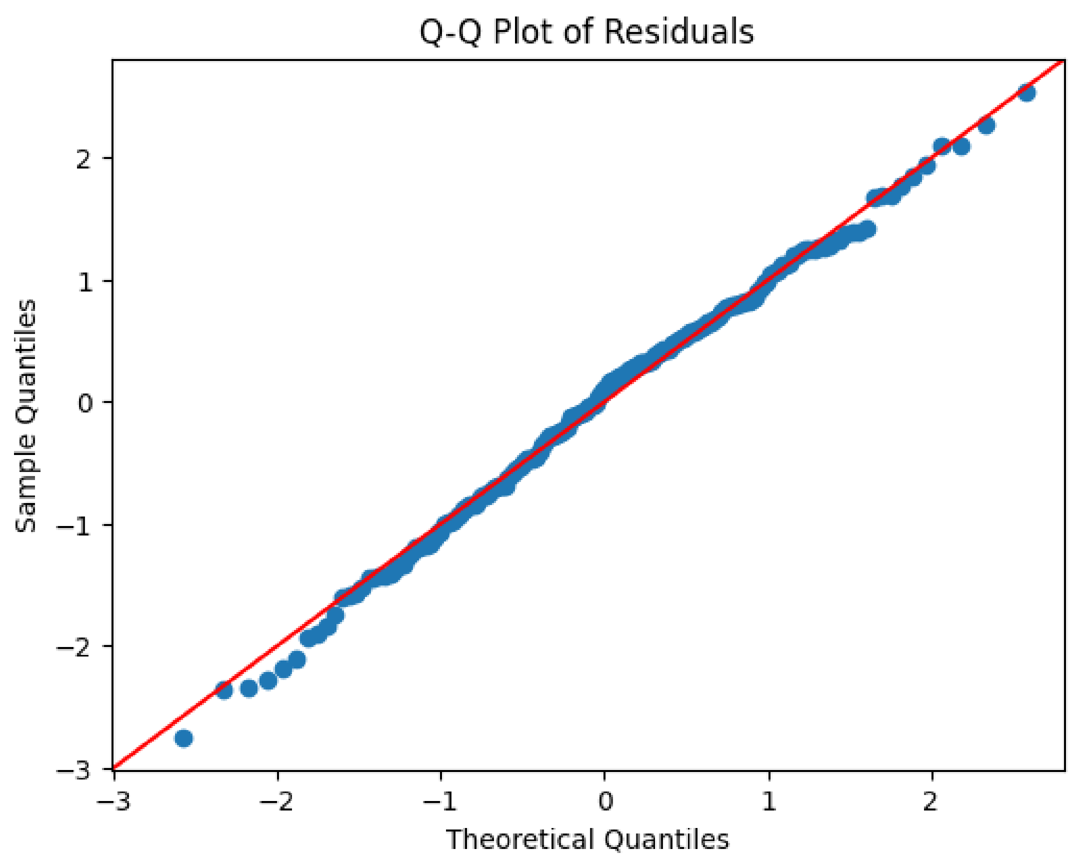
```
# interpretation: distribution looks roughly bell-shaped, hence residuals are␣
↪normal
```



[50]:
```
## check for normality of residuals using Shapiro-Wilk test
shapiro_test_residuals = stats.shapiro(residuals)
print(f'Shapiro-Wilk p-value = {shapiro_test_residuals.pvalue:.4f}')
```

```
Shapiro-Wilk p-value = 0.6547
```

[51]:
```
## Q-Q Plot for Residual Normality

residuals = pd.to_numeric(residuals, errors = 'coerce')
sm.qqplot(residuals, line = '45', fit = True)
plt.title('Q-Q Plot of Residuals')
plt.show()

# interpretation: points roughly follow the 45 degree line
```



1.3 Binary Logistic Regression Analysis

Predicting exam pass (1) or fail (0)

1.3.1 Data Preparation

[52]:
```
#### change exam_score into pass/fail binary variable

## threshold for pass/fail
pass_threshold = 70

## convert variable
student_habits['pass_fail'] = (pd.to_numeric(student_habits['exam_score'],␣
 ↪errors = 'coerce') >= pass_threshold).astype(int)

## view new variable
print(student_habits['pass_fail'].value_counts())

## remove NA data
student_habits = student_habits.dropna()
```

```
pass_fail
1    511
0    489
Name: count, dtype: int64
```

1.3.2 Train Model

[53]:
```
#### model

## split into train/test sets
train_data, test_data = train_test_split(student_habits, test_size = 0.2,␣
↪random_state = 33)

## train model
model_binary = smf.glm(
    formula = 'pass_fail ~ age + gender + study_hours_per_day +␣
↪social_media_hours + netflix_hours + \
              part_time_job + attendance_percentage + sleep_hours +␣
↪diet_quality + \
              exercise_frequency + parental_education_level + internet_quality␣
↪+ \
              mental_health_rating + extracurricular_participation',
    data = train_data,
    family = sm.families.Binomial()).fit()

print(model_binary.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              pass_fail   No. Observations:                  727
Model:                            GLM   Df Residuals:                      708
Model Family:                Binomial   Df Model:                           18
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -147.91
Date:                Thu, 16 Oct 2025   Deviance:                       295.81
Time:                        01:04:43   Pearson chi2:                     380.
No. Iterations:                     8   Pseudo R-squ. (CS):             0.6245
Covariance Type:            nonrobust
================================================================================
========================
                                           coef    std err          z      P>|z|
[0.025      0.975]
--------------------------------------------------------------------------------
------------------------
Intercept                              -23.7714      2.909     -8.172      0.000
-29.473     -18.070
gender[T.Female]                         0.1000      0.310      0.322      0.747
-0.508       0.708
```

```
gender[T.Other]                          -1.0071      0.678     -1.486      0.137
-2.336       0.322
part_time_job[T.Yes]                      0.0656      0.360      0.182      0.855
-0.639       0.771
diet_quality[T.Fair]                      0.3621      0.421      0.860      0.390
-0.463       1.187
diet_quality[T.Good]                     -0.0455      0.419     -0.109      0.914
-0.868       0.777
parental_education_level[T.Bachelor]      0.1871      0.330      0.567      0.571
-0.460       0.834
parental_education_level[T.Master]        0.1078      0.441      0.244      0.807
-0.757       0.973
internet_quality[T.Average]              -0.5625      0.440     -1.278      0.201
-1.425       0.300
internet_quality[T.Good]                 -0.3359      0.427     -0.787      0.431
-1.172       0.500
extracurricular_participation[T.Yes]     -0.1601      0.327     -0.490      0.624
-0.800       0.480
age                                      -0.0831      0.067     -1.241      0.215
-0.214       0.048
study_hours_per_day                       3.6929      0.342     10.806      0.000
3.023       4.363
social_media_hours                       -1.0014      0.158     -6.348      0.000
-1.311      -0.692
netflix_hours                            -0.9496      0.168     -5.669      0.000
-1.278      -0.621
attendance_percentage                     0.0671      0.016      4.202      0.000
0.036       0.098
sleep_hours                               0.7870      0.131      6.016      0.000
0.531       1.043
exercise_frequency                        0.6035      0.093      6.523      0.000
0.422       0.785
mental_health_rating                      0.8194      0.086      9.573      0.000
0.652       0.987
================================================================================
========================
```

1.3.3 Evaluate Binary Logistic Regression Model Prediction Accuracy

Use test (unseen) data

[54]:
```
#### assessing prediction accuracy on test data

## predict pass/fail
test_data['pred_prob'] = model_binary.predict(test_data)
test_data['pred_class'] = (test_data['pred_prob'] >= 0.5).astype(int)

## model performance
```

```
cm_test = confusion_matrix(test_data['pass_fail'], test_data['pred_class'])
accuracy_test = accuracy_score(test_data['pass_fail'], test_data['pred_class'])
auc_test = roc_auc_score(test_data['pass_fail'], test_data['pred_prob'])
precision_test = precision_score(test_data['pass_fail'],␣
 ↪test_data['pred_class'])
recall_test = recall_score(test_data['pass_fail'], test_data['pred_class'])
f1_test = f1_score(test_data['pass_fail'], test_data['pred_class'])
specificity = cm_test[0,0] / (cm_test[0,0] + cm_test[0,1])

## format confusion matrix
cm_test_clean = pd.DataFrame(cm_test,
                             index = ['Actual Fail (0)', 'Actual Pass (1)'],
                             columns = ['Predicted Fail (0)', 'Predicted Pass␣
 ↪(1)'])

# print results
print('Confusion Matrix:')
print(cm_test_clean)

print(f'\nAccuracy = {accuracy_test:.3f}')
print(f'Precision = {precision_test:.3f}')
print(f'Recall = {recall_test:.3f}')
print(f'Specificity = {specificity:.3f}')
print(f'F1 Score = {f1_test:.3f}')
print(f'AUC = {auc_test:.3f}')
```

```
Confusion Matrix:
                 Predicted Fail (0)  Predicted Pass (1)
Actual Fail (0)                  74                  11
Actual Pass (1)                  13                  84

Accuracy = 0.868
Precision = 0.884
Recall = 0.866
Specificity = 0.871
F1 Score = 0.875
AUC = 0.959
```

Paul_Ancajima_EDA_and_LinearModel

October 19, 2025

[28]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
from matplotlib import pyplot as plt
from scipy.stats import ttest_ind
```

1 1. Introduction

This notebook explores relationships between various student habits and how well they perform academically.

1.1 2. Data Loading and Initial Inspection

[29]:
```
student_habits = pd.read_csv("../data/student_habits_performance.csv")
student_habits.head()
```

[29]:
```
  student_id  age  gender  study_hours_per_day  social_media_hours  \
0      S1000   23  Female                  0.0                 1.2
1      S1001   20  Female                  6.9                 2.8
2      S1002   21    Male                  1.4                 3.1
3      S1003   23  Female                  1.0                 3.9
4      S1004   19  Female                  5.0                 4.4

   netflix_hours part_time_job  attendance_percentage  sleep_hours  \
0            1.1            No                   85.0          8.0
1            2.3            No                   97.3          4.6
2            1.3            No                   94.8          8.0
3            1.0            No                   71.0          9.2
4            0.5            No                   90.9          4.9

  diet_quality  exercise_frequency parental_education_level internet_quality  \
0        Fair                     6             Master        Average
1        Good                     6        High School        Average
2        Poor                     1        High School           Poor
3        Poor                     4             Master           Good
4        Fair                     3             Master           Good

   mental_health_rating extracurricular_participation  exam_score
0                     8                           Yes        56.2
1                     8                            No       100.0
2                     1                            No        34.3
3                     1                           Yes        26.8
4                     1                            No        66.4
```

[30]:
```
# We do not need to know their id
student_habits = student_habits.drop("student_id", axis=1)
student_habits.head()
```

[30]:
```
   age  gender  study_hours_per_day  social_media_hours  netflix_hours  \
0   23  Female                  0.0                 1.2            1.1
1   20  Female                  6.9                 2.8            2.3
2   21    Male                  1.4                 3.1            1.3
3   23  Female                  1.0                 3.9            1.0
4   19  Female                  5.0                 4.4            0.5

  part_time_job  attendance_percentage  sleep_hours diet_quality  \
0            No                   85.0          8.0         Fair
1            No                   97.3          4.6         Good
2            No                   94.8          8.0         Poor
3            No                   71.0          9.2         Poor
4            No                   90.9          4.9         Fair

   exercise_frequency parental_education_level internet_quality  \
0                   6                   Master          Average
1                   6              High School          Average
2                   1              High School             Poor
3                   4                   Master             Good
4                   3                   Master             Good

   mental_health_rating extracurricular_participation  exam_score
0                     8                           Yes        56.2
1                     8                            No       100.0
2                     1                            No        34.3
3                     1                           Yes        26.8
4                     1                            No        66.4
```

We have a mixute of numeric and non-numeric columns. Lets take a look at each.

[31]:
```
# Analyze numeric columns
student_habits.describe()
```

[31]:
```
              age  study_hours_per_day  social_media_hours  netflix_hours  \
count  1000.0000           1000.00000         1000.000000    1000.000000
mean     20.4980              3.55010            2.505500       1.819700
std       2.3081              1.46889            1.172422       1.075118
min      17.0000              0.00000            0.000000       0.000000
25%      18.7500              2.60000            1.700000       1.000000
50%      20.0000              3.50000            2.500000       1.800000
75%      23.0000              4.50000            3.300000       2.525000
max      24.0000              8.30000            7.200000       5.400000

       attendance_percentage  sleep_hours  exercise_frequency  \
count            1000.000000  1000.000000         1000.000000
mean               84.131700     6.470100            3.042000
std                 9.399246     1.226377            2.025423
min                56.000000     3.200000            0.000000
25%                78.000000     5.600000            1.000000
50%                84.400000     6.500000            3.000000
75%                91.025000     7.300000            5.000000
max               100.000000    10.000000            6.000000

       mental_health_rating   exam_score
count           1000.000000  1000.000000
mean               5.438000    69.601500
std                2.847501    16.888564
min                1.000000    18.400000
25%                3.000000    58.475000
50%                5.000000    70.500000
75%                8.000000    81.325000
max               10.000000   100.000000
```

[32]:
```
skew = student_habits['exam_score'].skew()
print(f'{skew:.3f}')
```

```
-0.156
```

The exam score standard deviation is relatively high, and two standard deviations above the mean (103.4) exceed the maximum possible score of 100. The max score being capped causes the distribution to not appear perfectly normal and slighly left-skewed with more students clustered near the upper limit of 100.

[33]:
```
exam_score = student_habits['exam_score']
mean = exam_score.mean()
se = exam_score.std(ddof=1) / np.sqrt(len(exam_score))
ci = stats.t.interval(0.95, df=len(exam_score)-1, loc=mean, scale=se)
print(f"95% CI for mean exam score: {ci}")
```

```
95% CI for mean exam score: (np.float64(68.55348547489275),
np.float64(70.64951452510725))
```

Confidence interval shows between 68.56-70.65, so we can say we are 95% confident the true mean of the population lies between these values.

[34]:
```
non_numeric = student_habits.select_dtypes(exclude=['number'])
non_numeric.head()
```

[34]:
```
   gender part_time_job diet_quality parental_education_level  \
0  Female            No         Fair                   Master
1  Female            No         Good              High School
2    Male            No         Poor              High School
3  Female            No         Poor                   Master
4  Female            No         Fair                   Master

  internet_quality extracurricular_participation
0          Average                           Yes
1          Average                            No
2             Poor                            No
3             Good                           Yes
4             Good                            No
```

[35]:
```
for feature in non_numeric:
    print(feature, ":", non_numeric[feature].unique())
```

```
gender : ['Female' 'Male' 'Other']
part_time_job : ['No' 'Yes']
diet_quality : ['Fair' 'Good' 'Poor']
parental_education_level : ['Master' 'High School' 'Bachelor' nan]
internet_quality : ['Average' 'Poor' 'Good']
extracurricular_participation : ['Yes' 'No']
```

[36]:
```
student_habits.isnull().sum()
```

[36]:
```
age                               0
gender                            0
study_hours_per_day               0
social_media_hours                0
netflix_hours                     0
part_time_job                     0
attendance_percentage             0
sleep_hours                       0
diet_quality                      0
exercise_frequency                0
parental_education_level         91
internet_quality                  0
mental_health_rating              0
extracurricular_participation     0
```

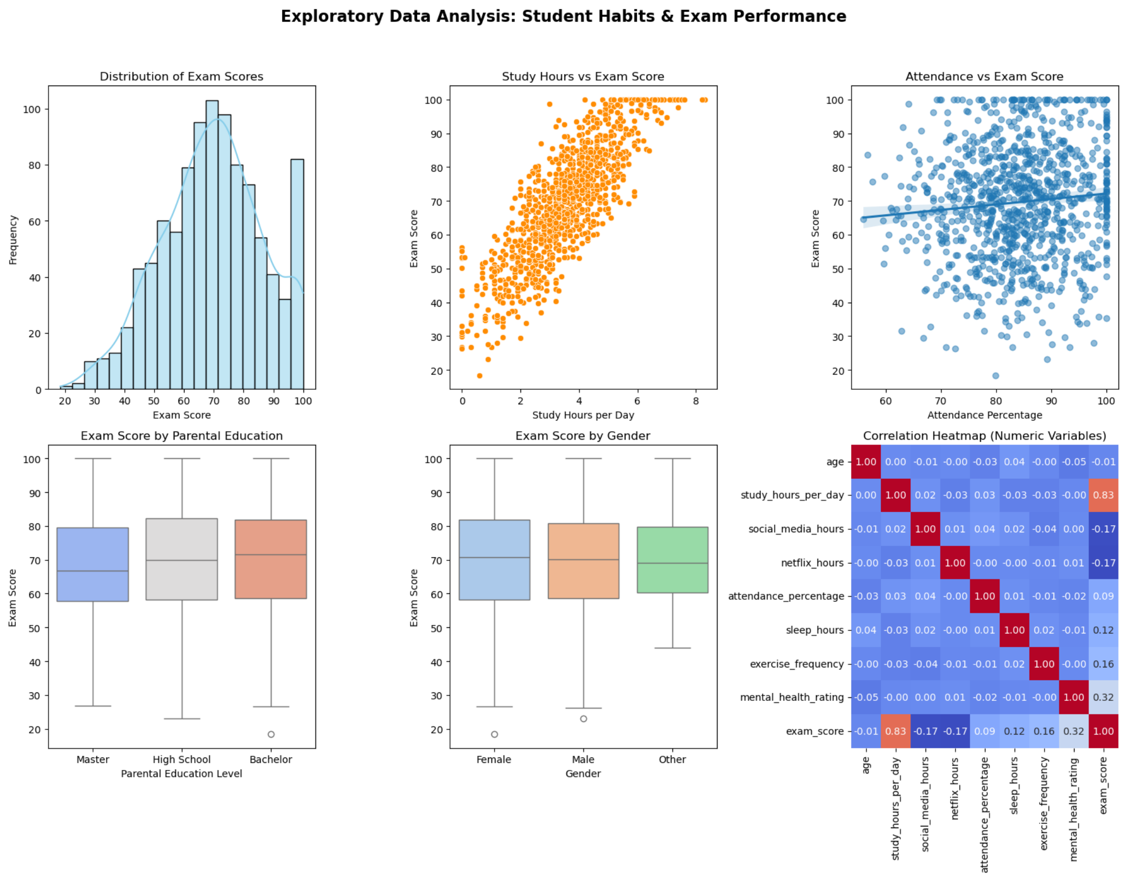
```
exam_score                        0
dtype: int64
```

We have 91 NaN values in parental_education_level, we can dive further into that later.

1.2 3. Exploratory Data Analysis (EDA)

[37]:
```python
# Ensure numeric data are properly selected
num_df = student_habits.select_dtypes(include=['number'])

plt.figure(figsize=(16, 12))
plt.subplots_adjust(hspace=0.4, wspace=0.3)

# Histogram of exam scores
plt.subplot(2, 3, 1)
sns.histplot(student_habits['exam_score'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of Exam Scores')
plt.xlabel('Exam Score')
plt.ylabel('Frequency')

# Study hours vs Exam score (scatter)
plt.subplot(2, 3, 2)
sns.scatterplot(x='study_hours_per_day', y='exam_score', data=student_habits,␣
 ↪color='darkorange')
plt.title('Study Hours vs Exam Score')
plt.xlabel('Study Hours per Day')
plt.ylabel('Exam Score')

# Attendance vs Exam score (regression line)
plt.subplot(2, 3, 3)
sns.regplot(x='attendance_percentage', y='exam_score', data=student_habits,␣
 ↪scatter_kws={'alpha':0.5})
plt.title('Attendance vs Exam Score')
plt.xlabel('Attendance Percentage')
plt.ylabel('Exam Score')

# Exam score by Parental Education (boxplot)
plt.subplot(2, 3, 4)
sns.boxplot(
    x='parental_education_level', y='exam_score',
    hue='parental_education_level', data=student_habits,
    palette='coolwarm', legend=False
)
plt.title('Exam Score by Parental Education')
plt.xlabel('Parental Education Level')
plt.ylabel('Exam Score')
```

```python
# Exam score by Gender (boxplot)
plt.subplot(2, 3, 5)
sns.boxplot(
    x='gender', y='exam_score',
    hue='gender', data=student_habits,
    palette='pastel', legend=False
)
plt.title('Exam Score by Gender')
plt.xlabel('Gender')
plt.ylabel('Exam Score')

# Correlation heatmap
plt.subplot(2, 3, 6)
sns.heatmap(num_df.corr(), annot=True, cmap='coolwarm', fmt='.2f', cbar=False)
plt.title('Correlation Heatmap (Numeric Variables)')

plt.suptitle('Exploratory Data Analysis: Student Habits & Exam Performance',␣
 ↪fontsize=16, fontweight='bold', y=1.03)
plt.tight_layout()
plt.show()
```



The histogram of exam scores shows a mostly normal distribution with a slight left skew, likely due to a concentration of high-achieving students. The scatterplot of Study Hours vs. Exam Score displays a clear positive linear trend, showing students who study more tend to perform better. Other visualizations suggest that some features have little impact on exam performance, and the correlation heatmap supports these observations by highlighting only a couple meaningful relationships among the variables.

1.2.1 Significance Testing

Handling Missing Values (Parental Education Level)

The dataset contains NaN values in the parental_education_level column. To determine whether these missing entries meaningfully affect student performance, we perform a t-test comparing exam scores between students with known and missing parental education data.

If the results show no significant difference, then it suggests that the missing data has little impact on exam outcomes. In that case, we can safely treat missing values as a separate category — for simplicity, we'll label them as "Below High School" to maintain completeness in the dataset.

[38]:
```
# Education dictionary
edu_order = {
    "Below High School": 0,
    "High School": 1,
    "Bachelor": 2,
    "Master": 3
}

# new column mapping education levels to their respective numerical values
student_habits["parental_education_level_num"] = (
    student_habits["parental_education_level"]
    .map(edu_order)
    .fillna(0)  # fill NaNs or unmapped with 0
)

# H_0: mean = mean
# H_1: mean != mean

# t-test whether parental education below highschool level affects exam score
low = student_habits[student_habits['parental_education_level_num'] ==␣
 ↪0]['exam_score']
high = student_habits[student_habits['parental_education_level'] !=␣
 ↪0]['exam_score']
t_stat, p_val = ttest_ind(low, high, equal_var=False)

print(f"T-statistic: {t_stat:.3f}, p-value: {p_val:.4f}")
```

```
T-statistic: 0.238, p-value: 0.8122
```

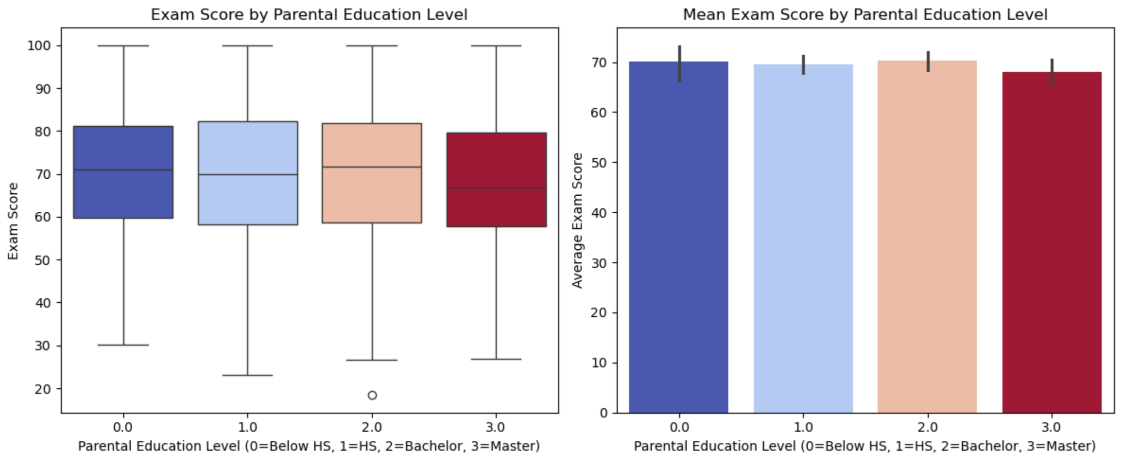
The t-statistic of 0.238 indicates that the means of the final exam scores differ by only about a quarter of a standard error, which is essentially negligible. The p-value (0.812) is well above any conventional significance level (alpha=0.05), so we fail to reject the null hypothesis.

This suggests that whether a student's parental education level is "Below High School" or any arbitrary label makes no meaningful difference in their final exam score.

[39]:
```
student_habits[['parental_education_level_num', 'exam_score']].describe()
```

[39]:
```
       parental_education_level_num   exam_score
count                   1000.000000  1000.000000
mean                       1.593000    69.601500
std                        0.870695    16.888564
min                        0.000000    18.400000
25%                        1.000000    58.475000
50%                        2.000000    70.500000
75%                        2.000000    81.325000
max                        3.000000   100.000000
```

[40]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.boxplot(
    x='parental_education_level_num',
    y='exam_score',
    hue='parental_education_level_num',  #  added
    data=student_habits,
    palette='coolwarm',
    legend=False
)
plt.title('Exam Score by Parental Education Level')
plt.xlabel('Parental Education Level (0=Below HS, 1=HS, 2=Bachelor, 3=Master)')
plt.ylabel('Exam Score')

plt.subplot(1, 2, 2)
sns.barplot(
    x='parental_education_level_num',
    y='exam_score',
    hue='parental_education_level_num',  #  added
    data=student_habits,
    estimator='mean',
    errorbar=('ci', 95),
    palette='coolwarm',
    legend=False
)
plt.title('Mean Exam Score by Parental Education Level')
plt.xlabel('Parental Education Level (0=Below HS, 1=HS, 2=Bachelor, 3=Master)')
plt.ylabel('Average Exam Score')

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



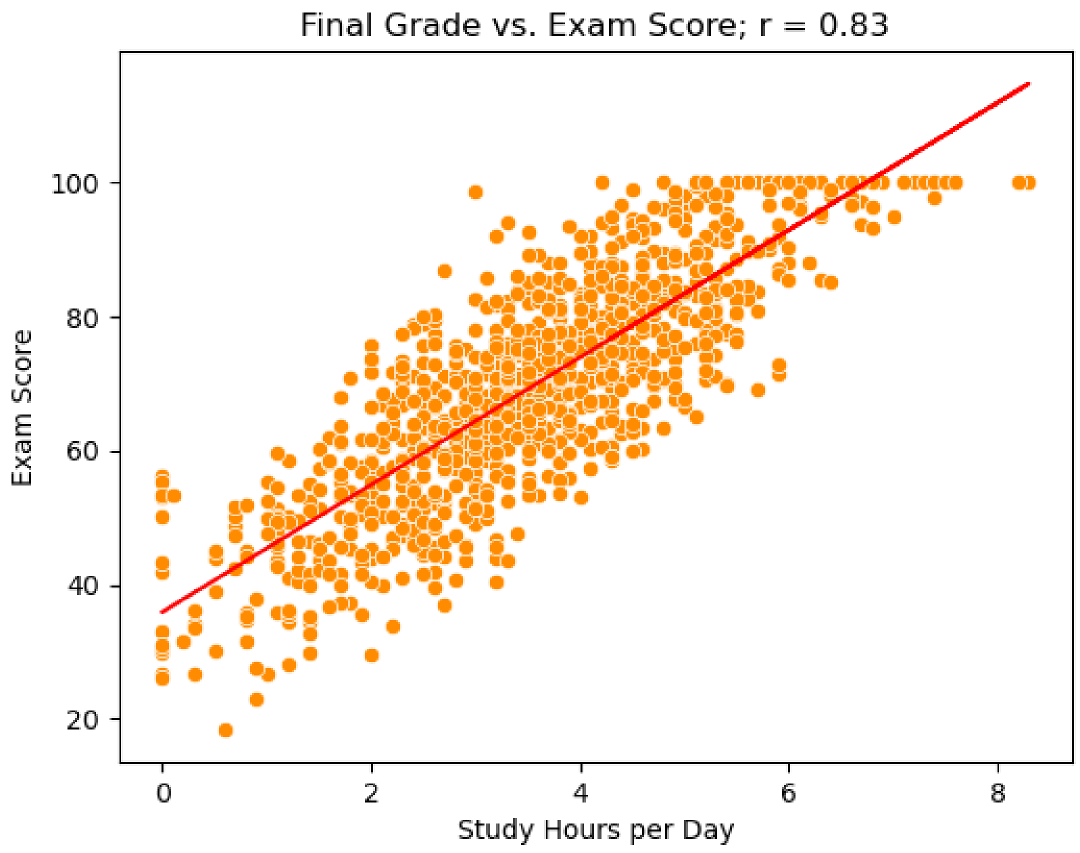
The visualizations of exam scores by parental education level support the results of the t-tests. The bar and box plots show that average exam scores are nearly identical across education levels, with overlapping confidence intervals. This aligns with the t-test results, which produced a high p-value (0.81), indicating no statistically significant difference in mean exam scores between students with missing or differing parental education levels. In other words, the data visualizations and inferential tests both suggest that parental education has little to no measurable effect on exam performance in this sample.

1.3 4. Regression Analysis

We seen in an earlier scatter plot, there is an obvious linear relationship with hours of study, Lets dive a bit deeper down at this trend.

[41]:
```
x = student_habits['study_hours_per_day']
y = student_habits['exam_score']

x_bar = x.mean()
y_bar = y.mean()

sns.scatterplot(x='study_hours_per_day', y='exam_score', data=student_habits,␣
 ↪color='darkorange')
plt.title('Study Hours vs Exam Score')
plt.xlabel('Study Hours per Day')
plt.ylabel('Exam Score')
```

```
# create best-fit line based on slope-intercept form
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x+b, color = 'red')

# correlation coefficient
corr = round(np.corrcoef(x, y)[0,1],2)
plt.title('Final Grade vs. Exam Score;' ' r = ' + "{:.2f}".format(corr))
plt.show()
```



We can visually see a line fit nicely through the scatter points and notice that r value of 0.83. Given 1 being a perfect linear relationship, we can confirm see that as Study Hours increase so will the final Exam Score.

We can manually calculate the R-squared value using 'Sum of Squares' to do some sanity checking

```
[42]: def compute_r_r2(x, y, round_to=3):
          x = np.asarray(x)
          y = np.asarray(y)
```

```
    # Fit line (slope and intercept)
    m, b = np.polyfit(x, y, 1)

    # Predicted values
    y_pred = m * x + b
    y_bar = y.mean()

    # Sum of squares
    SS_res = np.sum((y - y_pred)**2)
    SS_tot = np.sum((y - y_bar)**2)

    # R² and r
    R2 = 1 - (SS_res / SS_tot)
    r = np.sqrt(R2) if m > 0 else -np.sqrt(R2)  # sign matches slope

    return {
        'r': round(r, round_to),
        'R2': round(R2, round_to)
    }

compute_r_r2(x, y, round_to=2)
```

```
[42]: {'r': np.float64(0.83), 'R2': np.float64(0.68)}
```

```
[43]: res =[]
for feature in student_habits.select_dtypes(include='number').columns.
 ↪drop('exam_score'):
    if feature != 'exam_score':  # skip the target
        res.append((feature, compute_r_r2(student_habits[feature],␣
 ↪student_habits['exam_score'], round_to=2)))

res.sort(key=lambda x: x[1]['r'], reverse=True)
res
```

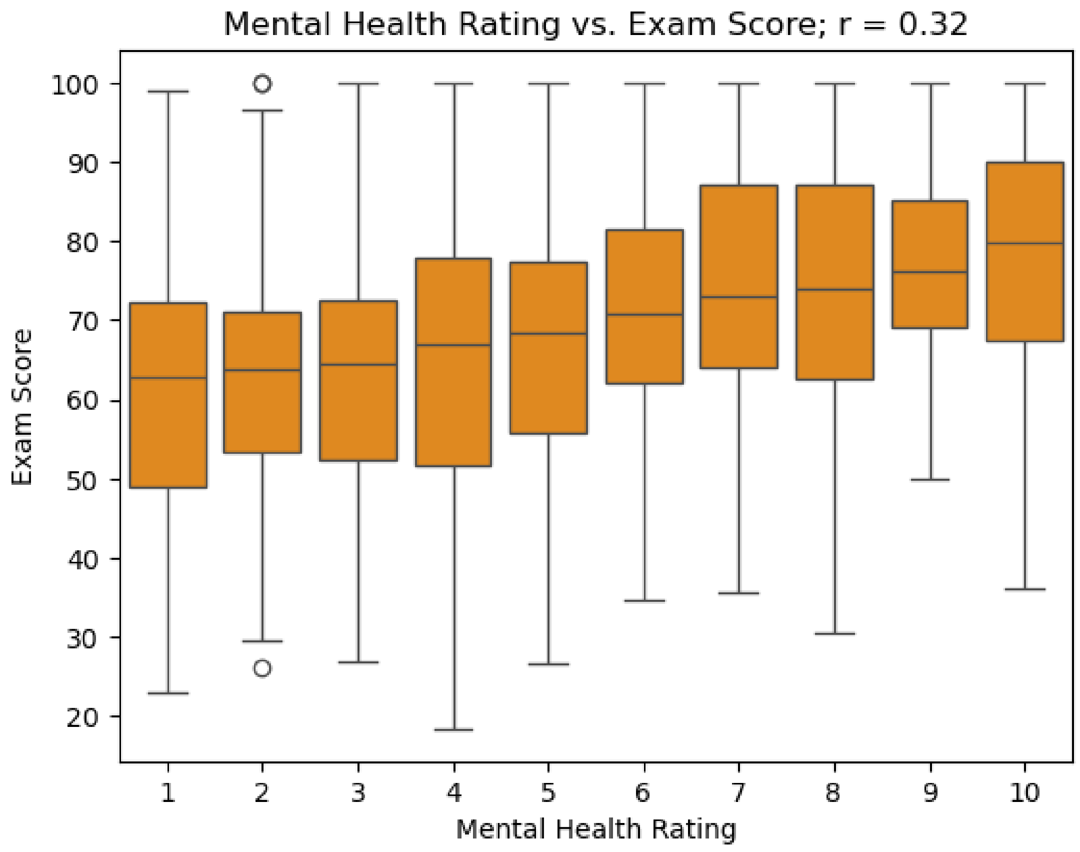
```
[43]: [('study_hours_per_day', {'r': np.float64(0.83), 'R2': np.float64(0.68)}),
 ('mental_health_rating', {'r': np.float64(0.32), 'R2': np.float64(0.1)}),
 ('exercise_frequency', {'r': np.float64(0.16), 'R2': np.float64(0.03)}),
 ('sleep_hours', {'r': np.float64(0.12), 'R2': np.float64(0.01)}),
 ('attendance_percentage', {'r': np.float64(0.09), 'R2': np.float64(0.01)}),
 ('age', {'r': np.float64(-0.01), 'R2': np.float64(0.0)}),
 ('parental_education_level_num',
  {'r': np.float64(-0.02), 'R2': np.float64(0.0)}),
 ('social_media_hours', {'r': np.float64(-0.17), 'R2': np.float64(0.03)}),
 ('netflix_hours', {'r': np.float64(-0.17), 'R2': np.float64(0.03)})]
```

Looking good, both values are 0.83. The second highest correlation according to above is the mental_health_rating. Lets visually look at that as well.

```
[44]: student_habits['mental_health_rating'].describe()
```

```
[44]: count    1000.000000
      mean        5.438000
      std         2.847501
      min         1.000000
      25%         3.000000
      50%         5.000000
      75%         8.000000
      max        10.000000
      Name: mental_health_rating, dtype: float64
```

```
[45]: x = student_habits['mental_health_rating']
      y = student_habits['exam_score']

      x_bar = x.mean()
      y_bar = y.mean()

      sns.boxplot(x='mental_health_rating', y='exam_score', data=student_habits,␣
       ↪color='darkorange')
      plt.title('Mental Health Rating vs Exam Score')
      plt.xlabel('Mental Health Rating')
      plt.ylabel('Exam Score')

      corr = round(np.corrcoef(x, y)[0,1],2)
      plt.title('Mental Health Rating vs. Exam Score;' ' r = ' + "{:.2f}".
       ↪format(corr))
      plt.show()
```



```
[46]: compute_r_r2(x, y, round_to=2)
```

```
[46]: {'r': np.float64(0.32), 'R2': np.float64(0.1)}
```

The results remain consistent, showing a slight linear relationship visually. Since correlations couldn't be computed for categorical features, we encoded them as numeric values. With this step complete, we can now move on to feature engineering and model preparation.

1.4 5. Feature Engineering

We can now remove a few extra columns created in earlier steps. The parental_education_level column contains 91 missing values, but previous t-tests confirmed that these have no significant effect on the target variable (exam_score). Therefore, dropping these rows or imputing averages is unnecessary, as the feature itself is not a strong predictor. For completeness, we'll retain all features during initial model training, using the previously created parental_education_level_num, and apply encoding to explore their overall impact.

```
[47]: non_numeric.head()
```

[47]:
```
   gender part_time_job diet_quality parental_education_level  \
0  Female            No         Fair                   Master
1  Female            No         Good              High School
2    Male            No         Poor              High School
3  Female            No         Poor                   Master
4  Female            No         Fair                   Master

  internet_quality extracurricular_participation
0          Average                           Yes
1          Average                            No
2             Poor                            No
3             Good                           Yes
4             Good                            No
```

[48]:
```python
diet_order = {
    "Poor": 0,
    "Fair": 1,
    "Good": 2
}

nominal_cols = [
    'gender',
    'part_time_job',
    'internet_quality',
    'extracurricular_participation'
]

encoded = student_habits.copy()
encoded['parental_education_level'] = encoded['parental_education_level'].
 ↪map(edu_order)
encoded['diet_quality'] = encoded['diet_quality'].map(diet_order)

encoded = pd.get_dummies(
    encoded,
    columns=nominal_cols,
    drop_first=True
)

encoded.head()
```

[48]:
```
   age  study_hours_per_day  social_media_hours  netflix_hours  \
0   23                  0.0                 1.2            1.1
1   20                  6.9                 2.8            2.3
2   21                  1.4                 3.1            1.3
3   23                  1.0                 3.9            1.0
4   19                  5.0                 4.4            0.5
```

```
   attendance_percentage  sleep_hours  diet_quality  exercise_frequency  \
0                   85.0          8.0             1                   6
1                   97.3          4.6             2                   6
2                   94.8          8.0             0                   1
3                   71.0          9.2             0                   4
4                   90.9          4.9             1                   3

   parental_education_level  mental_health_rating  exam_score  \
0                       3.0                     8        56.2
1                       1.0                     8       100.0
2                       1.0                     1        34.3
3                       3.0                     1        26.8
4                       3.0                     1        66.4

   parental_education_level_num  gender_Male  gender_Other  part_time_job_Yes  \
0                           3.0        False         False              False
1                           1.0        False         False              False
2                           1.0         True         False              False
3                           3.0        False         False              False
4                           3.0        False         False              False

   internet_quality_Good  internet_quality_Poor  \
0                  False                  False
1                  False                  False
2                  False                   True
3                   True                  False
4                   True                  False

   extracurricular_participation_Yes
0                               True
1                              False
2                              False
3                               True
4                              False
```

[49]:
```
encoded.drop(['parental_education_level'], axis=1, inplace=True)
encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   age                                1000 non-null   int64
 1   study_hours_per_day                1000 non-null   float64
 2   social_media_hours                 1000 non-null   float64
 3   netflix_hours                      1000 non-null   float64
 4   attendance_percentage              1000 non-null   float64
 5   sleep_hours                        1000 non-null   float64
 6   diet_quality                       1000 non-null   int64
 7   exercise_frequency                 1000 non-null   int64
 8   mental_health_rating               1000 non-null   int64
 9   exam_score                         1000 non-null   float64
 10  parental_education_level_num       1000 non-null   float64
 11  gender_Male                        1000 non-null   bool
 12  gender_Other                       1000 non-null   bool
 13  part_time_job_Yes                  1000 non-null   bool
 14  internet_quality_Good              1000 non-null   bool
 15  internet_quality_Poor              1000 non-null   bool
 16  extracurricular_participation_Yes  1000 non-null   bool
dtypes: bool(6), float64(7), int64(4)
memory usage: 91.9 KB
```

[50]:
```
encoded.isnull().sum()
```

[50]:
```
age                                  0
study_hours_per_day                  0
social_media_hours                   0
netflix_hours                        0
attendance_percentage                0
sleep_hours                          0
diet_quality                         0
exercise_frequency                   0
mental_health_rating                 0
exam_score                           0
parental_education_level_num         0
gender_Male                          0
gender_Other                         0
part_time_job_Yes                    0
internet_quality_Good                0
internet_quality_Poor                0
extracurricular_participation_Yes    0
dtype: int64
```

1.5 6. Model Training and Evaluation

Linear regression was chosen as the primary modeling approach because the target variable, exam score, is continuous and approximately normally distributed. This makes linear regression an appropriate and interpretable baseline for assessing how each predictor contributes to academic performance. Unlike tree-based or other non-parametric models, linear regression provides direct coefficient estimates that quantify the direction and magnitude of influence for each variable (e.g., how much exam score changes with an additional study hour).

Since the objective was to understand which factors most strongly influence performance rather than to maximize predictive power, linear regression provided the clear-

est and most statistically grounded framework.

[51]:
```python
X = encoded.drop('exam_score', axis=1)
y = encoded['exam_score']

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"Linear Regression  R² = {r2:.3f},  Mean Squared Error = {rmse:.2f}")

# Make a data frame with columns and coeeficients
coef_df = (
    pd.DataFrame({
        'Feature': X.columns,
        'Coefficient': model.coef_
    })
    .sort_values(by='Coefficient', ascending=False)
    .reset_index(drop=True)
)

print("\nTop 10 influential features:")
print(coef_df.head(10))
```

```
Linear Regression  R² = 0.898,  Mean Squared Error = 5.32

Top 10 influential features:
                        Feature  Coefficient
0            study_hours_per_day     9.601181
1                    sleep_hours     2.006770
2           mental_health_rating     1.943268
3             exercise_frequency     1.279372
4                   gender_Other     0.715778
5                    gender_Male     0.298347
6   parental_education_level_num     0.205816
7          attendance_percentage     0.147739
8                            age     0.068623
9               part_time_job_Yes    0.028203
```

The model's performance was evaluated using the coefficient of determination (R^2) and the root mean squared error (RMSE).

• R² = 0.898 indicates that approximately 89.8% of the variation in exam scores can be explained by the set of predictors included in the model. This represents an excellent fit, showing that the model captures nearly all the meaningful variability in student performance.

• RMSE = 5.32 means that, on average, the model's predictions deviate from the actual exam scores by about 5.3 points.

Together, these metrics demonstrate that the model is both highly accurate and well-calibrated, with minimal unexplained variance remaining.

1.6 7. Additional investigation

Although mental_health_rating initially appeared to have a more direct impact on exam performance, our regression model reveals that sleep_hours has a slightly higher coefficient. This suggests that adequate rest may indirectly enhance performance through improved mental well-being.

Meanwhile, study_hours_per_day stands out as the strongest independent predictor — it shows both a high correlation (r 0.83) and a large coefficient (~9.6), meaning each additional hour of study increases the predicted exam score by roughly 9.6 points.

Next, we will check for multicollinearity to for the top three predictors (study_hours, sleep_hours, and mental_health_rating) overlap in the variance they explain — that is, whether they are nearly redundant explanatory variables.

[52]:
```
encoded['sleep_hours'].describe()
```

[52]:
```
count    1000.000000
mean        6.470100
std         1.226377
min         3.200000
25%         5.600000
50%         6.500000
75%         7.300000
max        10.000000
Name: sleep_hours, dtype: float64
```

[53]:
```
_X = encoded[['study_hours_per_day', 'sleep_hours', 'mental_health_rating']]
vif_values = [variance_inflation_factor(_X.values, i) for i in range(_X.
 ↪shape[1])]

for feature, vif in zip(_X.columns, vif_values):
    print(f"{feature:25s} VIF = {vif:.2f}")
```

```
study_hours_per_day       VIF = 5.70
sleep_hours               VIF = 7.75
mental_health_rating      VIF = 4.20
```

Interpretation: The VIF scores indicate that study__hours__per__day, sleep__hours, and mental__health__rating explain overlapping variance. This makes sense since students who are well rested tend to study more and maintain better mental health. Because of this redundancy, we can drop 'sleep__hours' and re-evaluate the model to see if performance changes.

[54]:
```
# X = encoded.drop('exam_score', axis=1)
X = X.drop('sleep_hours', axis=1)
y = encoded['exam_score']

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"Linear Regression  R² = {r2:.3f},  Mean Squared Error = {rmse:.2f}")

# Make a data frame with columns and coeeficients
coef_df = (
    pd.DataFrame({
        'Feature': X.columns,
        'Coefficient': model.coef_
    })
    .sort_values(by='Coefficient', ascending=False)
    .reset_index(drop=True)
)

print("\nTop 10 influential features:")
print(coef_df.head(10))
```

```
Linear Regression  R² = 0.878,  Mean Squared Error = 5.83

Top 10 influential features:
                         Feature  Coefficient
0            study_hours_per_day     9.519527
1           mental_health_rating     1.981364
2                   gender_Other     1.426200
3             exercise_frequency     1.346150
4                    gender_Male     0.271688
5     parental_education_level_num     0.229571
6          attendance_percentage     0.154076
7                            age     0.100464
```

```
8  extracurricular_participation_Yes     0.089942
9                    part_time_job_Yes    -0.099819
```

Even after removing sleep_hours, model performance stayed nearly identical (R^2 = 0.878, MSE = 5.83), confirming that sleep_hours was highly correlated with other predictors (mainly study_hours_per_day) and contributed redundant information.

1.7 8. Conclusion

This analysis examined how various lifestyle and demographic factors influence student academic performance. Through exploratory analysis, we observed that study habits—particularly study hours per day—were most strongly correlated with exam performance (r = 0.83). Mental health, sleep, and exercise also showed moderate positive associations, while higher social media and streaming activity correlated with slightly lower exam scores.

Inferential testing revealed that parental education level had no statistically significant effect on exam scores (p = 0.81), suggesting that background factors were less influential than behavioral ones in this dataset. The linear regression model reinforced this pattern: study hours per day emerged as the dominant predictor, with each additional hour associated with approximately a 9.5-point increase in exam score. Other positive predictors included mental health, exercise, and sleep, while social media and Netflix use reduced predicted performance.

The model achieved an excellent fit (R^2 = 0.898, MSE = 5.32), explaining nearly 90% of the variance in exam scores. However, multicollinearity testing showed redundancy among study hours, sleep hours, and mental health rating. After removing sleep hours, model performance remained stable (R^2 = 0.878), confirming overlapping variance among these variables.

Overall, the findings suggest that active behavioral habits—consistent studying, adequate rest, and good mental health—drive academic success more strongly than demographic or background characteristics. Future analyses could extend this work by incorporating categorical logistic regression for pass/fail prediction or exploring non-linear effects to capture more complex behavioral patterns.