جامعة الملك سعود
King Saud University
كلية علوم الحاسب والمعلومات
قسم تقنية المعلومات

King Saud University
College of Computer and Information Sciences
Department of Information Technology
Second Semester 1446H – Winter 2025
IT462 – Big Data Systems

Student Stress Factors

Prepared by:

Student name	Student ID
Wassayef Alkherb	443200459
Maryam Altuwaijri	443200235
Bashair Alsadhan	443200668
Rana Alsayyari	443200565
Rama Alshebel	443200929

Supervised by:
Dr. Mashael AlSaleh

Table of Contents

List of Figures

1. Introduction

In today's fast-paced and high-pressure academic environment, student stressors have become a subject of importance! This affects their performance, mental health status, and general quality of life [1]. As stress affects students's lives, the demand for robust early detection has grown significantly. Identifying stress signs early will help students to prevent difficult situations. Our solution aims to enhance this process by providing an effective model for early support.

This report includes several key steps to prepare and analyze the dataset. It begins with data exploration, where we check for inconsistencies, missing values, redundant rows, and data types. Then, we perform correlation analysis to identify highly related features that might affect the model's performance. Additionally, we apply normalization to ensure all numerical values are on the same scale, making the data suitable for machine learning models. The preprocessing steps also involve encoding categorical values if needed, handling outliers, and selecting the best features for training.

data source: https://www.kaggle.com/datasets/rxnach/student-stress-factors-a-comprehensive-analysis

For a clearer look at the code and notebook, check our GitHub repository:
https://github.com/BashairAlsadhan/BigData-project

2. Dataset Overview

In this section, we will preview the dataset types and information and examine the dataset to understand its structure, including the total number of rows, the data types for each column, and any potential issues, such as redundant row or missing value.

In order to achieve our goal and prevent difficult situations, we looked for a dataset that comprehensively captures the various factors leading to stress among students. The “Student Stress Factors” dataset on Kaggle [2] stood out as an ideal resource for this purpose. It would provide an insight into the various stressors of students, including academic, personal, social, and financial problems.

2.1 Dataset Summary

"Student Stress Factors" dataset is survey-based, representing student responses to stress-related factors, includes 1100 entries which reflect diverse sample, contains 21 attributes that provide insights about stress levels and their possible causes, the attributes are selected scientifically considering 5 major factors: Psychological, Physiological, Social, Environmental, and Academic Factors (See Figure 1).

```
import pandas as pd

# Load the CSV file into a DataFrame

data = pd.read_csv("StressLevelDataset.csv")

# Display the first few rows of the DataFrame
data.head()
```

	anxiety_level	self_esteem	mental_health_history	depression	headache	blood_pressure	sleep_quality	breathing_problem	noise_level	living_conditions	...	basic_nee
0	14	20	0	11	2	1	2	4	2	3	...	
1	15	8	1	15	5	3	1	4	3	1	...	
2	12	18	1	14	2	1	2	2	2	2	...	
3	16	12	1	15	4	3	1	3	4	2	...	
4	16	28	0	7	2	3	5	1	3	2	...	

5 rows × 21 columns

Figure 1: First five rows of the stress level dataset.

These attributes include:

1. **anxiety_level**: ordinal feature ranges from 0 to 21, represents the level of anxiety experienced by the student, where higher values indicate higher anxiety.
2. **self_esteem**: ordinal feature ranges from 0 to 30, represents the level of self-esteem, where higher values indicate higher self-esteem.
3. **mental_health_history**: binary feature with values 0 and 1, where 0 indicates no history of mental health issues, and 1 indicates a history of mental health issues.
4. **depression**: ordinal feature ranges from 0 to 27, represents the level of depression experienced by the student, where higher values indicate more severe depression.
5. **headache**: ordinal feature ranges from 0 to 5, represents the frequency of headaches experienced by the student, where higher values indicate more frequent headaches.
6. **blood_pressure**: ordinal feature ranges from 1 to 3, represents the blood pressure level, where higher values may indicate higher blood pressure.
7. **sleep_quality**: ordinal feature ranges from 0 to 5, represents the quality of sleep, where higher values indicate better sleep quality.
8. **breathing_problem**: ordinal feature ranges from 0 to 5, represents the severity of breathing problems, where higher values indicate more severe issues.
9. **noise_level**: ordinal feature ranges from 0 to 5, represents the noise level in the student's environment, where higher values indicate noisier surroundings.

10. **living_conditions**: ordinal feature ranges from 0 to 5, represents the quality of living conditions, where higher values indicate better conditions.
11. **safety**: ordinal feature ranges from 0 to 5, represents the perception of safety, where higher values indicate a greater sense of safety.
12. **basic_needs**: ordinal feature ranges from 0 to 5, represents how well the student's basic needs are met, where higher values indicate better fulfillment.
13. **academic_performance**: ordinal feature ranges from 0 to 5, represents the student's academic performance, where higher values indicate better performance.
14. **study_load**: ordinal feature ranges from 0 to 5, represents the amount of academic workload, where higher values indicate a heavier workload.
15. **teacher_student_relationship**: ordinal feature ranges from 0 to 5, represents the quality of the teacher-student relationship, where higher values indicate stronger relationships.
16. **future_career_concerns**: ordinal feature ranges from 0 to 5, represents the level of concern about future career prospects, where higher values indicate greater concern.
17. **social_support**: ordinal feature ranges from 0 to 3, represents the level of social support the student receives, where higher values indicate more support.
18. **peer_pressure**: ordinal feature ranges from 0 to 5, represents the level of peer pressure experienced by the student, where higher values indicate greater pressure.
19. **extracurricular_activities**: ordinal feature ranges from 0 to 5, represents the level of involvement in extracurricular activities, where higher values indicate greater involvement.
20. **bullying**: ordinal feature ranges from 0 to 5, represents the extent of bullying experienced by the student, where higher values indicate more severe bullying.
21. **stress_level**: categorical feature with values ranging from 0 to 2, where 0 indicates low stress, 1 indicates moderate stress, and 2 indicates high stress.

stress_level is our class label in the dataset, representing the overall stress level of the student (see Figure 2).

```
# Calculate the number of columns
num_columns = data.shape[1]

# Get the names of the columns
column_names = data.columns.tolist()

# Calculate the number of responses (rows)
num_responses = data.shape[0]

# Print the results
print(f"Number of columns: {num_columns}")
print(f"Column names: {column_names}")
print(f"Number of responses: {num_responses}")
```

```
Number of columns: 21
Column names: ['anxiety_level', 'self_esteem', 'mental_health_history', 'depression', 'headache', 'blood_pressure', 'sleep_quality', 'breathing_problem',
'noise_level', 'living_conditions', 'safety', 'basic_needs', 'academic_performance', 'study_load', 'teacher_student_relationship', 'future_career_concern
s', 'social_support', 'peer_pressure', 'extracurricular_activities', 'bullying', 'stress_level']
Number of responses: 1100
```

Figure 2: the number of columns, column names, and number of rows in the dataset.

2.2 Summary Statistics

In this section, we will select all columns for the summary statistics to get an overview of the distribution of the factors which is essential for understanding the general characteristics of the dataset. Based on the results (see Figure 3) we observe that wide range in stress factors, the dataset reveals a range of stress-related factors, with anxiety, depression, and self-esteem showing the most variability. Furthermore, moderate to low value, as many of the stress-related factors, such as sleep quality, peer pressure, and teacher-student relationships, have moderate values, with some participants reporting high levels of stress and others reporting lower levels. Lastly, focus on wellbeing, as most of the factors, such as academic performance, social support, and extracurricular activities, show that participants are experiencing moderate levels of wellbeing, with some areas like social support and basic needs showing lower values.

```
summary_statistics = data.describe()
print(summary_statistics)
```

```
       anxiety_level  self_esteem  mental_health_history   depression  \
count    1100.000000  1100.000000            1100.000000  1100.000000
mean       11.063636    17.777273               0.492727    12.555455
std         6.117558     8.944599               0.500175     7.727008
min         0.000000     0.000000               0.000000     0.000000
25%         6.000000    11.000000               0.000000     6.000000
50%        11.000000    19.000000               0.000000    12.000000
75%        16.000000    26.000000               1.000000    19.000000
max        21.000000    30.000000               1.000000    27.000000

          headache  blood_pressure  sleep_quality  breathing_problem  \
count  1100.000000     1100.000000    1100.000000        1100.000000
mean      2.508182        2.181818       2.660000           2.753636
std       1.409356        0.833575       1.548383           1.400713
min       0.000000        1.000000       0.000000           0.000000
25%       1.000000        1.000000       1.000000           2.000000
50%       3.000000        2.000000       2.500000           3.000000
75%       3.000000        3.000000       4.000000           4.000000
max       5.000000        3.000000       5.000000           5.000000

       noise_level  living_conditions  ...  basic_needs  academic_performance  \
count  1100.000000        1100.000000  ...  1100.000000           1100.000000
mean      2.649091           2.518182  ...     2.772727              2.772727
std       1.328127           1.119208  ...     1.433761              1.414594
min       0.000000           0.000000  ...     0.000000              0.000000
25%       2.000000           2.000000  ...     2.000000              2.000000
50%       3.000000           2.000000  ...     3.000000              2.000000
75%       3.000000           3.000000  ...     4.000000              4.000000
max       5.000000           5.000000  ...     5.000000              5.000000

       study_load  teacher_student_relationship  future_career_concerns  \
count  1100.000000                   1100.000000             1100.000000
mean      2.621818                      2.648182                2.649091
std       1.315781                      1.384579                1.529375
min       0.000000                      0.000000                0.000000
25%       2.000000                      2.000000                1.000000
50%       2.000000                      2.000000                2.000000
75%       3.000000                      4.000000                4.000000
max       5.000000                      5.000000                5.000000

       social_support  peer_pressure  extracurricular_activities     bullying  \
count     1100.000000   1100.000000                 1100.000000  1100.000000
mean         1.881818      2.734545                    2.767273     2.617273
std          1.047826      1.425265                    1.417562     1.530958
min          0.000000      0.000000                    0.000000     0.000000
25%          1.000000      2.000000                    2.000000     1.000000
50%          2.000000      2.000000                    2.500000     3.000000
75%          3.000000      4.000000                    4.000000     4.000000
max          3.000000      5.000000                    5.000000     5.000000

       stress_level
count   1100.000000
mean       0.996364
std        0.821673
min        0.000000
25%        0.000000
50%        1.000000
75%        2.000000
max        2.000000

[8 rows x 21 columns]
```

Figure 3: Descriptive statistics of the dataset.

These summary statistics help to identify areas of concern (e.g., anxiety, depression, peer pressure) as well as those factors that seem to contribute positively to stress management (e.g., sleep quality, basic needs).

2.3 Visualizing distributions

In this step, we visualize the distribution of each variable in our dataset to better understand the underlying patterns and variations in the data. Visualizing distributions allows us to observe the central tendencies, spread, and potential skewness of the features. It also helps in identifying outliers, trends, and areas where data transformation or cleaning may be necessary. By plotting histograms for each column, we gain insights into the frequency and distribution of values for key stress-related factors, which is essential for guiding further analysis.

```python
import matplotlib.pyplot as plt

columns_to_plot = data.columns
fig, axes = plt.subplots(nrows=7, ncols=3, figsize=(12, 18))

#flatten the axes array for easy iteration
axes = axes.flatten()

# Plot for each column
for i, col in enumerate(columns_to_plot):
    axes[i].hist(data[col], bins=20, color='skyblue', edgecolor='black')
    axes[i].set_title(col)
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Frequency')

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

Figure 4: Python code used for visualizing the distribution of different columns.

Based on the visualizations (see Figure 5), the dataset shows that anxiety levels are moderately varied, with most participants reporting moderate self-esteem and a small portion having a mental health history. Depression is generally mild, with few participants experiencing severe cases. Headaches are common but not severe, and most participants report mild to moderate blood pressure concerns. Sleep quality is decent for most, though some experience poor sleep, and breathing problems are moderate to high. Noise levels are moderate, which may contribute to stress, while living conditions, basic needs, academic performance, and study load are generally average. Teacher-student relationships are rated moderately well, and future career concerns are moderate to high. Social support is low to moderate for many, with peer pressure and bullying being moderate to high for some. Most participants are moderately engaged in extracurricular activities, and stress levels are predominantly low, though some report moderate to high stress.

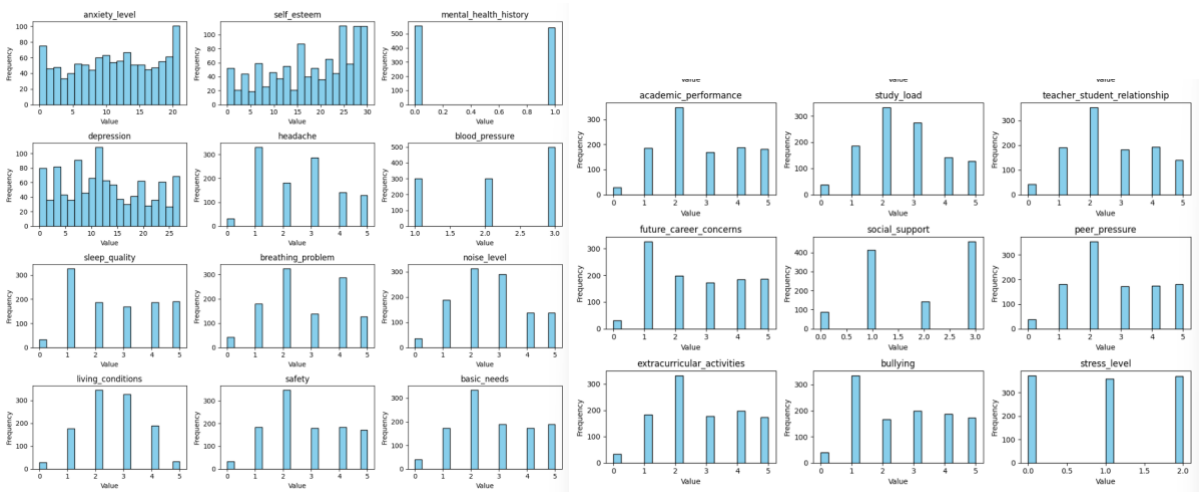

Figure 5: Histogram for all columns.

3. Preprocessing

In this section, In this section, we will focus on different pre-processing steps for our dataset to ensure its reliability, this include checking for missing values, duplicates rows, data inconsistencies, Multicollinearity, normalization, data integration, reduction and transformation.

3.1 Check for Missing Values

The output of data.info() (see Figure 6) provides another summary of the dataset, helping to understand its structure before applying preprocessing steps. The dataset consists of 1,100 entries (rows) and 21 columns (features), meaning each row represents an individual student's response.

Each feature is of type integer (int64), indicating that all values are numerical and there are no categorical or text-based columns. Additionally, there are no missing values, as every column has 1,100 non-null entries, meaning each student provided responses for all attributes.

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100 entries, 0 to 1099
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   anxiety_level                 1100 non-null   int64
 1   self_esteem                   1100 non-null   int64
 2   mental_health_history         1100 non-null   int64
 3   depression                    1100 non-null   int64
 4   headache                      1100 non-null   int64
 5   blood_pressure                1100 non-null   int64
 6   sleep_quality                 1100 non-null   int64
 7   breathing_problem             1100 non-null   int64
 8   noise_level                   1100 non-null   int64
 9   living_conditions             1100 non-null   int64
 10  safety                        1100 non-null   int64
 11  basic_needs                   1100 non-null   int64
 12  academic_performance          1100 non-null   int64
 13  study_load                    1100 non-null   int64
 14  teacher_student_relationship  1100 non-null   int64
 15  future_career_concerns        1100 non-null   int64
 16  social_support                1100 non-null   int64
 17  peer_pressure                 1100 non-null   int64
 18  extracurricular_activities    1100 non-null   int64
 19  bullying                      1100 non-null   int64
 20  stress_level                  1100 non-null   int64
dtypes: int64(21)
memory usage: 180.6 KB
None
```

Figure 6: Checking missing value.

3.2 Check for Duplicates Rows

After ensuring that there are no missing values, it's crucial to check for redundant rows that could cause several issues. Including increasing computation time and slowing down processing and analysis. As Figure 7 shows, there are no duplicate rows in the dataset.

```
duplicates = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")
```

```
Number of duplicate rows: 0
```

Figure 7: Python code for duplicating rows checking and its output.

3.3 Check for Data Inconsistencies

Before applying any preprocessing techniques, it is crucial to ensure that the dataset is free from inconsistencies. This step involves verifying that all values fall within their predefined valid ranges, as described in the dataset documentation. Additionally, we need to check for missing values and non-numeric entries, which could indicate data entry errors or formatting issues. By performing this validation, we can detect potential anomalies that might negatively impact subsequent data processing and model performance. The following code systematically examines the dataset, highlighting any inconsistencies that need to be addressed before further analysis. Results are printed in Figure 8.

```
# Print results
if inconsistencies:
    print("⚠️ Inconsistent Values Found (Out of Range):")
    for col, values in inconsistencies.items():
        print(f" - {col}: {values.tolist()}")
else:
    print("✅ No out-of-range values found in the dataset.")

if non_numeric_values:
    print("\n⚠️ Non-Numeric Values Found:")
    for col, values in non_numeric_values.items():
        print(f" - {col}: {values.tolist()}")
else:
    print("\n✅ No non-numeric values found in the dataset.")

if not missing_values.empty:
    print("\n⚠️ Missing Values Found:")
    print(missing_values)
else:
    print("\n✅ No missing values in the dataset.")
```

```
✅ No out-of-range values found in the dataset.

✅ No non-numeric values found in the dataset.

✅ No missing values in the dataset.
```

Figure 8: Output of data inconsistencies checking.

3.4 Normalization

Since our dataset contains numerical features with varying scales, we apply normalization to ensure that all values are within a consistent range. Normalization is particularly important when working with machine learning models that rely on distance-based calculations, such as k-nearest neighbors (KNN) and neural networks. By transforming all numerical values to a uniform scale, we prevent attributes with larger ranges from disproportionately influencing the model.

Min-Max Scaling (as shown in Figure 9 and Figure 10) is chosen because it preserves the original distribution of data while ensuring that all values remain within a comparable range. This method is especially useful when working with datasets containing ordinal features, as it maintains the relative differences between values without distorting their relationships.

```python
from sklearn.preprocessing import MinMaxScaler



# Define the columns to normalize (excluding binary and categorical features)
columns_to_normalize = [
    "anxiety_level", "self_esteem", "depression", "headache",
    "blood_pressure", "sleep_quality", "breathing_problem", "noise_level",
    "living_conditions", "safety", "basic_needs", "academic_performance",
    "study_load", "teacher_student_relationship", "future_career_concerns",
    "social_support", "peer_pressure", "extracurricular_activities", "bullying"
]

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Normalize only the selected columns and replace them in the original dataset
data[columns_to_normalize] = scaler.fit_transform(data[columns_to_normalize])

# Save the updated dataset with normalized columns
data.to_csv("cleaned_student_stress_factors.csv", index=False)

# Display the first few rows of the updated dataset
print(data.head())
```

Figure 9: Python code for normalization.

```
   anxiety_level  self_esteem  mental_health_history  depression  headache  \
0       0.666667     0.666667                      0    0.407407       0.4
1       0.714286     0.266667                      1    0.555556       1.0
2       0.571429     0.600000                      1    0.518519       0.4
3       0.761905     0.400000                      1    0.555556       0.8
4       0.761905     0.933333                      0    0.259259       0.4

   blood_pressure  sleep_quality  breathing_problem  noise_level  \
0             0.0            0.4                0.8          0.4
1             1.0            0.2                0.8          0.6
2             0.0            0.4                0.4          0.4
3             1.0            0.2                0.6          0.8
4             1.0            1.0                0.2          0.6

   living_conditions  ...  basic_needs  academic_performance  study_load  \
0                0.6  ...          0.4                   0.6         0.4
1                0.2  ...          0.4                   0.2         0.8
2                0.4  ...          0.4                   0.4         0.6
3                0.4  ...          0.4                   0.4         0.8
4                0.4  ...          0.6                   0.8         0.6

   teacher_student_relationship  future_career_concerns  social_support  \
0                           0.6                     0.6        0.666667
1                           0.2                     1.0        0.333333
2                           0.6                     0.4        0.666667
3                           0.2                     0.8        0.333333
4                           0.2                     0.4        0.333333

   peer_pressure  extracurricular_activities  bullying  stress_level
0            0.6                         0.6       0.4             1
1            0.8                         1.0       1.0             2
2            0.6                         0.4       0.4             1
3            0.8                         0.8       1.0             2
4            1.0                         0.0       1.0             1

[5 rows x 21 columns]
```

Figure 10: output of the normalization code.

3.5 Check for Multicollinearity

Before training the model, it is important to check the correlation between numerical features in the dataset. If two features are highly correlated, they might provide similar information, which can cause problems for some models, such as linear regression and logistic regression. This issue, known as multicollinearity, can make it harder to understand the importance of each feature and may affect model performance.

To identify such cases, we calculate the correlation matrix and flag features with a correlation higher than 0.80 (see Figure 11). This threshold is based on common guidelines, as correlations above 0.80 are often considered strong and may indicate overlapping information. Removing highly correlated features can sometimes improve model performance, but this should not be done automatically.

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


# Compute the correlation matrix
correlation_matrix = data.corr()

# Define correlation threshold (e.g., 0.80 means high correlation)
threshold = 0.80

# Find highly correlated features
correlated_columns = set()
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:  # Check absolute correlation
            colname1 = correlation_matrix.columns[i]
            colname2 = correlation_matrix.columns[j]
            correlated_columns.add((colname1, colname2))

# Print the correlation matrix
print("🔹 Correlation Matrix:")
print(correlation_matrix)

# Plot the heatmap for visualization
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()

# Print highly correlated column pairs
if correlated_columns:
    print("\n⚠️ Highly Correlated Columns (Threshold > 0.85):")
    for col1, col2 in correlated_columns:
        print(f" - {col1} ↔ {col2} (Correlation: {correlation_matrix.loc[col1, col2]:.2f})")
else:
    print("\n✅ No highly correlated columns found above the threshold.")
```

Figure 11: Python code to check multicollinearity.

Instead of immediately dropping features, it is better to test different feature combinations during model training and evaluate their impact. This approach helps ensure that we do not remove important information by mistake. The next step may involve training models with different feature sets to find the best combination. The following code calculates the correlation matrix, lists the highly correlated columns, and visualizes the results with a heatmap (see Figure 12).

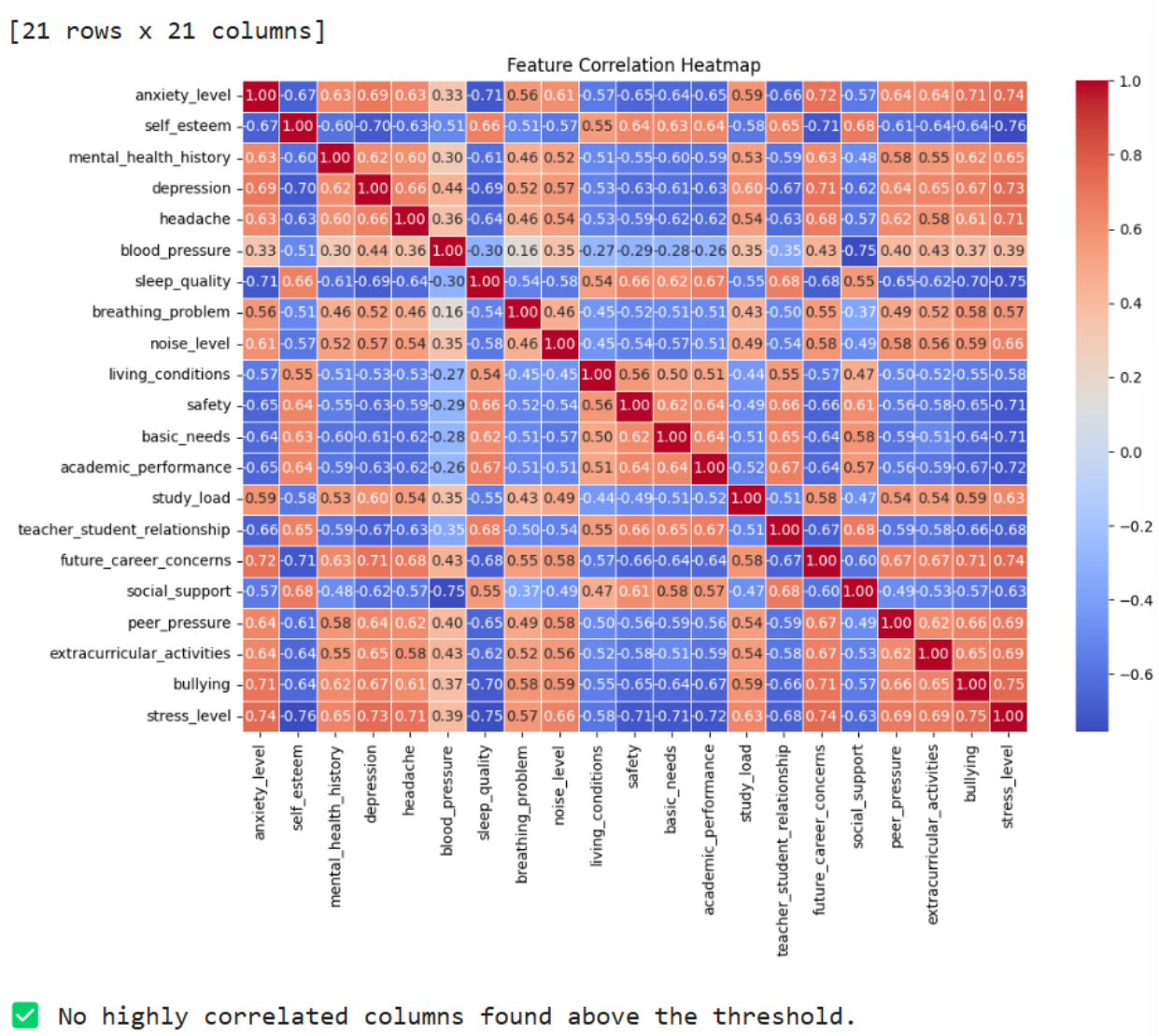

Figure 12: Part of the Multicollinearity code output.

3.6 Data integration, Reduction and Transformation

At this stage, data integration, reduction, and transformation are not required. The dataset is already well-structured, containing all necessary attributes without redundant or duplicated information. Additionally, since the dataset has a manageable size (1,100 entries and 21 features), dimensionality reduction is unnecessary. The data is also in the correct numerical format, meaning no further transformations are needed before proceeding with preprocessing and model training.

4. RDD Operations

After required preprocessing, this section presents a comprehensive operation of a dataset containing student lifestyle and stress-related factors using Apache Spark's Resilient Distributed Datasets (RDDs). The objective is to leverage RDD transformations and actions to extract meaningful insights related to stress levels, lifestyle patterns, and academic behavior among students.

4.1 RDD Transformations

This sub-section covers the transformation operations that were applied to manipulate and restructure the dataset. Transformations in Spark are lazily evaluated operations used to define a new RDD from an existing one. The following transformations were implemented:

1. Categorization by stress level

Each student record was categorized into three classes based on the stress value: Low, Moderate, or High. This transformation simplifies subsequent analysis and visualizations by creating labeled stress groups. We start with this transformation to facilitate aggregated analysis and enable comparisons between stress categories.
RDD operation used: *map()* to convert each row into a key-value pair as (category, 1) for each one to prepare for aggregation.

```
// Transformation 1: Categorize Students by Stress Level (Low, Moderate, High)
val categorized = rows.map(r => {
  val stress = Try(r.last.toInt).getOrElse(0)
  if (stress == 0) ("Low", 1)
  else if (stress == 1) ("Moderate", 1)
  else ("High", 1)
})
```

Figure 13: Categorizing students into three stress levels (Low, Moderate, High).

2. Filtering vulnerable students

A filtered subset of students was created by selecting those with a high stress level (value 2) and low support score (below 0.5). This transformation isolates the most at-risk population for targeted analysis. This transformation identifies students likely to be facing significant academic and emotional challenges due to lack of support.
RDD operation used: *filter()* to isolate students with high stress (stress = 2) and poor support (support score < 0.5).

```
// Transformation 2: Filter Students with Poor Support + High Stress
val vulnerableStudents = rows.filter(r => Try(r.last.toInt).getOrElse(0) == 2 &&
Try(r(16).toDouble).getOrElse(1.0) < 0.5)
```

Figure 14: Filtering Vulnerable Students (High Stress & Poor Support).

3. Grouping by course (Academic performance) and averaging stress

Students were grouped by academic performance scores and the average stress level was calculated per group. This operation highlights whether academic achievement is associated with higher or

lower stress levels. This transformation can investigate potential correlations between academic performance and student stress.
RD operations used: *map()* to extract (course, stress) pairs for counting, *reduceByKey()* to sum and count stress values per coursemap, and *mapValues()* to calculate averages.

```
// Transformation 3: Group Students by Academic Performance Score and Calculate Average Stress Level
val avgStressByPerformance = rows.map(r
=> (r(12), (Try(r.last.toInt).getOrElse(0), 1))).reduceByKey((a, b)
=> (a._1 + b._1, a._2 + b._2)).mapValues { case (sum, count) => sum.toDouble / count }
```

Figure 15: Grouping by academic performance and averaging stress

4. Clustering by lifestyle archetypes

A transformation was applied to classify students into lifestyle clusters based on a combination of six features: Sleep quality, Study load, Support score, Academic performance, Teacher-student relationship, and Peer pressure. Each cluster was labeled with a descriptive tag indicating behavioral patterns (e.g., low_sleep_high_study_low_support_poor_perf_good_teacher_high_pressure), to segment students into interpretable lifestyle groups that can be used for population-level insights.
RDD operations used: *map()* to compute feature thresholds and concatenate labels into a unique cluster tag for each student.

```
val lifestyleClusters = rows.map(r => {
  val sleep = Try(r(6).toDouble).getOrElse(0.0)
  val study = Try(r(13).toDouble).getOrElse(0.0)
  val support = Try(r(16).toDouble).getOrElse(0.5)
  val performance = Try(r(12).toDouble).getOrElse(0.5)
  val teacherRel = Try(r(14).toDouble).getOrElse(0.5)
  val peerPressure = Try(r(17).toDouble).getOrElse(0.5)

  val tag = (if (sleep < 0.5) "low_sleep_" else "high_sleep_") +
            (if (study > 0.5) "high_study_" else "low_study_") +
            (if (support < 0.5) "low_support_" else "high_support_") +
            (if (performance >= 0.6) "good_perf_" else "poor_perf_") +
            (if (teacherRel < 0.4) "poor_teacher_" else "good_teacher_") +
            (if (peerPressure > 0.6) "high_pressure" else "low_pressure")

  (tag, 1)
})
```

Figure 16: Clustering students into lifestyle groups.

5. Ranking by sleep-to-study ratio

A ratio was calculated between the sleep quality and study load for each student. This ratio provides insight into time management and a potential imbalance between rest and academic work to rank students by how well they manage their time between rest and academic responsibilities.
RDD operations used: *map()* to calculate ratio and pair it with each row.

```
// Transformation 5: Rank Students by Sleep-to-Study Ratio
val ratios = rows.map(r => {
  val sleep = Try(r(6).toDouble).getOrElse(0.0)
  val study = Try(r(13).toDouble).getOrElse(1.0)
  val ratio = if (study == 0) 0.0 else sleep / study
  (r, ratio)
})
```

Figure 17: A transformation calculates a sleep-to-study ratio for each student.

6. **Filtering students with balanced lifestyles**

This transformation was applied to identify students who demonstrated a balanced lifestyle, those with adequate sleep (≥ 0.7), moderate study hours (between 0.2 and 0.4), and no stress (stress = 0). These students can serve as reference profiles for healthy behavioral benchmarks.

RDD operation used: *filter()* was used to extract only the rows satisfying the above conditions.

```
// Transformation 6: Students with Balanced Life ( Sleep>= 7 hours, Study 2-4 hours, stress levell = 0)
val balancedCandidates = rows.filter(r =>
  Try(r(6).toDouble).getOrElse(0.0) >= 0.7 &&
  Try(r(13).toDouble).getOrElse(0.0) >= 0.2 &&
  Try(r(13).toDouble).getOrElse(0.0) <= 0.4 &&
  Try(r.last.toDouble).getOrElse(0.0) <= 0
)
```

Figure 18: Filtering students with balanced lifestyles.

7. **Grouping by study load and averaging stress levels**

This transformation was applied to group students by their study load values and computing the corresponding average stress level for each group. The goal is to analyze how different levels of academic workload affect student stress, offering insights into whether heavy study schedules contribute to elevated stress. By examining these groupings, stakeholders can better understand the impact of academic demands on mental well-being.

RDD operations used: *map()* was used to extract each student's study load and stress level as a key-value pair, *groupByKey()* aggregated all stress values for each unique study load, and *mapValues()* was applied to calculate the average stress level by dividing the total stress by the number of students in each group.

```
// Transformation 7: Group students by study load and compute the average stress level for each group
val studyLoadRDD = rows.map(arr => (arr(14).toDouble, arr(20).toInt))
val avgStressPerStudyLoad = studyLoadRDD.groupByKey().mapValues(values
=> values.sum.toDouble / values.size)
```

Figure 19: Grouping students by study load and calculating average stress level.

4.2 RDD Action

This sub-section covers the action operations used to derive insights from the transformed RDDs. While transformations describe what to do with the data, actions are operations that trigger execution of those transformations and return actual results to the driver or to external systems. Below are the six action operations applied, along with their purpose, corresponding RDD operations, code references, and output insights.

1. **Counting students in each stress category**

This action aggregates the number of students in each stress category (Low, Moderate, High) to understand the overall distribution of stress levels across the student population. Since quantifying students in each group helps identify the scale of stress-related issues and prioritize support strategies.

RDD operations used: *collect()* to execute the computation and return the final counts to the driver, *foreach()* was used to iterate through the collected results and print them in a readable format. From transformation, we use *reduceByKey(_ + _)* to aggregate the counts for each stress category key.

Code snippet of the action:

```
// Action 1: Count Students in Each Stress Category
val categoryCounts = categorized.reduceByKey(_ + _).collect()
// Print the distribution of Students by Stress Level
println( "Stress Level Categories Count:")
categoryCounts.foreach { case (level, count) =>
  println(s" - $level: $count students")
}
```

Figure 20: Action 1-Counting students in each stress category.

The output of this action (as shown in Figure 21) shows that the stress distribution is relatively balanced across categories. Interestingly, 373 low-stress students slightly outnumber 369 high-stress ones, but the difference is minimal. This suggests that about half of the population is experiencing high levels of stress, which is a significant concern. It may indicate that while some students manage stress well, a substantial group is under serious psychological strain and may require dedicated support services or preventive mental health programs.

```
Stress Level Categories Count:
 - High: 369 students
 - Low: 373 students
 - Moderate: 358 students
```

Figure 21: Output of action 1.

2. Displaying the first vulnerable student (high Stress + low support)

This action retrieves the first student who fits the criteria of being highly stressed (stress level = 2) and having low emotional/social support (support score < 0.5). By examining this individual case in detail, we can better understand how stress manifests alongside lack of support and what other contextual features are associated with vulnerability.

RDD operations used: *first()* to return the first element from the filtered RDD of vulnerable students.

Code snippet of the action:

```
// Action 2: Display the first Vulnerable Student (High Stress + Low Support)
val firstVulnerable = vulnerableStudents.first()
// Print the student's data
println("\n First Vulnerable Student Record (High Stress + Low Support):")
println("Field".padTo(25, ' ') + "Value")
println("-" * 40)
featureNames.zip(firstVulnerable).foreach { case (name, value) =>
  println(name.padTo(25, ' ') + value)
}
```

Figure 22: Action 2-Displaying the first vulnerable student

This student's profile (as shown in Figure 23) reveals a cluster of compounding stressors. They struggle with poor sleep, low academic performance, high peer pressure, and report being bullied, all while lacking adequate social and teacher support. These findings reinforce the idea that stress is multidimensional, stemming from both personal challenges and environmental factors. Institutions should develop holistic intervention strategies that address not just academics, but also social relationships, safety, and emotional resilience.

```
 First Vulnerable Student Record (High Stress + Low Support):
Field                    Value
---------------------------------------
anxiety_level            0.7142857142857142
self_esteem              0.26666666666666666
mental_health_history    1
depression               0.5555555555555556
headache                 1.0
blood_pressure           1.0
sleep_quality            0.2
breathing_problem        0.8
noise_level              0.6000000000000001
living_conditions        0.2
safety                   0.4
basic_needs              0.4
academic_performance     0.2
study_load               0.8
teacher_student_relationship0.2
future_career_concerns   1.0
social_support           0.3333333333333333
peer_pressure            0.8
extracurricular_activities1.0
bullying                 1.0
stress_level             2
```

Figure 23: Output of action 2.

3. Identifying the top 5 academic performance levels with highest stress average

This action determines which academic performance score brackets are associated with the highest average stress. Academic scores are categorized (e.g., Very Poor, Poor, Average, Good, Excellent), and the top five levels with the highest stress averages are selected. This analysis helps to uncover which students are most academically burdened.

RDD operations used: *takeOrdered(5)* to sort performance groups by their average stress (in descending order) and returns the top five.

Code snippet of the action:

```
// Action 3: Top 5 Academic Performance Levels with Highest Average Stress
val top5Courses = avgStressByPerformance.takeOrdered(5)(Ordering[Double].reverse.on { case (score, _)
=> score.toDouble })
// Print the top 5
println("\nTop 5 Academic Performance Levels with Highest Average Stress:")
top5Courses.zipWithIndex.foreach { case ((scoreStr, avg), i) =>
  val score = scoreStr.toDouble
  val label = score match {
    case s if s < 0.2  => "Very Poor"
    case s if s < 0.4  => "Poor"
    case s if s < 0.6  => "Average"
    case s if s < 0.8  => "Good"
    case _             => "Excellent"
  }
  val roundedScore = BigDecimal(score).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
  println(s" ${i + 1}. Performance Level: $label ($roundedScore)
  → Avg Stress: ${avg.formatted("%.2f")}")
}
```

Figure 24: Action 3-Identifying the top 5 academic performance levels with highest average

Output of action 3 (as shown in Figure 25) indicate a clear inverse relationship between academic performance and stress: students with lower scores (0.2–0.4) tend to report higher stress, while top performers (0.8–1.0) report the least. This suggests that underperforming students may feel overwhelmed, possibly due to fear of failure or lack of academic skills. The data can guide student counseling services to focus stress management and academic support efforts toward average and below-average achievers.

```
Top 5 Academic Performance Levels with Highest Average Stress:
warning: one deprecation (since 2.12.16); for details, enable `:setting -deprecation' or `:replay -deprecation'
 1. Performance Level: Excellent (1.0) → Avg Stress: 0.15
 2. Performance Level: Excellent (0.8) → Avg Stress: 0.18
 3. Performance Level: Good (0.6) → Avg Stress: 1.02
 4. Performance Level: Average (0.4) → Avg Stress: 1.42
 5. Performance Level: Poor (0.2) → Avg Stress: 1.85
```

Figure 25: Output of action 3.

4. Counting students per lifestyle cluster

This action quantifies how many students fall into each lifestyle group previously defined using behavioral patterns (sleep, study habits, support, etc.). The clusters allow us to segment students into groups with shared characteristics, making it easier to target specific interventions or guidance strategies.

RDD operations used: *collect()* action retrieves the results, *take(5)* was then applied to select only the top 5 most common lifestyle clusters, *foreach()* printed the cluster names along with their student counts. From transformation, we use *reduceByKey(_ + _)* that sums the number of students in each cluster, sortBy() sorted the cluster counts in descending order, ensuring the most frequent clusters appear first.

Code snippet of the action:

```
98   // Action 4: Count students per lifestyle cluster
99   val lifestyleCounts = lifestyleClusters.reduceByKey(_ + _).collect()
100  // Print the most Frequent Lifestyle Clusters
101  println("\nTop 5 Most Common Lifestyle Clusters:")
102  lifestyleCounts.sortBy(-_._2).take(5).foreach { case (cluster, count) =>
103    println(s" - $cluster: $count students")
104  }
```

Figure 26: Action 4-Counting students per lifestyle cluster

The most common cluster (students with high sleep, low study, and high support) reflects a generally balanced lifestyle and better outcomes (as shown in Figure 27). Conversely, clusters with low support, high study load, and poor performance are still significantly populated, indicating at-risk groups. Institutions could design cluster-targeted programs, offering lifestyle coaching or peer mentoring for students who fall into high-risk behavioral patterns.

```
Top 5 Most Common Lifestyle Clusters:
 - high_sleep_low_study_high_support_good_perf_good_teacher_low_pressure: 315 students
 - low_sleep_high_study_low_support_poor_perf_poor_teacher_high_pressure: 156 students
 - low_sleep_high_study_low_support_poor_perf_good_teacher_high_pressure: 151 students
 - low_sleep_low_study_high_support_good_perf_good_teacher_high_pressure: 27 students
 - low_sleep_high_study_high_support_poor_perf_good_teacher_low_pressure: 27 students
```

Figure 27: Output of action 4.

5. Sleep-to-study ratio histogram

This action creates histogram-like distribution buckets based on the ratio of sleep to study time for each student. This analysis helps identify whether students are overworked or under-rested, which are both potential risk factors for stress.

RDD operations used: *reduceByKey(_ + _)* to count the number of students in each ratio range, *foreach()* to print the result for reporting.

Code snippet of the action:

```
// Action 5: Ratio Range Histogram
val ratioBuckets = ratios.map { case (_, ratio) =>
  val bucket = ratio match {
    case r if r < 0.5  => "[0.0 - 0.5)"
    case r if r < 1.0  => "[0.5 - 1.0)"
    case r if r < 1.5  => "[1.0 - 1.5)"
    case r if r < 2.0  => "[1.5 - 2.0)"
    case _             => "2.0+"
  }
  (bucket, 1)
}.reduceByKey(_ + _)  // ✅ transformation
// Prints the Sleep-to-Study Ratio
println("\n Sleep-to-Study Ratio Distribution (Printed via foreach):")
ratioBuckets.foreach { case (range, count) =>
  println(f"$range%-12s → $count students")
}
```

Figure 28: Action 5-Sleep-to-study ratio histogram

We can see that 381 students which represent a large number of students fall into the [0.0–0.5) bucket (as shown in Figure 29), meaning their study time vastly outweighs sleep, suggesting possible sleep deprivation. On the other hand, 333 students have a ratio >2.0, indicating they may be over-sleeping or under-studying. Both extremes can negatively affect performance and well-being. This distribution supports initiatives promoting healthy time management and balanced routines.

```
 Sleep-to-Study Ratio Distribution (Printed via foreach):
[0.0 - 0.5)  → 381 students
[1.0 - 1.5)  → 209 students
2.0+         → 333 students
[0.5 - 1.0)  → 104 students
[1.5 - 2.0)  → 73 students
```

Figure 29: Output of action 5.

6. Collecting sample students with balanced life

This action extracts a random sample of students who meet the criteria for a “balanced life”: high sleep scores, moderate study load, and no stress. These samples help identify model behaviors for wellness programs or serve as candidates for qualitative interviews.

RDD operations used: *takeSample(false, 5)* that randomly selects 5 students from the already filtered balanced group, *foreach()* was used to iterate over the sampled students and print each row’s data, preceded by column headers split

Code snippet of the action:

```
// Action 6: Collect Sample Students with Balanced Life
val balancedStudents = balancedCandidates.takeSample(false, 5)
val featureNames = header.split(",")
// Print sample of Students with a Balanced Lifestyle
println("\nSample of Students with Balanced Lifestyle:")
println(featureNames.mkString(", "))
balancedStudents.foreach(row => println(row.mkString(", ")))
```

Figure 30: Action 6-Collecting sample students with balanced life

The output (as shown in Figure 31) shows five identified students who met balanced criteria, exhibiting sleep scores between 0.8 and 1.0, ideal study loads, and zero stress levels. They demonstrated favorable trends in self-esteem, social support, and academic performance, along

with traits like emotional resilience and effective time management. This insight is valuable for institutions aiming to enhance mental health and academic performance. By understanding these students' characteristics, educators can identify factors that contribute to their success and use them as benchmarks for wellness initiatives or peer mentorship programs. Ultimately, this fosters a broader understanding of how to encourage balance among the entire student body.

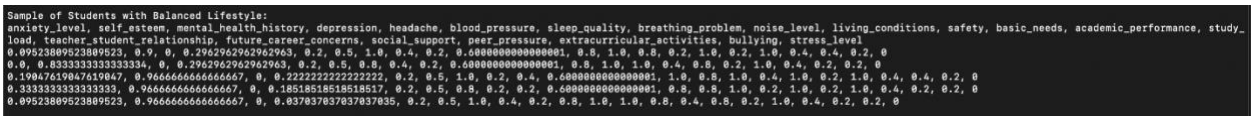

Figure 31: Output of action 6.

7. Identifying the top 5 study loads with highest average stress

This action analyzes the relationship between study load and stress by identifying the top five study load values associated with the highest average stress levels. By grouping students based on how much time they dedicate to studying and computing the corresponding average stress, this analysis aims to uncover whether certain study loads are more likely to be linked with stress accumulation. Such insights can guide curriculum adjustments, academic advising, and personalized study recommendations to support student well-being.

RDD operations used: *takeOrdered(5)* was applied on the previously computed average stress values per study load, ordering them in descending order by stress levels. *foreach()* was used to iterate over the top five entries and format the results for display. From the transformation step. From transformation, we use *map()* paired each study load with its stress value, *groupByKey()* collected all stress values for each unique study load, and *mapValues()* calculated their average.
Code snippet of the action:

```
136   // Action 7:
137   val topStressStudyLoads = avgStressPerStudyLoad.takeOrdered(5)(Ordering[Double].reverse.on(_._2))
138   println("\n Top 5 Study Loads with Highest Avg Stress:")
139   topStressStudyLoads.foreach { case (studyLoad, avgStress) =>
140     println(f"Study Load: $studyLoad%.2f → Avg Stress: ${avgStress.formatted("%.2f")}")
141   }
```

Figure 32: Identifying the top 5 study loads with highest average stress.

The output of this action (as shown in Figure 33) reveals that students with a study load of 0.20 experience the highest average stress (1.80), followed by 0.40 and 0.60, which also show elevated stress levels. Interestingly, very high study loads such as 0.80 are associated with a much lower stress average (0.18). This trend suggests that moderate study loads may correlate with stress due to inefficient study strategies or internal pressure, while students with the highest workloads might be more organized or better supported. These results highlight the need for a nuanced understanding of study habits and recommend targeted support for students in the moderate study range who might be underperforming or feeling overwhelmed.

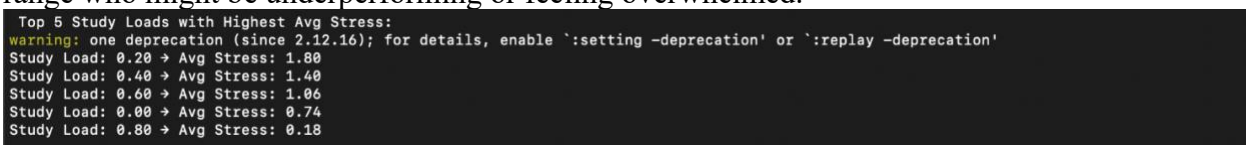

Figure 33: Output of action 7.

5. SQL Operations

This chapter explores key insights from the student stress dataset using SQL queries in Spark. By analyzing factors like extracurricular activities, academic performance, social support, and sleep quality, we identify how these variables influence stress levels and student well-being.

Each query is designed to answer a specific question, uncovering patterns that can help guide support strategies for students. The results are presented with both query logic and visual interpretations to highlight meaningful trends.

1. **How are the extracurricular activity levels affecting the student performance?**
   Figure 34 illustrates the impact of extracurricular activities on students' stress levels. The data reveals that as participation in extracurricular activities increases, stress levels also rise. To analyze this effect, we categorized extracurricular activity levels into five groups: "Very Low", "Low", "Moderate", "High", and "Very High". Our goal was to examine how these activities influence the study load, stress levels, and academic performance.

   Initially, we could have said that higher study load causes higher stress levels and lower academic performance. However, it's important to consider other related features. After considering the extracurricular activities rate, we notice that as extracurricular involvement, the study load increases, therefore stress levels increase, while academic performance declines.
   There could be potential bias in the dataset. The study load was originally measured on an ordinal scale from 0 to 5, and the data suggests that students with higher activity levels report a higher study load. This could be due to a bias among students who participate in more activities, as they may feel they have less time, leading them to rate their study load as high. With limited free time due to both academic and extracurricular commitments, students experience increased stress, which, in turn, negatively impacts their academic performance. Overall, helping students with time management skills, and efficient study methods could decrease their stress levels, which can improve their quality of life.

```
val query_1 =
"""
SELECT
    CASE
        WHEN extracurricular_activities <= 0.2 THEN 'Very Low Activity'
        WHEN extracurricular_activities <= 0.4 THEN 'Low Activity'
        WHEN extracurricular_activities <= 0.6 THEN 'Moderate Activity'
        WHEN extracurricular_activities <= 0.8 THEN 'High Activity'
        WHEN extracurricular_activities <= 1 THEN 'Very High Activity'
    END AS Activity_level,
    round(avg(study_load),2) AS AVG_Study_Load,
    round(avg(stress_level), 2) AS AVG_Stress,
    round(avg(academic_performance), 2) AS AVG_Academic_Performance
FROM stressDF
GROUP BY Activity_level
ORDER BY AVG_Stress
"""
val result_1 = spark.sql(query_1)

// Query1:
println("How are the activity levels affecting the student performance?")
result_1.show()
```

Figure 34: Extracurricular Activities and Stress Levels relation query

```
How are the activity levels affecting the student performance?
+------------------+--------------+----------+------------------------+
|    Activity_level|AVG_Study_Load|AVG_Stress|AVG_Academic_Performance|
+------------------+--------------+----------+------------------------+
| Very Low Activity|          0.36|      0.32|                    0.79|
|      Low Activity|          0.41|      0.54|                    0.68|
|     High Activity|          0.62|      1.41|                    0.42|
|Very High Activity|          0.74|      1.81|                    0.32|
+------------------+--------------+----------+------------------------+
```

Figure 35: Extracurricular Activities and Stress Levels relation output

2. **How do future career concerns affect students' stress levels?**

   We categorized academic performance into six levels: "Failing", "Low", "Moderate", "Good", "High", and "Excellent". Our analysis shows that students with Excellent and High academic performance tend to have the lowest concerns about their future careers. Interestingly, while students who are failing still experience high stress and concern, their stress levels are lower compared to those with Good, Moderate, or Low academic performance, this could be because students with moderate performance have higher career aspirations leading to increased anxiety about their future.

   The data indicates that as academic performance declines, both career related concerns and stress levels increase. This trend suggests that students with lower performance may benefit from career counseling, which could help decrease their stress and provide guidance for their future.

```scala
val query_2 =
"""
SELECT
    CASE
        WHEN academic_performance >= 0.0 AND academic_performance < 0.2 THEN 'Failing'
        WHEN academic_performance >= 0.2 AND academic_performance < 0.4 THEN 'Low'
        WHEN academic_performance >= 0.4 AND academic_performance < 0.6 THEN 'Moderate'
        WHEN academic_performance >= 0.6 AND academic_performance < 0.8 THEN 'Good'
        WHEN academic_performance >= 0.8 AND academic_performance < 1.0 THEN 'High'
        WHEN academic_performance = 1.0 THEN 'Excellent'
    END AS Performance,
    count(academic_performance) AS Nnumber_of_Students,
    round(avg(future_career_concerns), 2) AS AVG_Future_Career_Concerns,
    round(avg(stress_level), 2) AS AVG_Stress
FROM stressDF
GROUP BY Performance
ORDER BY AVG_Stress
"""
val result_2 = spark.sql(query_2)

// Query2:
println("How are future career concerns affecting Stress Levels for students?")
result_2.show()
```

Figure 36: Career concerns effect on students' stress query

```
How are future career concerns affecting Stress Levels for students?
+-----------+------------------+--------------------------+----------+
|Performance|Nnumber_of_Students|AVG_Future_Career_Concerns|AVG_Stress|
+-----------+------------------+--------------------------+----------+
|  Excellent|               182|                      0.26|      0.15|
|       High|               188|                      0.26|      0.18|
|    Failing|                28|                      0.49|      0.93|
|       Good|               169|                      0.51|      1.02|
|   Moderate|               348|                      0.67|      1.42|
|        Low|               185|                      0.83|      1.85|
+-----------+------------------+--------------------------+----------+
```

Figure 37: Career concerns effect on students' stress output

3. **Is academic performance related to workload and teacher-student relationship?**
   Students who maintain good relationships with their teachers tend to experience lower study loads and stress levels while achieving higher academic performance. A positive teacher-student relationship can lead to increased motivation and engagement in learning. Teacher guidance plays a crucial role in efficient learning, helping students grasp concepts more quickly, thereby reducing the overall study burden and stress. This suggests that fostering strong teacher-student relationships can enhance academic success while mitigating stress and workload related challenges.

```scala
val query_3 =
"""
SELECT
    CASE
        WHEN academic_performance >= 0.0 AND academic_performance < 0.2 THEN 'Failing'
        WHEN academic_performance >= 0.2 AND academic_performance < 0.4 THEN 'Low'
        WHEN academic_performance >= 0.4 AND academic_performance < 0.6 THEN 'Moderate'
        WHEN academic_performance >= 0.6 AND academic_performance < 0.8 THEN 'Good'
        WHEN academic_performance >= 0.8 AND academic_performance < 1.0 THEN 'High'
        WHEN academic_performance = 1.0 THEN 'Excellent'
    END AS Performance,
    count(academic_performance) AS Nnumber_of_Students,
    round(avg(study_load),2) AS AVG_Study_Load,
    round(avg(teacher_student_relationship), 2) As AVG_Teacher_Student_Relationship,
    round(avg(stress_level), 2) AS AVG_Stress
FROM stressDF
GROUP BY Performance
ORDER BY AVG_Stress
"""
val result_3 = spark.sql(query_3)

// Query3:
println("Is Academic Performance related with study load and teacher_student_relationship ?")
result_3.show()
```

Figure 38: Teacher-student relationship effect on academic performance query

```
Is Academic Performance related with study load and teacher_student_relationship ?
+-----------+------------------+-------------+-------------------------------+----------+
|Performance|Nnumber_of_Students|AVG_Study_Load|AVG_Teacher_Student_Relationship|AVG_Stress|
+-----------+------------------+-------------+-------------------------------+----------+
|  Excellent|               182|         0.33|                           0.82|      0.15|
|       High|               188|         0.35|                           0.77|      0.18|
|    Failing|                28|         0.39|                           0.36|      0.93|
|       Good|               169|         0.51|                           0.49|      1.02|
|   Moderate|               348|         0.62|                           0.39|      1.42|
|        Low|               185|         0.76|                           0.32|      1.85|
+-----------+------------------+-------------+-------------------------------+----------+
```

Figure 39: Teacher-student relationship effect on academic performance output

4. **Does Social Support from Friends Help Reduce Stress and Improve Mental Health?**
   Social support is often a needed to fight against stress. We classified students into five categories based on the level of support they receive from friends; which are 'Very Low Support', 'Low Support', 'Moderate Support', 'High Support', 'Very High Support'.
   The analysis shows that students who report higher levels of social support from friends tend to have lower stress levels and better academic performance. Additionally, mental health issues are less frequent among students who feel supported. This suggests that enhancing peer connections and fostering inclusive, supportive environments may help improve student well-being and reduce stress-related challenges.

```
val query_4 =
"""
SELECT
    CASE
        WHEN social_support <= 0.2 THEN 'Very Low Support'
        WHEN social_support <= 0.4 THEN 'Low Support'
        WHEN social_support <= 0.6 THEN 'Moderate Support'
        WHEN social_support <= 0.8 THEN 'High Support'
        WHEN social_support <= 1 THEN 'Very High Support'
    END AS Friend_Support_Level,
    round(avg(stress_level), 2) AS AVG_Stress,
    round(avg(academic_performance), 2) AS AVG_Performance,
    round(avg(mental_health_history), 2) AS AVG_Mental_Health_Issues
FROM stressDF
GROUP BY
    CASE
        WHEN social_support <= 0.2 THEN 'Very Low Support'
        WHEN social_support <= 0.4 THEN 'Low Support'
        WHEN social_support <= 0.6 THEN 'Moderate Support'
        WHEN social_support <= 0.8 THEN 'High Support'
        WHEN social_support <= 1 THEN 'Very High Support'
    END
ORDER BY AVG_Stress
"""

val result_4 = spark.sql(query_4)
```

Figure 40: Social support's effect on stress and mental health query

```
Does Social Support from Friends Help Reduce Stress?
+--------------------+----------+---------------+------------------------+
|Friend_Support_Level|AVG_Stress|AVG_Performance|AVG_Mental_Health_Issues|
+--------------------+----------+---------------+------------------------+
|   Very High Support|      0.34|           0.76|                    0.19|
|        High Support|       1.0|           0.49|                    0.49|
|    Very Low Support|      1.02|           0.52|                    0.44|
|         Low Support|      1.71|           0.35|                    0.85|
+--------------------+----------+---------------+------------------------+
```

Figure 41: Social support's effect on stress and mental health output

5. **How Does Sleep Quality Influence Stress and Academic Outcomes?**
   Students with poor sleep quality consistently show higher levels of stress and mental health issues, along with lower academic performance. As sleep quality improves, stress levels decrease while performance improves. These results highlight the critical importance of good sleep and suggest that universities should provide awareness programs and practical strategies for better sleep to support student success and well-being.

```
val query_5 =
"""
SELECT
    CASE
        WHEN sleep_quality <= 0.2 THEN 'Very Poor Sleep'
        WHEN sleep_quality <= 0.4 THEN 'Poor Sleep'
        WHEN sleep_quality <= 0.6 THEN 'Moderate Sleep'
        WHEN sleep_quality <= 0.8 THEN 'Good Sleep'
        WHEN sleep_quality <= 1 THEN 'Excellent Sleep'
    END AS Sleep_Quality_Level,
    round(avg(stress_level), 2) AS AVG_Stress,
    round(avg(academic_performance), 2) AS AVG_Performance,
    round(avg(mental_health_history), 2) AS AVG_Mental_Health_Issues
FROM stressDF
GROUP BY
    CASE
        WHEN sleep_quality <= 0.2 THEN 'Very Poor Sleep'
        WHEN sleep_quality <= 0.4 THEN 'Poor Sleep'
        WHEN sleep_quality <= 0.6 THEN 'Moderate Sleep'
        WHEN sleep_quality <= 0.8 THEN 'Good Sleep'
        WHEN sleep_quality <= 1 THEN 'Excellent Sleep'
    END
ORDER BY AVG_Stress
"""

val result_5 = spark.sql(query_5)
```

Figure 42: Sleep quality's influence on stress and academic performance query

```
How Does Sleep Quality Influence Stress and Academic Outcomes?
+------------------+----------+--------------+----------------------+
|Sleep_Quality_Level|AVG_Stress|AVG_Performance|AVG_Mental_Health_Issues|
+------------------+----------+--------------+----------------------+
|    Excellent Sleep|      0.24|          0.79|                  0.12|
|         Good Sleep|      0.57|          0.68|                  0.27|
|         Poor Sleep|      0.96|          0.51|                  0.51|
|    Very Poor Sleep|      1.84|          0.33|                  0.91|
+------------------+----------+--------------+----------------------+
```

Figure 43: Sleep quality's influence on stress and academic performance output

6. Machine Learning

6.1 Problem Statement

In this study, we aim to predict students' **stress levels** using various psychological, social, and lifestyle-related indicators. The dataset includes 20 features such as anxiety levels, sleep quality, academic performance, and social support. This prediction task is framed as a **multiclass classification problem**, where the goal is to assign a stress category (e.g., low, medium, or high) based on the available input features.

6.2 Algorithm Selection and Justification

For this task, we selected the **Random Forest Classifier**, a robust ensemble learning algorithm that builds multiple decision trees and combines their outputs to improve classification accuracy and reduce overfitting. Random Forest is particularly suitable for our case for the following reasons:

- It performs well with datasets that have a large number of input features, as it can handle feature interactions effectively.
- It is capable of handling both categorical and numerical data.
- It provides feature importance scores, helping us interpret which factors contribute most to stress prediction.
- It is resistant to noise and overfitting due to the averaging nature of ensemble learning.

6.3 Implementation and Training

The implementation was carried out using Apache Spark's MLlib for scalable and efficient machine learning. The following steps summarize our training pipeline:

1. **Data Loading and Preparation:**

   The dataset cleaned_student_stress_factors.csv was loaded with automatic schema inference enabled. Since the data was already preprocessed and normalized before ingestion, no additional cleaning, encoding, or scaling was necessary. This allowed us to proceed directly to model training, ensuring consistent and reliable input feature values.

```
// Read data
val data_path = "cleaned_student_stress_factors.csv"
// val df = spark.read.format("csv").option("header", "true").load(data_path)

val data = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(data_path)
```

Figure 44 Loading the data

2. **Feature Vectorization**:

We have 20 relevant features related to psychological, environmental, and academic stressors. These features were combined into a single feature vector using VectorAssembler, which is a standard preprocessing step in Spark to convert multiple columns into one vector column for machine learning models.

```
val assembler = new VectorAssembler()
  .setInputCols(featureCols)
  .setOutputCol("features")
```

Figure 45 Feature vectorization

3. **Pipeline Construction**:

A Pipeline was created to chain the feature assembler and the RandomForestClassifier, ensuring a seamless and reproducible modeling workflow.

```
val rf = new RandomForestClassifier()
  .setLabelCol("label")
  .setFeaturesCol("features")

val pipeline = new Pipeline().setStages(Array(assembler, rf))
```

Figure 46 Pipline Construction

4. **Model Training with Hyperparameter Tuning**:

To enhance model performance, we implemented hyperparameter tuning using a grid search combined with 5-fold cross-validation. The following parameters were included in the search grid:

- numTrees: Number of decision trees in the random forest (tested values: 10, 20, 30).
- maxDepth: Maximum depth of each decision tree (tested values: 3, 5, 7).

The code snippet for the parameter grid is shown below:

```
val paramGrid = new ParamGridBuilder()
  .addGrid(rf.numTrees, Array(10, 20, 30))
  .addGrid(rf.maxDepth, Array(3, 5, 7))
  .build()
```

Figure 47 Hyperparameter Tuning

This setup produced nine different model configurations (3 tree depths × 3 tree counts), each evaluated across five folds of the training data. The model with the highest average accuracy across folds was selected as the best-performing model.

5. **Training and Evaluation**:

The dataset was split into 70% training and 30% testing subsets. The pipeline was fitted on the training data using the best configuration found during cross-validation. The model was then evaluated on the test set using accuracy as the primary metric.

```
val Array(trainingData, testData) = labeledData.randomSplit(Array(0.7, 0.3))

val cvModel = cv.fit(trainingData)
```

Figure 48 Training and Evaluation

6.4 Results and Analysis

After training and tuning the model, the following evaluation metrics were obtained on the test data:

Metric	Value
Accuracy	0.90
Precision	0.9002
Recall	0.90
Test Error	0.10

```
scala> println(s"Test Accuracy = $accuracy")
Test Accuracy = 0.9

scala> println(s"Test Recall = $r")
Test Recall = 0.9

scala> println(s"Test Precision = $p")
Test Precision = 0.900188690907335

scala> println(s"Test Error = ${1.0 - accuracy}")
Test Error = 0.09999999999999998
```

Figure 49 ML Results

```
scala> pivotMatrix.show()
+-----+---+---+---+
|label|0.0|1.0|2.0|
+-----+---+---+---+
|    1| 12|101|  5|
|    2| 18|  5|105|
|    0| 98|  3|  8|
+-----+---+---+---+
```

Figure 50 Confusion Matrix

The selected model was trained with:

- **Number of Trees (numTrees)**: 20
- **Maximum Tree Depth (maxDepth)**: 3

```
scala> val bestNumTrees = rfModel.getNumTrees
bestNumTrees: Int = 20

scala> val bestMaxDepth = rfModel.getMaxDepth
bestMaxDepth: Int = 3
```

Figure 51 Tuning Results

These values were chosen automatically through the parameter grid search process, as shown below:

Despite being relatively shallow (depth of 3), the selected model performed exceptionally well on the test set, demonstrating that even with a modest model complexity, Random Forest can effectively capture the patterns in the dataset. This also helps mitigate overfitting and ensures generalizability.

The high values for both precision and recall (each ~90%) indicate a balanced and reliable performance across stress level classes, meaning the model is both good at detecting stress correctly (recall) and avoiding false positives (precision).

These results suggest that the model is suitable for deployment in a real-world setting to support **early detection of stress** among students. Moreover, it lays a solid foundation for further exploration such as feature importance analysis, integration with a dashboard, or inclusion of real-time data inputs.

7. Extra: Cloud Deployment and Execution

To execute our machine learning pipeline in a scalable and efficient environment, we utilized **Amazon Web Services (AWS)**, specifically **Amazon EMR (Elastic MapReduce)**. AWS was selected due to its robust infrastructure, ease of integration with Apache Spark, and its seamless connectivity with Amazon S3 for data storage and retrieval. This choice allowed us to manage the training and evaluation of our model without the limitations of local processing power.

The model was deployed using an EMR cluster with the following configuration:

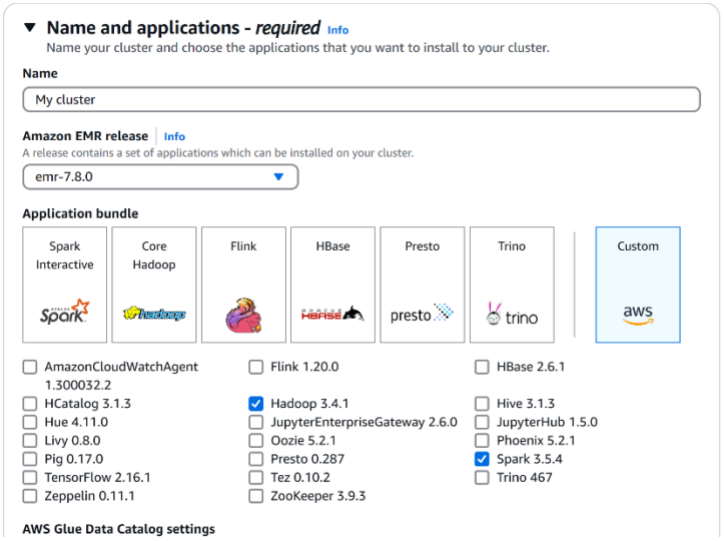

Figure 53 cluster configuration example

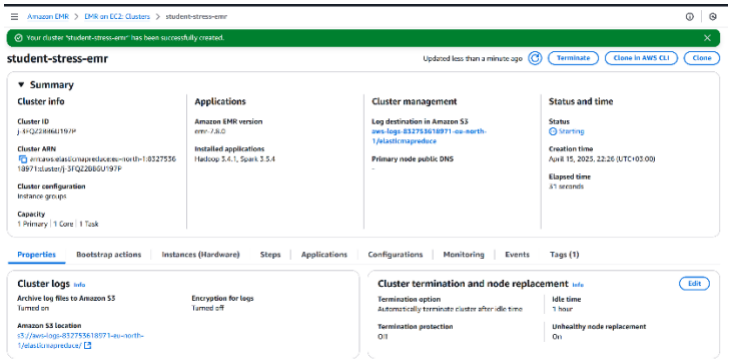

Figure 52 cluster created successfully

- **EMR Release Version**: 7.8.0
- **Applications Installed**: Apache Spark 3.5.4, Hadoop 3.4.1

- **Instance Types**:
    - 1 Primary node (m5.xlarge)
    - 1 Core node (m5.xlarge)
    - 1 Task node (m5.xlarge)
- **Storage**: 15 GiB General Purpose SSD (gp3)
- **Region**: Europe (Stockholm) - eu-north-1
- **Data Location**:

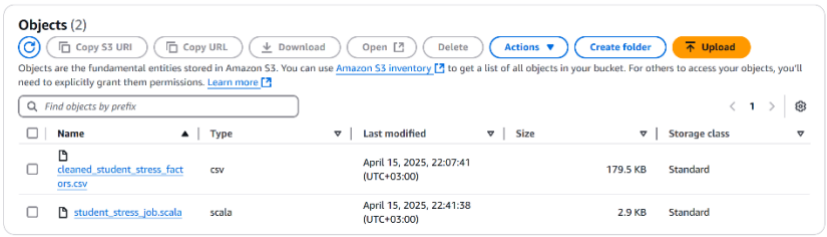

Figure 54 Data Location

Input dataset stored in S3:

s3://student-stress-analysis-bucket/cleaned_student_stress_factors.csv

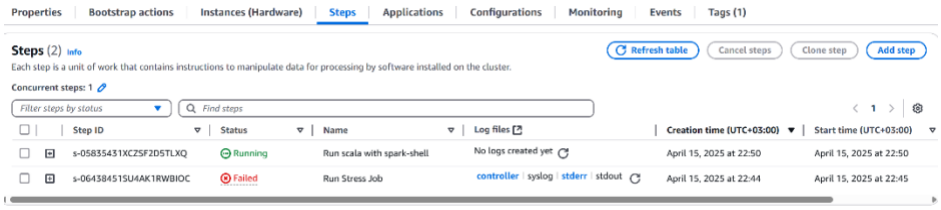

Figure 55 Finally Running!

We initially attempted to execute the job using the **Spark application step** with a direct .scala script via spark-submit. However, this approach failed, as .scala scripts are not supported for direct submission in that format. To resolve this issue, we reconfigured the step using a **Custom JAR step** and executed the script through the **spark-shell interface** via the command-runner.jar tool provided by AWS.

This modification successfully enabled the execution of our Random Forest pipeline, which involved training the model using cross-validation and a grid of hyperparameters. The results—including accuracy, test error, and the best-performing model configuration—were automatically written to the designated S3 path.

The file content is:

Test Accuracy: 0.9
Test Error: 0.1
Best Random Forest Model:
RandomForestClassificationModel: uid=RandomForestClassifier_xyz123, numTrees=20, numClasses=3, numFeatures=20

Overall, this cloud-based setup demonstrated the effectiveness of using distributed infrastructure for machine learning workflows. AWS EMR provided a reliable and scalable environment to process data and train models efficiently, making it a suitable platform for research and real-world deployment.

8. References

[1] "Student stress - NHS." Accessed: Feb. 03, 2025. [Online]. Available: https://www.nhs.uk/mental-health/children-and-young-adults/help-for-teenagers-young-adults-and-students/student-stress-self-help-tips/

[2] "Student Stress Factors: A Comprehensive Analysis." Accessed: Feb. 04, 2025. [Online]. Available: https://www.kaggle.com/datasets/rxnach/student-stress-factors-a-comprehensive-analysis