Machine learning assignment 1

Name	ID
Malak Alaa Mohamed	23012115
Maram Mohamed Mahmoud	23012197
Ganna Mohamed Abdul Fattah	23012138

Regression models:

Dataset: https://www.kaggle.com/datasets/larsen0966/student-performance-data-set
The dataset used in this analysis is the Student Performance Dataset
Which comes from Portuguese secondary schools and includes a mix of information about students — things like their age, family background, study habits, and school performance.
The main goal is to predict their **final grade (G3)**, which is influenced by both academic and personal factors
To do this, we explore a few different modeling approaches:

1. **Simple regression**
2. **Multiple Linear Regression**
3. **Polynomial Regression**

Data Exploration:

Before building any model, we first explored and cleaned the dataset to ensure it was reliable and ready for analysis. This step is crucial because poor data quality can lead to misleading results or underperforming models.

Step 1: Data Inspection

We began by loading the dataset and viewing the first few rows using df.head(). This gave us a quick snapshot of the data and helped identify what types of features we were working with

Next, we used df.info() and df.describe() to:

- Check for data types
- Understand basic statistics like means, minimums, and maximums.
- Detect any irregularities such as extreme outliers or unexpected data values.

Step 2: Handling Missing Values and Duplicates

We checked for missing values using df.isnull().sum() and for duplicates using df.drop_duplicates(inplace=True)

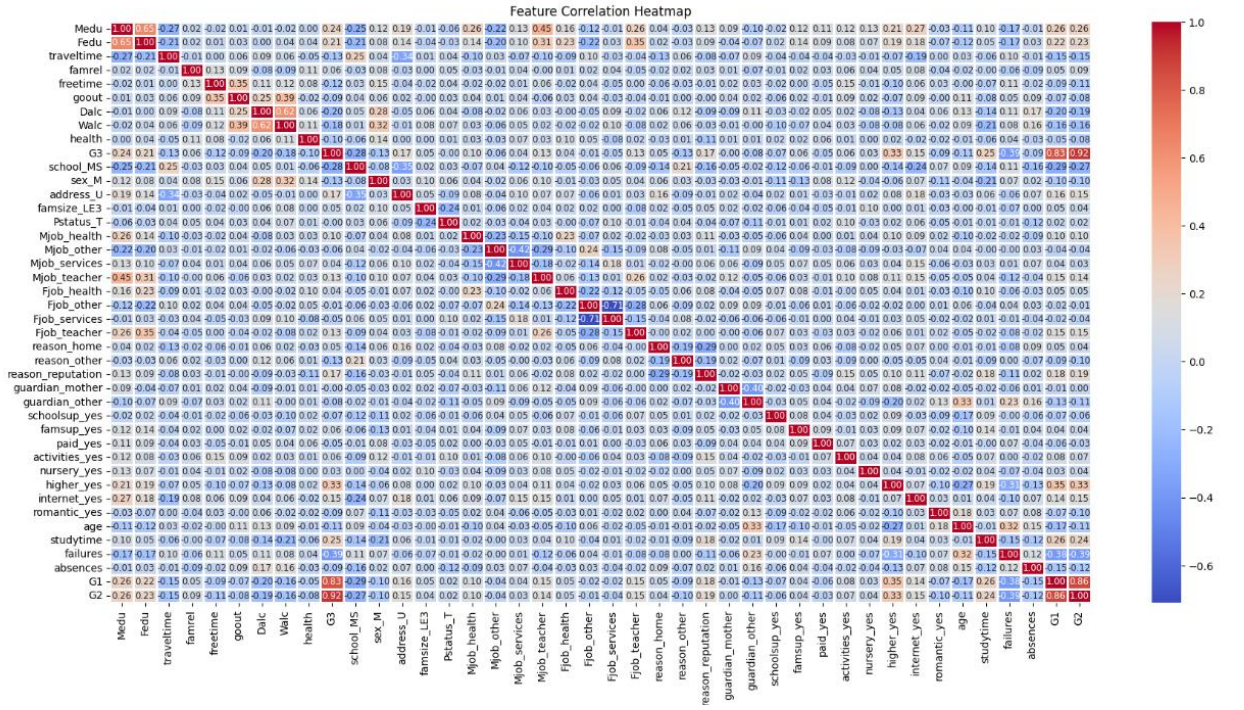
the dataset had no missing data and the duplicated values were dropped

Step 3: Visual Exploration

To better understand the relationships and patterns in the dataset, we used a variety of visualizations. Each one served a specific purpose in guiding our analysis and feature selection

1. Feature Correlation Heatmap

The **correlation analysis** played a crucial role in helping us decide which features to prioritize and how to visualize them. Understanding the relationships between features and the target variable (G3) guided us in focusing on the most relevant features during the EDA



From the correlation matrix, we identified that certain features, such as **G2** and **G1**, had strong positive correlations with G3. These features became the **primary focus** of our visualizations since they had the most direct influence on the target variable. By highlighting the relationships between G2/G1 and G3, we could visualize how performance in previous grades affects the final grade

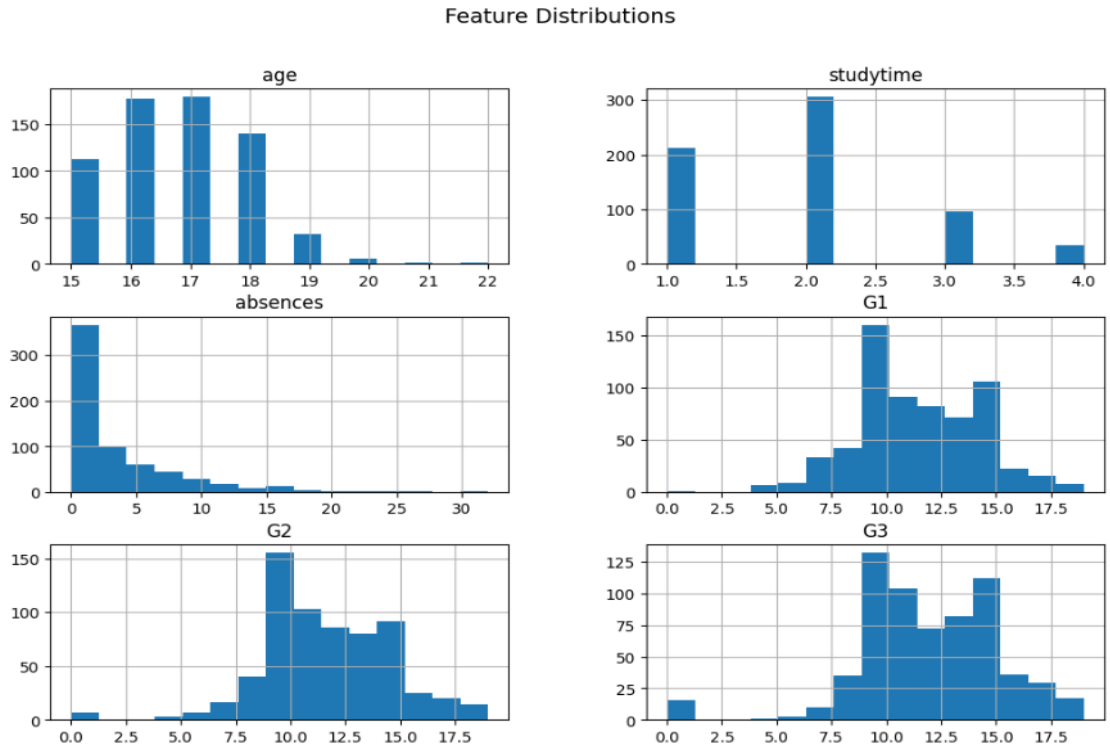
2. Histogram

We used histograms to explore the distribution of numeric features such as age, studytime, absences, G1, G2, and G3.

Purpose:

- To see if the data is skewed, normally distributed, or has outliers.
- Understand how common or rare certain values are

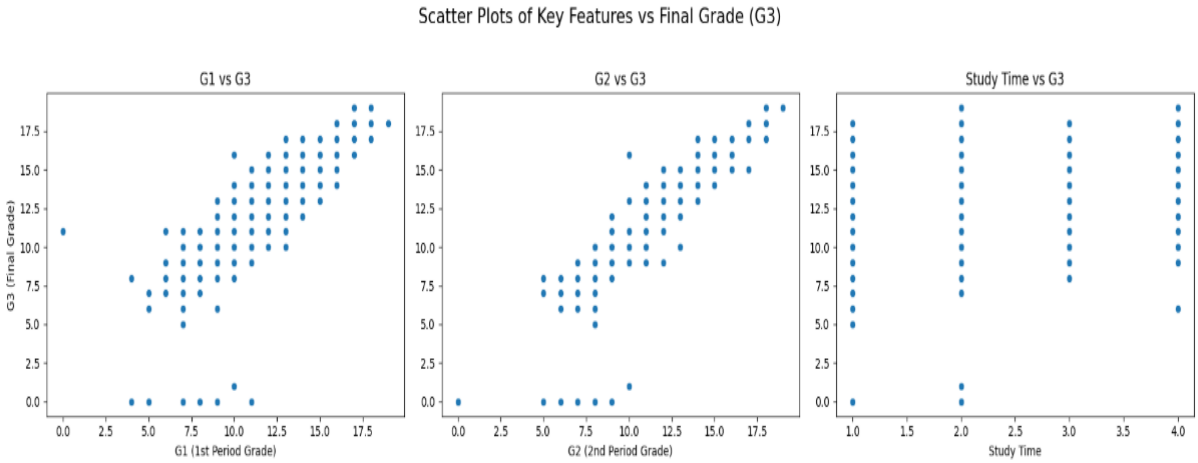
What we observed:



- Grades (G1, G2, G3) were fairly normally distributed, but many students scored low.
- Absences had a long tail, indicating some students missed many classes

3. Scatter Plots

We plotted scatter plots of G1 vs G3, G2 vs G3, and study time vs G3



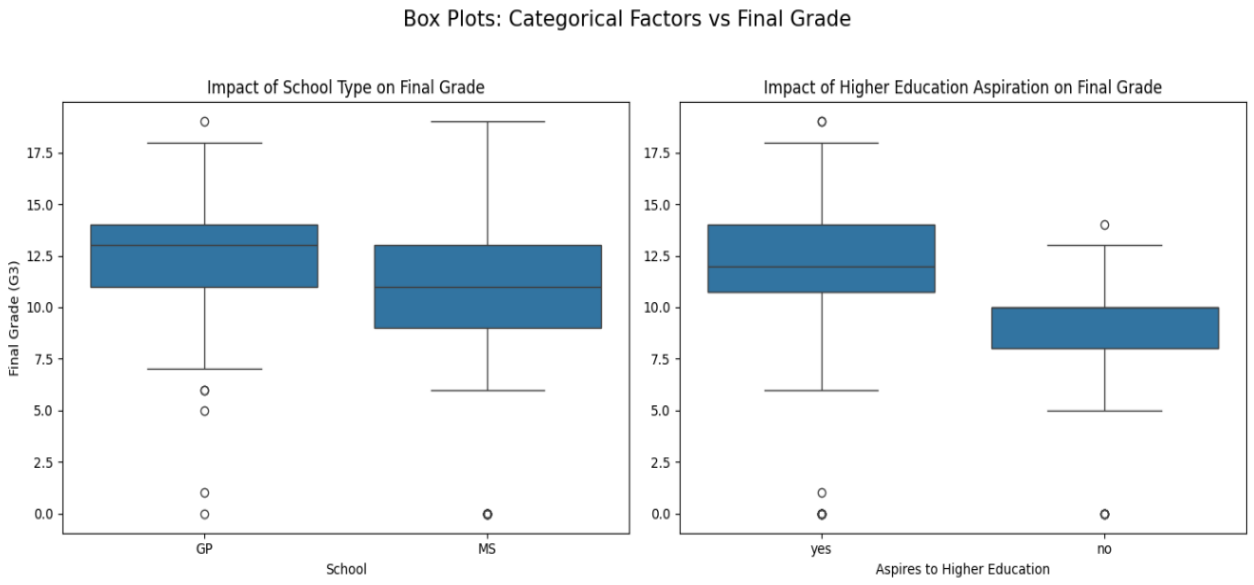
Purpose:

- To identify relationships or correlations between two numeric variables.
- To visually assess how changes in one feature may affect the final grade

What we observed:

- There was a **strong linear relationship between G2 and G3**, suggesting that second-period performance is a good predictor of the final grade.
- A moderate correlation between G1 and G3.
- The relationship between study time and G3 was weak, suggesting that more study time doesn't always guarantee better results

4. Box Plots



We used box plots to examine how the final grade (G3) varied across categories like school and higher (whether students plan to pursue higher education).

Purpose:

- To see if categorical features impact the target variable.
- To identify whether students from one school perform better than another or whether aspirations for higher education influence grades.

What we observed:

- Students who expressed interest in higher education tended to have higher final grades.
- The difference in performance between schools wasn't very significant, but the spread of grades varied.

Preprocessing:

Preprocessing is an essential step before applying any machine learning model. It ensures that the dataset is clean, consistent, and suitable for analysis

Step 1: Encoding Categorical Variables

Many columns in the dataset are categorical (like school, sex, address, etc.). Since machine learning models work with numerical data, we used **One-Hot Encoding** to convert these categories into numeric form.

```
# Identify categorical columns
categorical_features = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                        'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                        'nursery', 'higher', 'internet', 'romantic']

# Apply One-Hot Encoding
encoder = OneHotEncoder(drop="first", sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical_features]) # applies the transformation to our actual data.

# Convert encoded data back to a DataFrame
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical_features))

# Drop original categorical columns and merge encoded features
df = df.drop(columns=categorical_features).reset_index(drop=True)
df = pd.concat([df, encoded_df], axis=1)
```

- **OneHotEncoder(drop="first"): This transforms each categorical column into several binary columns**
- **We drop the original categorical columns (now replaced by encoded ones)**
- **Merge the newly encoded DataFrame with our main dataset**

Step 3: Feature Selection Based on Correlation

We calculated the **correlation matrix** to identify which features are most related to the target variable G3

This helped us **reduce dimensionality** and focus on features that matter the most for predicting final grades.

```
# Calculate correlation between all features and G3
correlation_matrix = df.corr()

# Sort features based on correlation with G3
correlations = correlation_matrix["G3"].abs().sort_values(ascending=False)
print(correlations)
```

Step 4: Dropping Less Relevant Features

After reviewing the correlation values and visualizations, we removed some weak predictors that had little to no relationship with the target variable.

```
# removes features with weak relationships to the target
df = df.drop(columns=['Fjob_other', 'guardian_mother', 'Pstatus_T', 'nursery_yes', 'Fjob_health', 'Mjob_services', 'reason_home', 'famsize_LE3'])
```

Step 5: Defining Target and Features

we separated the dataset into the feature matrix X and target variable y

Then we split the data into training and testing sets

Splitting ensures we can test the model on unseen data to evaluate performance accurately

```
# Define features X and target y
X = df.drop(columns=["G3"])  # G3 is the target
y = df["G3"]

# Split the dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Simple Linear Regression Model:

Simple Linear Regression helps us **understand the relationship between two variables** one **independent variable** and one **dependent variable**

Step 1: Selecting Features

```
X = df['G2'] # Selecting G2 as the strongest predictor(because it has the highest correlation with G3)
y = df['G3'] # Selecting G3 as the Target variable
```

X is what we'll use to make predictions — in this case, the **second-period grade (G2)**.
y is what we're trying to predict — the **final grade (G3)**.
We chose G2 because it had the **strongest correlation** with G3.

Step 2: Splitting the Dataset

Splitting the data helps us: Check if the model is actually learning or just memorizing.
See how it will perform on real-world data.
Improve the model by understanding its strengths and weaknesses

```
# Define features X and target y
X = df.drop(columns=["G3"])  # G3 is the target
y = df["G3"]

# Split the dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- X_train, y_train: used to train the model.
- X_test, y_test: used to test how well the model performs.
- test_size=0.2 means 20% of the data is used for testing.
- random_state=42 ensures reproducibility

Step 3: Training the Model

the most important step where the machine learns from the data

By training the model the machine

Looks at the **input data** (like previous grades, study time, absences)
Compares it to the **correct answers**

Learn the **relationship** between them

```
# Train on the strongest single feature
model_simple = LinearRegression().fit(X_train[['G2']], y_train)
```

Step 4: Making Predictions and Evaluating

```
# Evaluate
y_pred= model_simple.predict(X_test[['G2']])
print(f"Simple Linear Regression:")
print(f"  R²: {r2_score(y_test, y_pred):.3f}")
print(f"  MSE: {mean_squared_error(y_test, y_pred):.3f}")
```

We use the trained model to predict final grades (G3) based on the test set's G2 values**.**

R^2 shows how much of the variation in G3 can be explained by G2. Closer to 1 = better.

MSE tells us the average squared error between the actual and predicted grades. Lower = better.

Step 5: Plotting the Regression Line

This model helps us:

Understand the linear relationship between two continuous variables.
Establish a simple predictive baseline before trying more complex models.

- **Multiple Linear Regression Model**

The difference here is that we working with many features not only the most correlated one but also with other correlated features

1- Additional pre-processing phase

Includes how can we choose the most important and significant features whether numerical or categorical

The numerical features has been extracted and filtered In the first pre-processing phase in simple linear regression

The categorical features have many ways to define the most significant ones like

ANOVA (Analysis of Variance) tests whether the means of the target variable (G3 grades) differ significantly across categories (school, higher ,...etc). It quantifies whether a categorical feature has a meaningful impact on the target.

It produces the f_stat and p_value

What we really looking for here is the p_value as :

p-value: Probability that observed differences occurred by chance.

$p < 0.05 \rightarrow$ Feature is significant (groups are not equal).

To start using ANOVA we import the necessary libraries from **scipy**

```
from scipy.stats import f_oneway # to start using ANOVA
```

I start getting the most significant features using the following :

1- Making an empty list named 'Significant features' to store the categorical features with p_value < 0.05
2- Making a for loop to walk on each (value) or category in the categorical features using **.unique ()** to get each unique value and **(inner loop)** to create g3 (target) for each category in the categorical feature and after the inner loop we append the feature in the list **(outer loop)**

After these steps we have the categorical feature in list ‘significant features’

```
# testing categorical features to know what to keep and what to drop
significant_features = []

for feature in categorical_features:
    feature_values = df2[feature].unique()
    groups = [df2[df2[feature] == value ]['G3'] for value in feature_values]
    #run anova
    f_stat, p_value = f_oneway(*groups)
    if p_value < 0.05:
        significant_features.append(feature)

print (significant_features)
```

Next

We start to filter the df that has been encoded and filtered in the simple linear regression we keep the numeric features and extract the categories that is not a set or a subset of the ‘significant features’-> the most significant categorical features so the criteria is selecting :

1. Numerical features , AND
2. Encoded columns of significant features list that has been produces from anova test('school', 'higher')

We faced a problem while extracting because the list significant features contains the whole category -> feature (‘Higher’) and the df has the category+value (‘higher_yes’)

So it has been solved using method **.startwith(feat , ‘_’)** and if the value start_with any category in the significant feature list we take it in the selected cols from the df

```
# Ready the training set: extract encoded significant features + numerical features
import re
from sklearn.model_selection import train_test_split
```

```
# Get all columns from the encoded DataFrame
all_columns = df.columns.tolist()
```

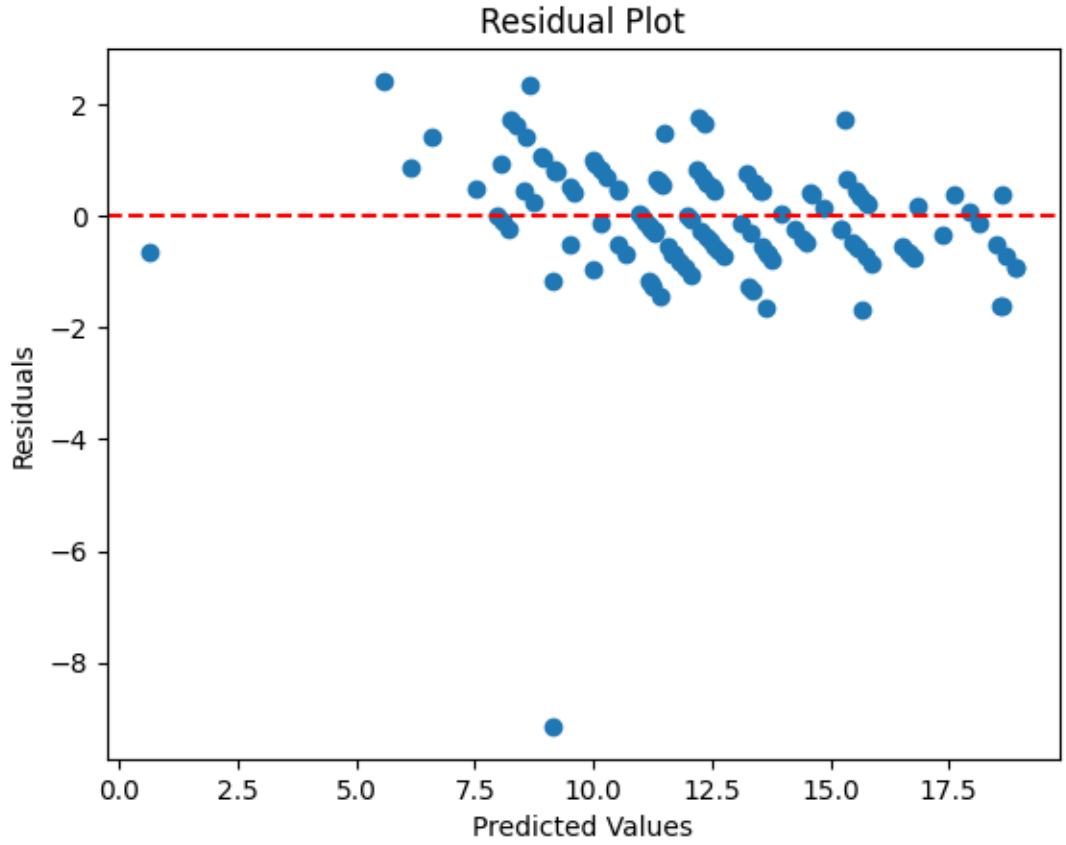
```
# Selecting
# 1. Numerical features , AND
# 2. Encoded columns of significant features list that has been produces from anova test('school',
'higher')
selected_columns = [
   col for col in all_columns
   if (
      col in numeric_features  # Keep numerical features
      or any(col.startswith(feat + '_') for feat in significant_features  # Keep encoded significant
categoricals
   )
   )
]

# Filter the DataFrame
X = df[selected_columns]
y = df['G3']

# Corrected train_test_split syntax (parentheses and assignment)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
print('Processing successful! Ready for training.')
print(f"Training set shape: {X_train.shape}, Test set shape: {X_test.shape}")
```

Next

We split the data into training and testing then evaluate the model using r2 score , mean square error , mean absolute error

```
97]

mean absolute error :  0.7588196489677822

mean squared error :  1.3890696806422826

R2 Scorre :  0.8575563845364519
```

We made a residual plot to see if the data follows patterns or not and the plot indicated that there is no nonlinearity and the data doesn't follow patterns



- ## Polynomial Regression Model

For the polynomial phase we use the most significant feature means the high correlated one and in our data set we discovered that G2 is the most correlated one with high positive correlation = 0.92
So we only need the G2 col = feature in our training data and the target G3
We need the following steps :

1- Scale
2- Transform feature into polynomial feature
3- Fit / train
4- Evaluate

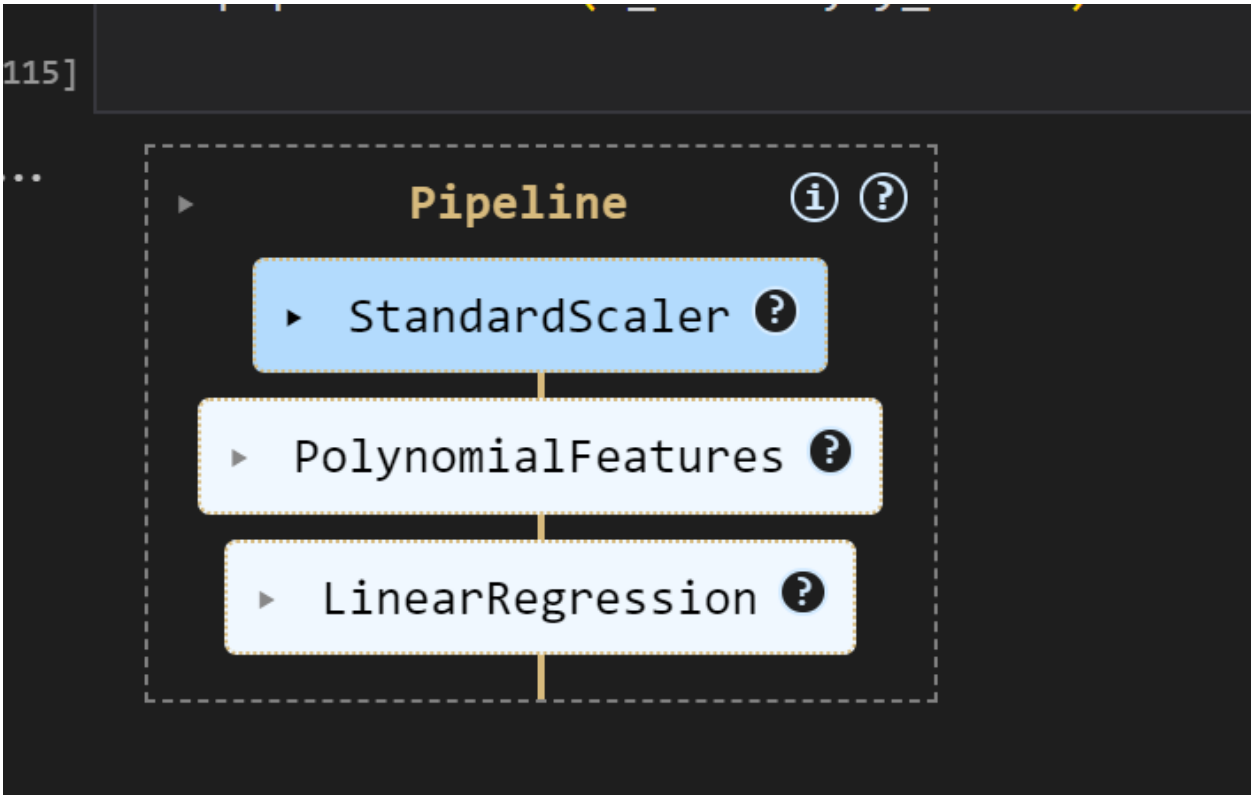
To make the process easier we used pipeline and polynomial features for transforming also standard scaler , linear regression for training the model

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Firstly we take an instance from make_pipeline , **then** order the steps inside it

Scale -> transform to poly feature -> linear regression then passing the training and testing data

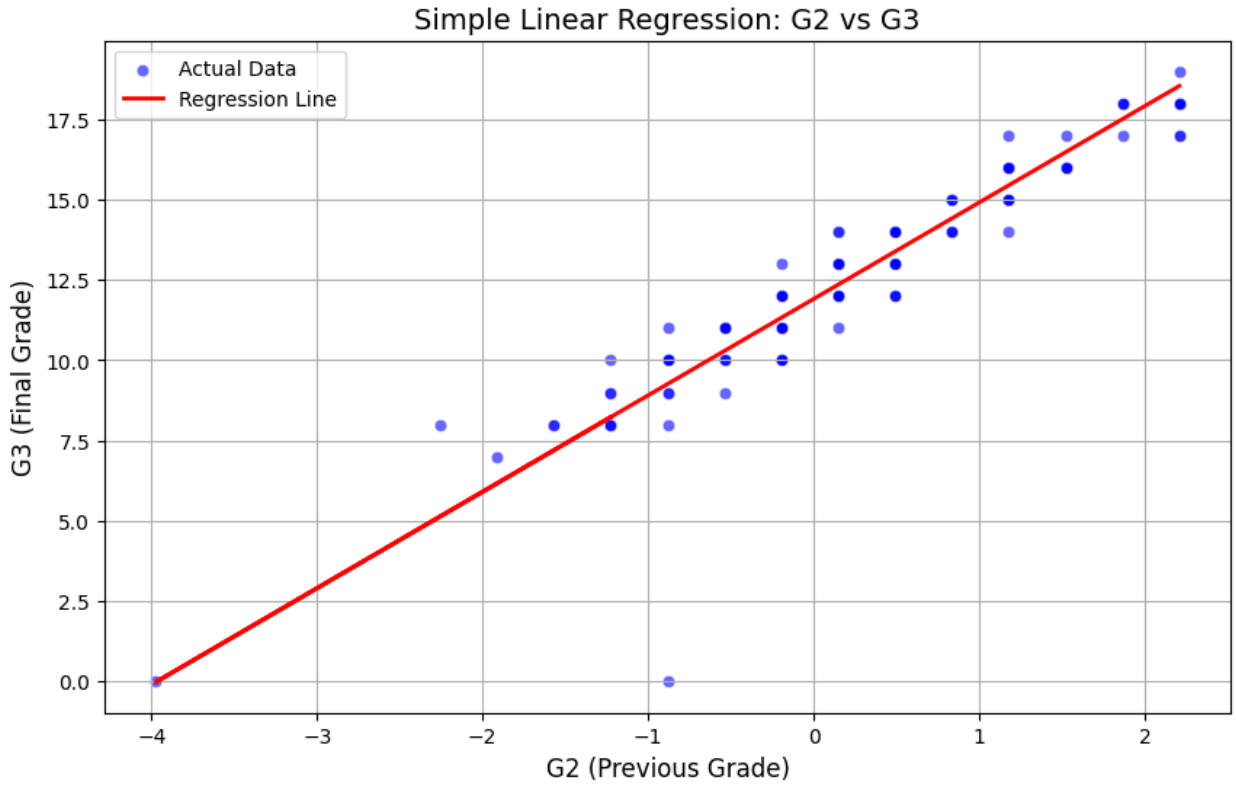
```
#coming back to the filtered and encoded data set -> df to get the X and Y
X = df[['G2']] # to make it data frame not a series -> important for sklearn
Y = df['G3']


X_train , X_test , y_train , y_test = train_test_split(X ,y , test_size = 0.20 , random_state = 42)


#use pipelineto make the process ordered well and easier
#inside pipeline we do the follwing :
#1- scale
#2- make polynomial features
#3- do the regression
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(
   StandardScaler(),
   PolynomialFeatures(degree= 2),
   LinearRegression()
)

pipeline.fit(X_train , y_train)
```



We Applied this 3 times with diff degrees 2 , 3 , 4 then training the model with the instance name .fit(x_train , y_train)
And evaluating the model using .scorr(x_test , y_test)

- **Comparing Between Models**

1- Linear regression model



The model fits the data perfectly

Simple Linear Regression:

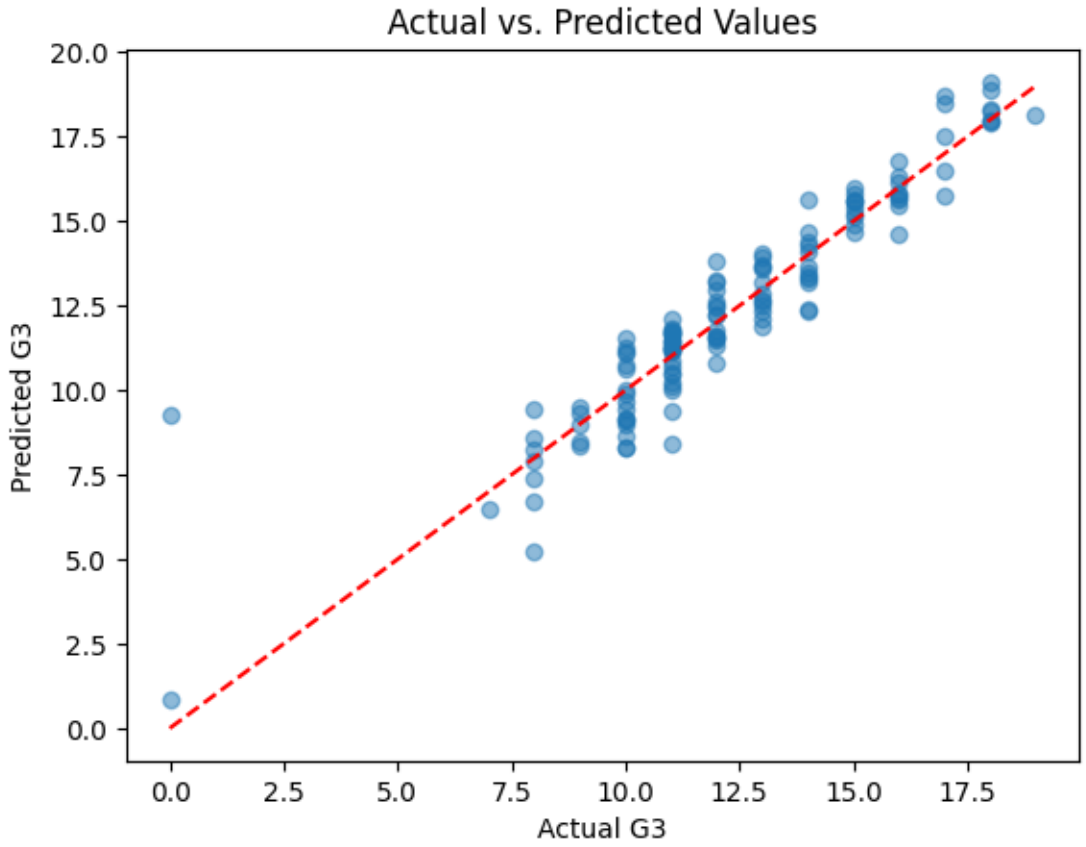
R^2: 0.869

MSE: 1.279

Which is the best score between them all with diff = 0.01

- **Multiple linear Model**

We discovered that the model fits the data also well here



mean absolute error : 0.7588196489677822

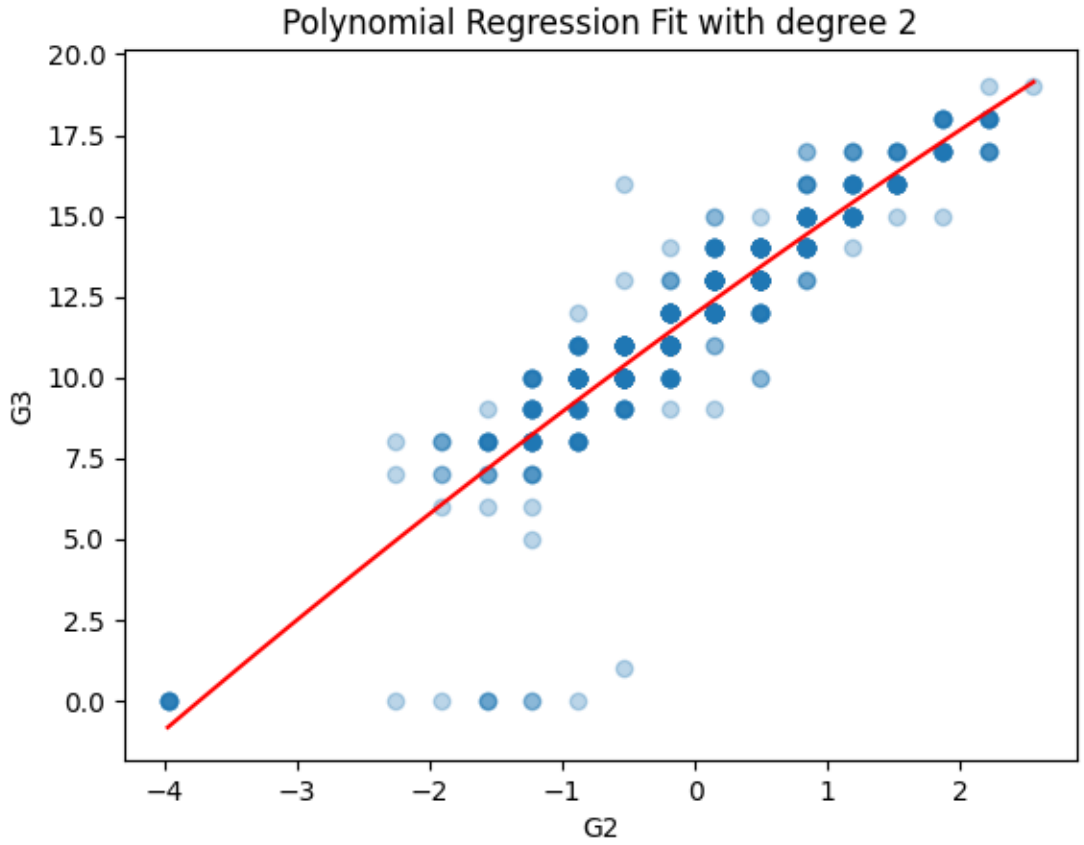
mean squared error : 1.3890696806422826

R2 Scorre : 0.8575563845364519

There is a small diff which indicates that the data works better with the simple linear regression model

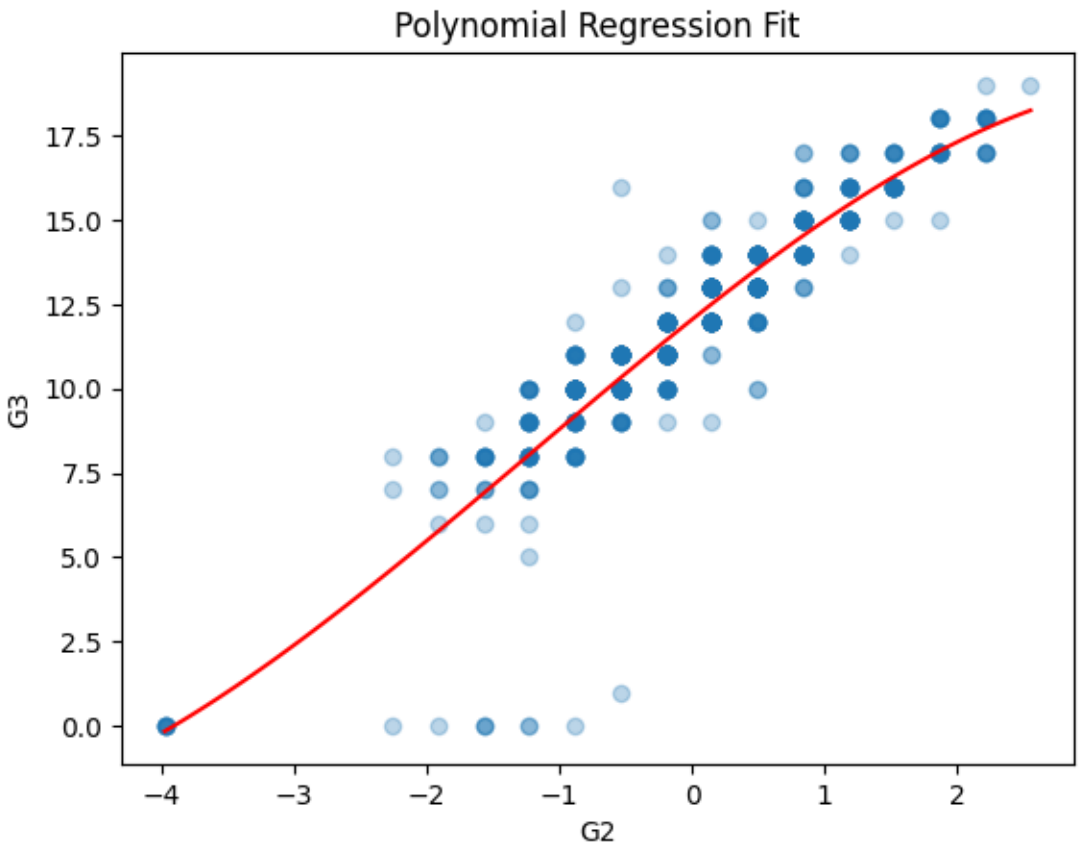
- **Polynomial Regression Models**

1- With degree = 2



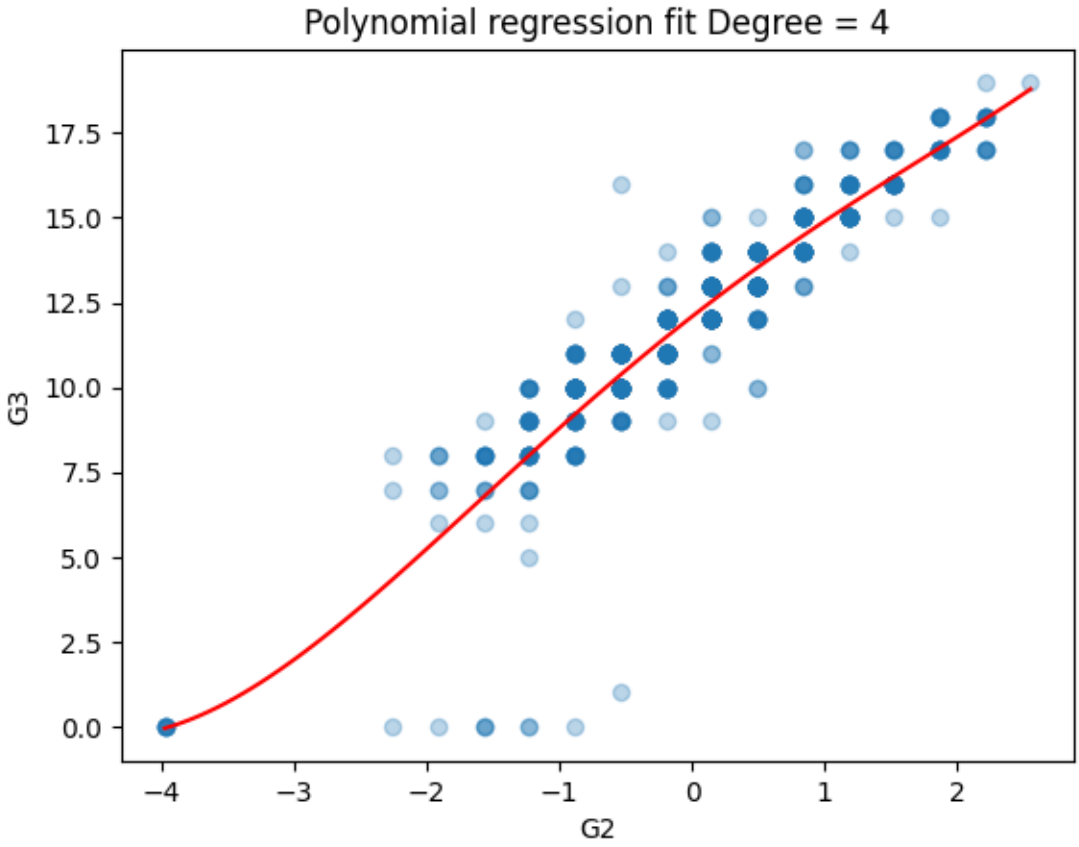
There are small data out of the line here but also the model works good with test score = 0.868 , Which is the closest one to the simple linear regression model with diff = 0.01

2- With degree = 3



the diff that we increased the degree and it doesn't affect significantly but the model still works good with test score = 0.864 which indicated that the score decreased by 0.02

3- With degree = 4



Also doesn't affect on the model performance significantly with test score = 0.861 which indicated that the score has decreased by 0.03 so after all of this we discovered that all models works well with the data but each time we increase the degree In polynomial regression the score decreases but small values but it decreases , and the best fit for this data is the Simple Linear Regression model .

- **Logistic Regression Model**

Dataset used in this model:
https://www.kaggle.com/datasets/chanchalagorale/employees-stress-level-dataset?resource=download
This dataset contains 3,000 synthetic records representing software employees working across various states in India. The goal is to analyze factors influencing stress levels based on attributes like working hours, job satisfaction, sleep, exercise, social habits, and work environment.

The following is the steps used for this model:

1. Data Exploration and Cleaning:
Before building any model, we first explored and cleaned the dataset to ensure it was reliable and ready for analysis. This step is crucial because poor data quality can lead to misleading results or underperforming models. We removed the Employee ID column, as it served as a unique identifier and didn't provide any predictive value for the Stress_Level. Duplicates were checked and missing values were accounted for to ensure data consistency.

Loading the Dataset

```
[241]: df = pd.read_csv(r"C:\Users\ganna\OneDrive\Desktop\data.csv")
print("Dataset Preview:")
df.head(5)
```

```
Dataset Preview:
```

[241]:

	Employee_Id	Avg_Working_Hours_Per_Day	Work_From	Work_Pressure	Manager_Support	Sleeping_Habit	Exercise_Habit	Job_Satisfaction	Work_Life_Balance	Social_
0	EMP0001	6.7	Home	3	4	4	2	5	No	
1	EMP0002	6.9	Home	1	1	5	4	3	Yes	
2	EMP0003	15.1	Office	3	2	3	5	1	No	
3	EMP0004	10.7	Hybrid	3	2	5	3	2	No	
4	EMP0005	11.8	Home	2	2	4	1	5	No	

Removing the Employee ID Column

As the ID columns are irrelevant for model training and could introduce noise.

[127]:
```
# removing employee id column
df = df.drop(df.columns[0], axis=1)
df.head()
```

[127]:

	Avg_Working_Hours_Per_Day	Work_From	Work_Pressure	Manager_Support	Sleeping_Habit	Exercise_Habit	Job_Satisfaction	Work_Life_Balance	Social_Person	Lives_
0	6.7	Home	3	4	4	2	5	No	5	
1	6.9	Home	1	1	5	4	3	Yes	3	
2	15.1	Office	3	2	3	5	1	No	2	
3	10.7	Hybrid	3	2	5	3	2	No	1	
4	11.8	Home	2	2	4	1	5	No	5	

Checking for Duplicated Rows and Missing Values

[128]:
```
# checking for duplicated rows
print(df.duplicated().sum())
```

```
0
```

[129]:
```
# checking for missing values
print(df.isnull().sum())
```

```
Avg_Working_Hours_Per_Day    0
Work_From                    0
Work_Pressure                0
Manager_Support              0
Sleeping_Habit               0
Exercise_Habit               0
Job_Satisfaction             0
Work_Life_Balance            0
Social_Person                0
Lives_With_Family            0
Working_State                0
Stress_Level                 0
dtype: int64
```

Encoding Categorical Variable

To prepare the dataset for logistic regression, we encoded categorical variables into numeric values using LabelEncoder. This step ensures that the model can process and interpret non-numeric columns, as logistic regression requires numerical inputs for its computations.

[130]:
```
# encode categorical variables
label_encoder = LabelEncoder()
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

Correlation Matrix

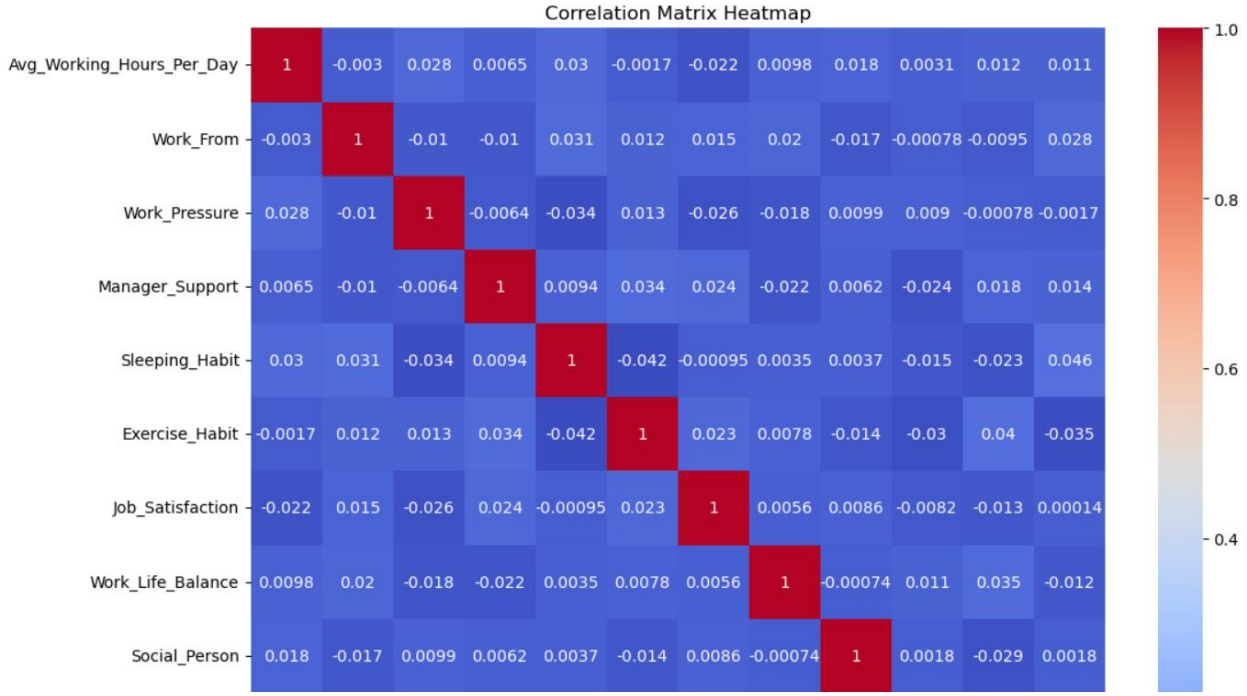
We computed and visualized the correlation matrix to understand relationships between numerical variables. This step is essential for identifying features with weak correlation to the target variable, which could negatively affect the model's performance. However, I couldn't conclude a whether or not a variable was better suited than another with this specific dataset.

```
[131]: corr_matrix = df.corr()
       corr_matrix
```

[131]:

	Avg_Working_Hours_Per_Day	Work_From	Work_Pressure	Manager_Support	Sleeping_Habit	Exercise_Habit	Job_Satisfaction	Work_Life_Bal
Avg_Working_Hours_Per_Day	1.000000	-0.003008	0.027801	0.006462	0.029929	-0.001651	-0.022256	0.00
Work_From	-0.003008	1.000000	-0.010137	-0.010406	0.031236	0.011868	0.014880	0.02
Work_Pressure	0.027801	-0.010137	1.000000	-0.006351	-0.034359	0.013189	-0.026346	-0.01
Manager_Support	0.006462	-0.010406	-0.006351	1.000000	0.009377	0.033863	0.023788	-0.02
Sleeping_Habit	0.029929	0.031236	-0.034359	0.009377	1.000000	-0.042258	-0.000945	0.00
Exercise_Habit	-0.001651	0.011868	0.013189	0.033863	-0.042258	1.000000	0.023153	0.00
Job_Satisfaction	-0.022256	0.014880	-0.026346	0.023788	-0.000945	0.023153	1.000000	0.00
Work_Life_Balance	0.009790	0.020324	-0.018261	-0.021806	0.003479	0.007792	0.005584	1.00
Social_Person	0.017985	-0.016822	0.009903	0.006213	0.003731	-0.014120	0.008645	-0.00
Lives_With_Family	0.003089	-0.000776	0.008997	-0.024494	-0.014506	-0.030415	-0.008194	0.01
Working_State	0.012249	-0.009473	-0.000784	0.017628	-0.023062	0.039754	-0.012849	0.03
Stress_Level	0.011047	0.028026	-0.001650	0.014309	0.045525	-0.034788	0.000144	-0.01

Visualized Matrix



Separating Features and Target and Splitting the Dataset

The dataset was split into training and testing sets, with 80% allocated for training and 20% for testing. This division helps evaluate the model's ability to generalize on unseen data, ensuring reliable predictions in real-world scenarios.

[262]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

[263]:
```
# separating features (X) and target variable (Y)
X = df.drop('Stress_Level', axis=1)
Y = df['Stress_Level']
```

[264]:
```
# splitting thw data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Building and Training the Logistic Regression Model

We trained a logistic regression model using the training set.

[265]:
```
# building and training a Logistic Regression model
lr_multiple = LogisticRegression()
lr_multiple.fit(X_train, Y_train)
```

Getting Coefficients and Intercept

Helps understand how each feature contributes to the prediction.

[266]:
```
lr_multiple.coef_
```

[266]:
```
array([[-0.00536061, -0.05154667,  0.03593249, -0.04796006, -0.02683676,
         0.03322575,  0.0058842 , -0.00994625,  0.01331281,  0.13453825,
         0.02292682],
       [ 0.00705104, -0.01960363, -0.01977333,  0.00625223, -0.02908334,
         0.07119281,  0.02382587, -0.13812893, -0.02957162, -0.0489833 ,
         0.02645998],
       [-0.00445262,  0.01221593, -0.00983206,  0.02033951, -0.01277632,
        -0.03381998, -0.01489046,  0.04357436,  0.01472115, -0.11085155,
         0.00721558],
       [-0.02371208,  0.0151872 ,  0.00404932, -0.01787733,  0.03414443,
         0.00888924, -0.02341082,  0.24010105,  0.00700585, -0.03229788,
        -0.0586722 ],
       [ 0.02647426,  0.04374717, -0.01037641,  0.03924566,  0.03455199,
        -0.07948782,  0.00859122, -0.13560023, -0.0054682 ,  0.05759448,
         0.00206982]])
```

[267]:
```
lr_multiple.intercept_
```

[267]:
```
array([-0.0335146 , -0.04024322,  0.11906546,  0.16952043, -0.21482806])
```

Making Predictions

Predicts class labels for the test dataset using the trained model.

[268]:
```
# predicting the class labels for the test data
Y_pred = lr_multiple.predict(X_test)
Y_pred
```

[268]:
```
array([1, 2, 5, 2, 2, 5, 4, 4, 5, 4, 5, 2, 3, 4, 1, 1, 1, 4, 2, 2, 1, 1,
       4, 1, 5, 2, 2, 5, 2, 3, 5, 5, 5, 5, 5, 2, 5, 2, 5, 2, 5, 5, 2, 1,
       4, 4, 4, 1, 4, 4, 1, 1, 1, 1, 3, 2, 4, 1, 3, 2, 1, 4, 2, 5, 5, 5,
       5, 1, 5, 2, 4, 4, 2, 5, 4, 2, 1, 2, 5, 4, 4, 1, 1, 5, 2, 5, 4, 4,
       5, 3, 1, 1, 2, 1, 4, 4, 5, 2, 5, 2, 2, 5, 4, 5, 1, 2, 5, 2, 2, 2,
       5, 4, 2, 4, 1, 4, 5, 2, 4, 1, 3, 5, 2, 4, 5, 1, 1, 5, 1, 5, 2, 2,
       5, 2, 1, 5, 1, 4, 2, 4, 2, 5, 1, 2, 5, 2, 5, 2, 1, 2, 1, 4, 4, 4,
       5, 2, 4, 4, 4, 5, 5, 5, 2, 2, 4, 1, 5, 4, 1, 5, 4, 5, 5, 4, 5, 1,
       2, 2, 1, 5, 5, 2, 1, 5, 4, 1, 2, 2, 5, 4, 2, 1, 5, 5, 4, 4, 2, 5,
       5, 4, 5, 1, 4, 2, 2, 1, 5, 4, 1, 4, 1, 2, 5, 5, 1, 2, 5, 5, 2, 4,
       2, 1, 5, 5, 4, 2, 4, 4, 4, 5, 2, 2, 5, 5, 4, 1, 2, 2, 2, 4, 4, 4,
       5, 1, 3, 2, 4, 5, 4, 4, 2, 5, 2, 2, 2, 2, 4, 3, 3, 2, 5, 2, 1, 5,
       4, 2, 2, 2, 4, 4, 2, 5, 4, 4, 2, 5, 5, 2, 2, 2, 1, 5, 2, 5, 3, 1,
       2, 2, 5, 2, 1, 3, 1, 2, 5, 1, 5, 1, 5, 2, 4, 5, 2, 5, 4, 2, 4, 1,
       4, 2, 1, 2, 2, 3, 5, 2, 5, 3, 2, 2, 1, 2, 4, 1, 3, 5, 2, 4, 2, 2,
       2, 2, 4, 2, 1, 3, 5, 1, 4, 1, 4, 2, 4, 2, 2, 4, 1, 5, 3, 4, 5, 3,
       2, 1, 1, 5, 1, 1, 1, 5, 2, 5, 2, 3, 4, 2, 2, 1, 2, 4, 1, 2, 5, 1,
       5, 2, 2, 5, 5, 2, 1, 2, 1, 5, 5, 4, 2, 1, 5, 1, 5, 4, 5, 5, 4, 1,
       2, 3, 5, 5, 2, 2, 4, 4, 5, 1, 4, 3, 5, 4, 4, 1, 4, 5, 1, 2, 2, 1,
       3, 1, 4, 4, 4, 2, 5, 1, 5, 2, 5, 3, 4, 4, 5, 1, 2, 2, 2, 2, 5, 1,
       5, 2, 5, 5, 4, 3, 4, 2, 1, 2, 1, 1, 4, 1, 5, 1, 4, 1, 4, 5, 5, 5,
       2, 1, 2, 5, 2, 4, 1, 5, 2, 1, 2, 5, 5, 4, 2, 4, 1, 2, 1, 1, 4, 1,
       5, 5, 2, 5, 4, 1, 5, 4, 4, 4, 5, 4, 3, 1, 4, 4, 4, 5, 2, 5, 2, 5,
       4, 5, 5, 4, 4, 5, 5, 1, 2, 2, 4, 1, 5, 1, 5, 2, 1, 2, 2, 1, 1, 5,
       2, 4, 2, 4, 2, 2, 1, 1, 1, 1, 2, 4, 4, 4, 5, 4, 2, 5, 4, 1, 4, 1,
       3, 4, 2, 5, 1, 1, 5, 5, 5, 2, 2, 1, 2, 5, 4, 4, 5, 5, 2, 2, 5, 2,
       2, 5, 4, 4, 1, 5, 4, 5, 2, 1, 4, 2, 1, 4, 1, 1, 5, 5, 1, 5, 5, 2,
       4, 1, 4, 4, 1, 2], dtype=int64)
```

Evaluating the Model

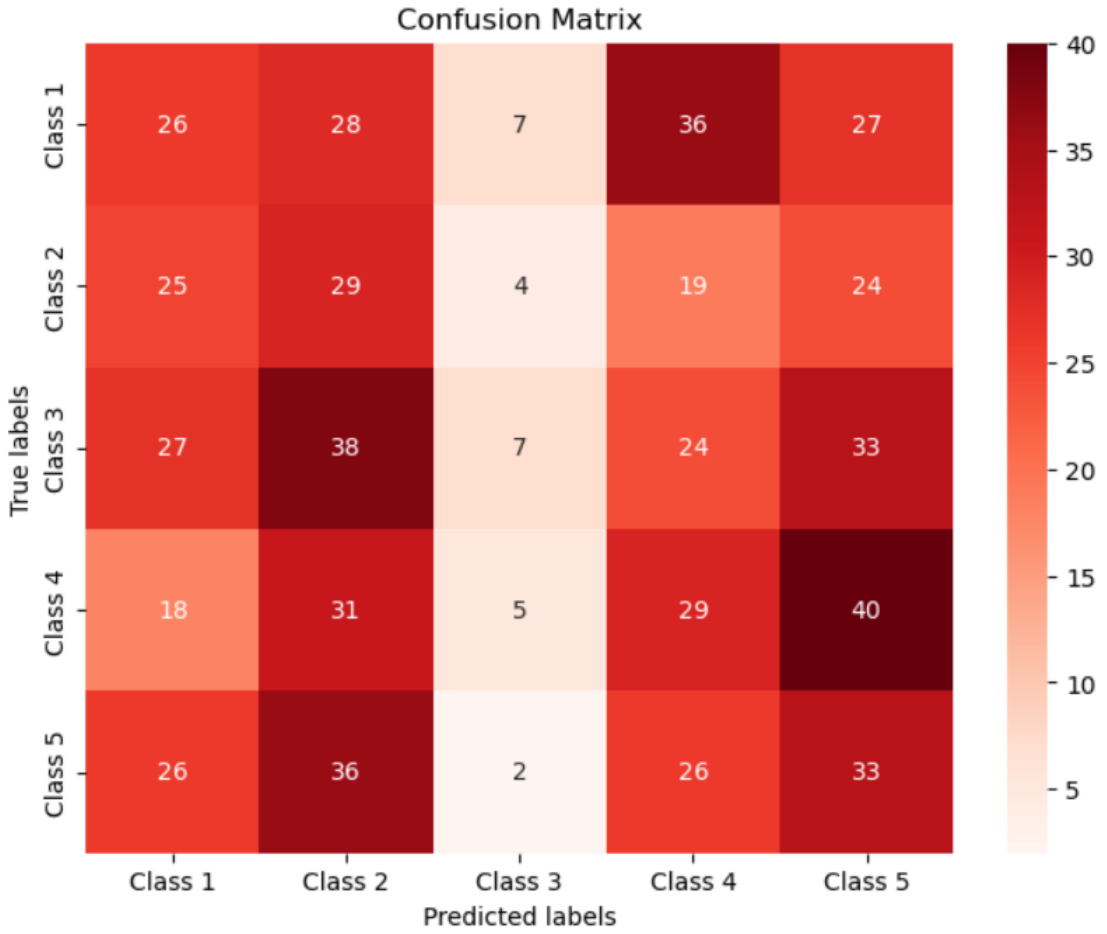
Provides insights into how well the model performs, beyond just accuracy.

[270]:
```
# evaluating the model
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(Y_test, Y_pred))
```

```
Accuracy: 0.20666666666666667
Classification Report:
              precision    recall  f1-score   support

           1       0.21      0.21      0.21       124
           2       0.18      0.29      0.22       101
           3       0.28      0.05      0.09       129
           4       0.22      0.24      0.23       123
           5       0.21      0.27      0.24       123

    accuracy                           0.21       600
   macro avg       0.22      0.21      0.20       600
weighted avg       0.22      0.21      0.19       600
```

Visualizing the Confusion Matrix

Provides detailed insights into the classification performance across all classes (1 to 5).



You probably have noticed the accuracy of the model was considerably low, why is that?

Potential Causes:

1. **Class Imbalance**:
    - If the stress levels (1 to 5) are unevenly distributed, the model may struggle with minority classes.
2. **Linear Assumption of Logistic Regression**:
    - Logistic regression assumes a linear relationship between features and the target. If the relationships are non-linear, this limits the model's predictive power.
3. **Weak Predictive Features**:
    - Some features may lack sufficient correlation with Stress_Level, impacting the model's ability to capture meaningful patterns.