Predicting the Impact of Environmental and Health Factors on Mental Health using Supervised Machine Learning Models

Group Name: Yellow Group
Team Members: Aarushi Kotwani
Vatsal Dalal
Zahra Ahmadi Angali
Lakshit Gupta

ABSTRACT

This study aims to analyze the impact of various environmental and health factors on mental health across over 180 countries using supervised machine learning methods. Utilizing two datasets com prising attributes like physical inactivity, air pollution, dietary habits, and specific disease burdens. Bidirectional Long Short-Term Memory (Bi-LSTM) networks were employed to predict the prevalenc e of mental disorders. Along with exploration of other supervised learning algorithms such as Linea r Regression and Support Vector Regression (SVR). Principal Component Analysis (PCA) is applied f or dimensionality reduction before fitting the models. Performance metrics, including Mean Square d Error (MSE) and R-squared, indicate the effectiveness of the model in capturing the predictions. T he Bi-LSTM model achieved an R-squared value of 0.9043, indicating the highest predictive accurac y among the models. The SVR model, with optimal parameters, achieved a Test Mean Squared Erro r (MSE) of 0.0038 and an R-squared value of 0.8727. The Linear Regression model also performed w ell, with a Test MSE of 0.0039 and an R-squared value of 0.8709.

INTRODUCTION

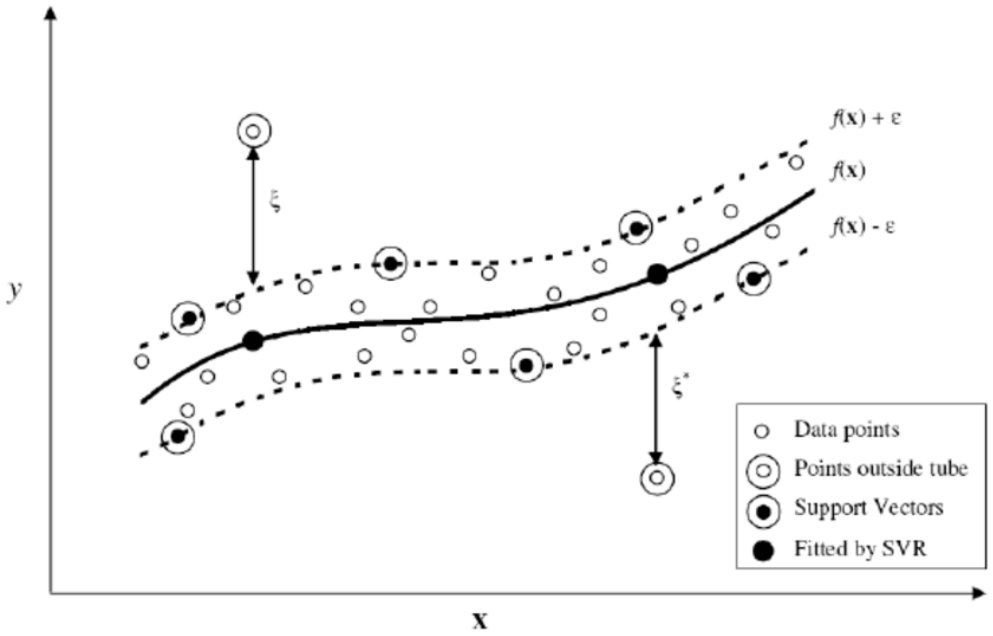
Mental health disorders constitute a significant global health challenge, impacting millions of individuals and posing substantial socio-economic burdens and is influenced by various factors ranging from environmental conditions to individual health behaviors. This project looks at two big sets of data obtained through the ourworldindata.org website [5], one about different health risk factors and their effects, and another about how much different diseases contribute to the overall health problems. Predicting mental disorders using health and environmental factors is crucial for proactive healthcare measures. This study focuses on applying advanced machine learning techniques, specifically Bidirectional Long Short-Term Memory (Bi-LSTM) networks, Support Vector Regression (SVR) and Linear Regression, to model and predict mental disorder trends based on a range of features including physical activity, dietary habits, pollution levels, and various health conditions across different countries. By using data from 1990 to 2019, the aim is to create robust models that can identify patterns and provide accurate predictions.

THEORITICAL BACKGROUND

Principal Component Analysis[6] :

PCA is an unsupervised learning algorithm that derives dense information from data while reducing its dimensions without losing much information. It simplifies the data by transforming the features into principal components that can explain the most variance in the data. These components capture the most variance, with the first (PC1) capturing the most and the second (PC2) capturing the next most variance, orthogonal to PC1. By transforming data into a lower-dimensional space, PCA highlights key patterns, making complex data easier to analyze and interpret.

Support Vector Regressor (SVR)[1] [2]



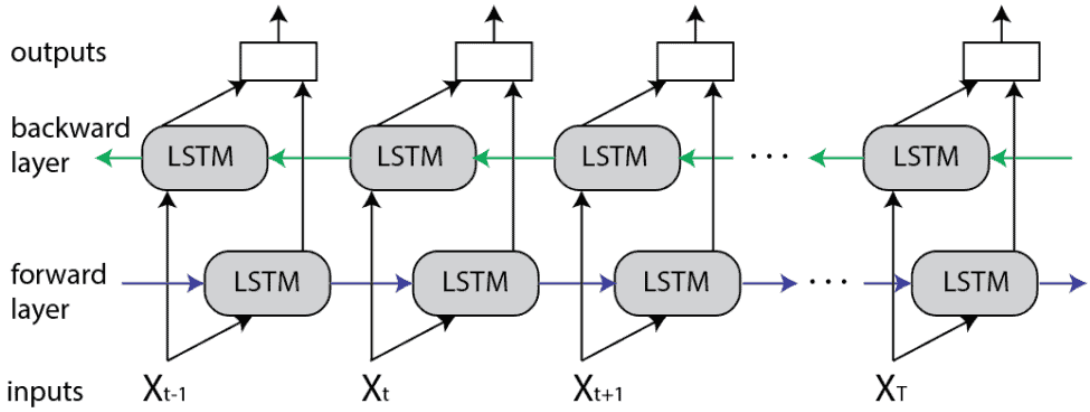
Support Vector Fitted by SVR

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. SVM works by finding a hyperplane in a high-dimensional space that best separates data into different classes. It aims to maximize the margin (the distance between the hyperplane and the nearest data points of each class) while minimizing classification errors. SVM can handle both linear and non-linear classification problems by using various kernel functions.
The Radial Kernel is used when the data exhibits non-linear relationships among its features. It computes the similarity between data points in a higher-dimensional space and is particularly useful when the classes are not easily separable by a straight line or plane. The parameter tuned in a Radial SVM is the regularization parameter C and gamma. The C parameter controls the trade-off between maximizing the margin and minimizing the classification error. Gamma decides how much each data point should influence the shape of the separation. A small gamma means each point has a big influence, so the separation will be smoother. A big gamma means each point has less influence, so the separation can be more irregular to fit the data better.

Linear Regression[3]

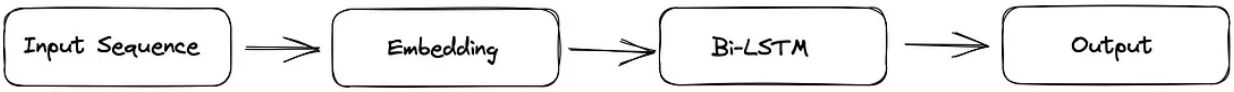
Linear regression is a statistical method used to predict the value of a dependent variable based on the value of one or more independent variables. In this analysis, the dependent variable is the outcome we aim to predict, while the independent variables are the predictors used to make this prediction.
The method involves estimating the coefficients of a linear equation that best fit the data. This equation can involve one or more independent variables. Linear regression aims to fit a straight line (or surface, in the case of multiple variables) that minimizes the differences between the predicted and actual values. Using a "least squares" approach, linear regression finds the best-fit line for a set of paired data, allowing us to estimate the value of the dependent variable from the independent variables.

Bidirectional Long Short-Term Memory (Bi-LSTM)[4]



Basic Diagram of Bi-directional LSTM

Bi-LSTM (Bidirectional Long Short-Term Memory) is a type of recurrent neural network (RNN) that processes sequential data in both forward and backward directions. It combines the power of LSTM with bidirectional processing, allowing the model to capture both past and future context of the input sequence.



- **Input Sequence:** A sequence of data points, such as words or characters, each represented as a vector or embedded representation.
- **Embedding:** Input sequences are transformed into dense vector representations (embeddings) that capture semantic meanings for more meaningful processing.
- **Bi-LSTM:** The core component, consisting of two LSTM layers—one processing the sequence forward and the other backward. Each layer has its own set of parameters.
- **Output:** The Bi-LSTM output combines hidden states from both forward and backward layers at each time step, often by concatenation or other transformations.

METHODOLOGY

Data Preparation

For preprocessing, the necessary datasets were loaded using Pandas library. The two datasets used were "Disease burden by risk factor" and "Share of total disease burden by cause". These datasets provided valuable information on various health risk factors and the associated disease burdens.

To simplify the data, the columns are renamed to shorter and more descriptive names, making the data easier to read and work with during analysis. Moving on, the data is checked for any missing values or inconsistencies as missing values can significantly impact the results of an analysis, the missing values are then removed from the data sets. Also removing the duplicate entries of multiple rows in the dataset that could lead to data redundancy, limiting our unique countries to a count of around 185. To ensure consistency across the dataset, the data was then normalized to per 100,000 of the population. Additionally, the data types of all columns were verified to ensure uniformity and correctness. Followed by using the correlation method on the “share of total disease burden” the 6 most highly correlated variables are selected, shaping the new form of the dataset.

After ensuring the data integrity, the two datasets were merged using common identifiers "Entity" (the name of the country) and "Year" (the year of the data). Through the merging process a comprehensive dataset is created that integrates both health risk factors and disease burdens providing a more complete and detailed view, enabling to draw thorough and insightful analysis.

Selecting relevant features such as low physical activity, non-exclusive breastfeeding, air pollution, child wasting, high systolic blood pressure, high fasting plasma glucose, high body mass index, secondhand smoke, unsafe sanitation, unsafe water source, diet low in vegetables, diet low in fruits, diet high in sodium, drug use, household air pollution, high LDL cholesterol, iron deficiency, zinc deficiency, smoking, vitamin A deficiency, ambient particulate matter pollution, substance use disorders, skin and subcutaneous diseases, musculoskeletal disorders, neoplasms, neurological disorders, sense organ diseases, and child stunting, with the target variable being mental disorders.

To standardize the data, MinMaxScaler is used to normalize the features and the target variable. This ensures that all data points are within a comparable range, enhancing the performance of the machine learning model.

Principal Component Analysis

Before proceeding with the analysis, the combined dataset was standardized to ensure each feature had an equal impact. With the application of PCA, the dataset's dimensions were reduced, resulting in fewer principal components that contain the essential information. The optimal number of components was determined by analyzing the explained variance ratio and selecting the components that collectively accounted for a substantial proportion of the total variance using the “Elbow Method”. This selection process made it easier to see relationships and patterns among

health risk factors and disease burdens. The aim of this process is to simplify the data while preserving its core insights, leading to clearer and more efficient further analysis. After performing PCA, the data (from 1990 to 2019) is split into training and testing sets using a 70:30 ratio. This ensures that the model is trained on a robust dataset and is tested effectively.

Support Vector Regressor (SVR)

An SVR (Support Vector Regression) model with a Radial Basis Function (RBF) kernel is initialized, leveraging RBF's ability to handle non-linear relationships between features and the target variable. GridSearchCV is used for hyperparameter tuning, with parameters {'C': [0.01, 0.1, 1, 10, 50, 100], 'gamma': [0.01, 0.1, 1, 10, 50, 100]}. The best parameters are selected based on the Negative Mean Squared Error scoring metric; this metric ensures that lower MSE values result in higher scores aligning with the maximization objective of GridSearchCV. The model is then trained on the training data and used to make predictions on the testing data. Its performance is evaluated using Mean Squared Error (MSE) and R-squared (R^2) metrics. The actual versus predicted values are plotted, between the model's predictions and the actual values. Additionally, a histogram of residuals (differences between actual and predicted values) is plotted to analyze the distribution of prediction errors.

Linear Regression

A Linear Regression model is initialized and trained on the training data. Predictions are made on the test data, and the model's performance is evaluated using Mean Squared Error (MSE) and R-squared (R^2) metrics. A plot of actual versus predicted values for the test set provides a visual evaluation, and a histogram of residuals (differences between actual and predicted values) helps analyze the distribution of prediction errors.

Bidirectional Long Short-Term Memory (Bi-LSTM)

A Sequential model is built with three Bidirectional LSTM layers, each followed by a Dropout layer to prevent overfitting, to capture the temporal dependencies in the sequence data. A Dense layer with a single unit is used for the regression task to predict the target variable. The model is compiled using the 'rmsprop' optimizer, 'mse' (mean squared error) as the loss function, and 'mae' (mean absolute error) as a metric. An early stopping callback are used to stop training if the validation loss does not improve for 10 consecutive epochs. And, a reduced learning rate[7] callback to reduce the learning rate after the model stops improving with the hope of fine-tuning model weights for 5 consecutive epochs. The model is trained for up to 100 epochs with a batch size of 32, using the training data. Testing data is used to monitor the performance during training. The training and validation loss, as well as the mean absolute error, are plotted over epochs to visualize the model's performance. The model makes predictions on the test data. A line plot is used to compare the actual and predicted values of the target variable. The residuals (difference between actual and predicted values) are computed and plotted as a histogram to analyze the model's prediction errors.

COMPUTATIONAL RESULTS

Principal Component Analysis (PCA)

With application of PCA a total of 4 of the principal components which explained the majority of the variance were selected. This method helped identify the most important patterns in the data.

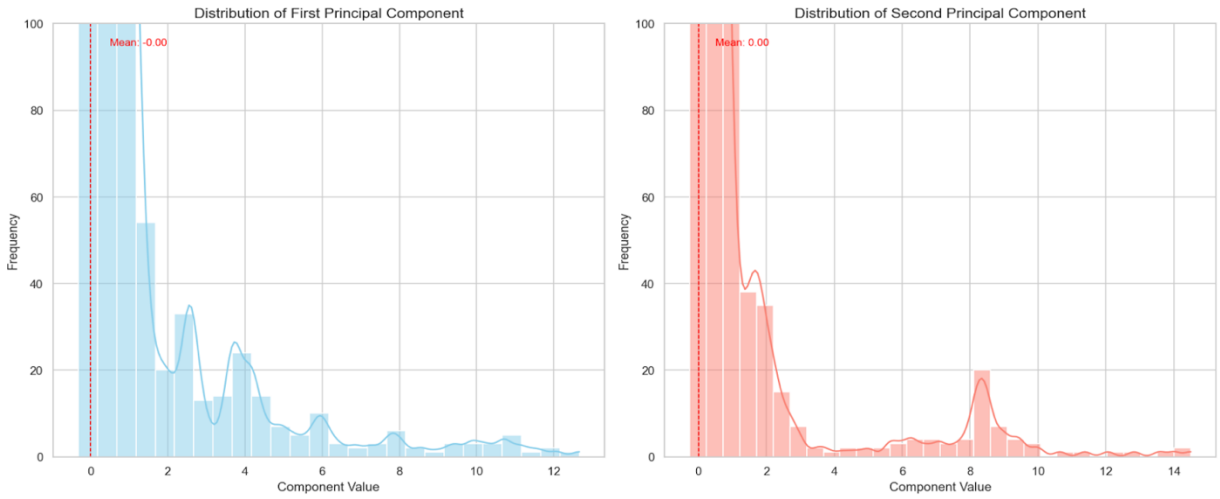


Figure 1. Frequency vs Component Value plot for the first and second principal component

The plot above shows the normalized singular values, indicating the variance captured by each principal component.

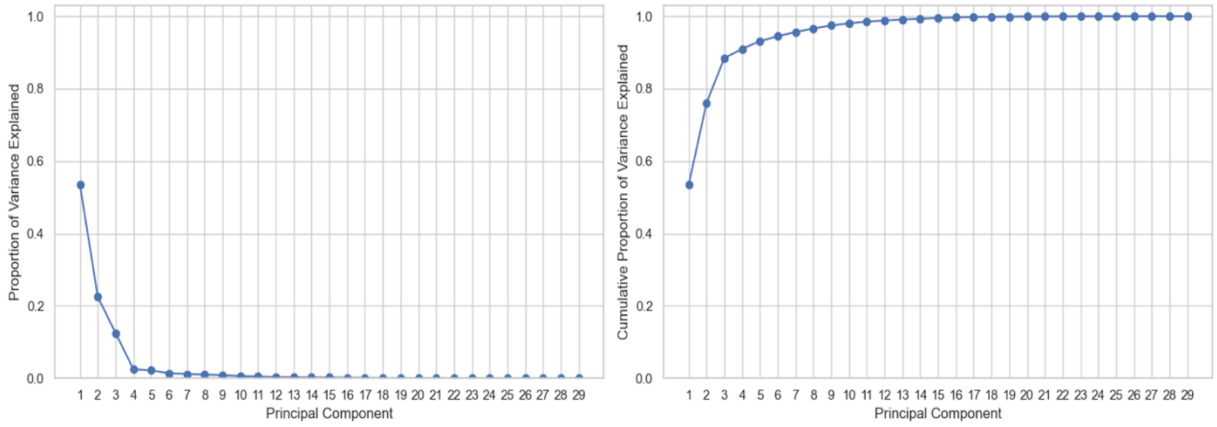


Figure 2. Plot for Proportion of Variance Explained and Cumulative Proportion of Variance Explained

The above plot shows the proportion of variance explained by each principal component, helping to determine the components that capture the most significant information.

```
PCA Loadings:
                                          PC1       PC2       PC3       PC4
low_physical_activity                0.212415  0.145885 -0.118299  0.148716
non_exclusive_breastfeeding          0.198540 -0.157623  0.172540 -0.070970
air_pollution                        0.249433 -0.004436  0.010712 -0.141177
child_wasting                        0.203092 -0.165662  0.209707 -0.032775
high_systolic_blood_pressure         0.224138  0.133343 -0.155266 -0.008425
high_fasting_plasma_glucose          0.227395  0.129165 -0.125603  0.042586
child_stunting                       0.193075 -0.173807  0.230871 -0.024464
high_body_mass_index                 0.197867  0.165229 -0.168983  0.247438
secondhand_smoke                     0.231702  0.081320 -0.097590 -0.213587
unsafe_sanitation                    0.200048 -0.162379  0.230774  0.045155
unsafe_water_source                  0.202230 -0.159259  0.225930  0.053575
diet_low_in_vegetables               0.223073 -0.003001  0.061173  0.199295
diet_low_in_fruits                   0.237983  0.091141 -0.098777 -0.049547
diet_high_in_sodium                  0.179092  0.152992 -0.227137 -0.321276
drug_use                             0.200096  0.138842 -0.136886  0.133721
household_air_pollution              0.236732 -0.074069  0.099677 -0.143585
high_ldl_cholesterol                 0.221231  0.128952 -0.130133  0.169879
iron_deficiency                      0.222299 -0.059561  0.128716  0.096306
zinc_deficiency                      0.179360 -0.170202  0.241040  0.061467
smoking                              0.212048  0.153815 -0.172452 -0.102328
vitamin_a_deficiency                 0.178420 -0.188677  0.228016  0.018896
ambient_particulate_matter_pollution 0.230421  0.097758 -0.120554 -0.113656
substance_use_disorders             -0.009187  0.253340  0.152362  0.655318
skin_and_subcutaneous_diseases      -0.026409  0.286467  0.254842 -0.308609
musculoskeletal_disorders           -0.020727  0.316480  0.269477 -0.066615
neoplasms                           -0.022955  0.290043  0.219899  0.113174
neurological_disorders              -0.034357  0.315475  0.279734 -0.045605
sense_organ_diseases                 0.003371  0.302272  0.197044 -0.087097
mental_disorders                    -0.033414  0.283241  0.262180 -0.190954
```

Figure 3. Contribution of each variable for each PC

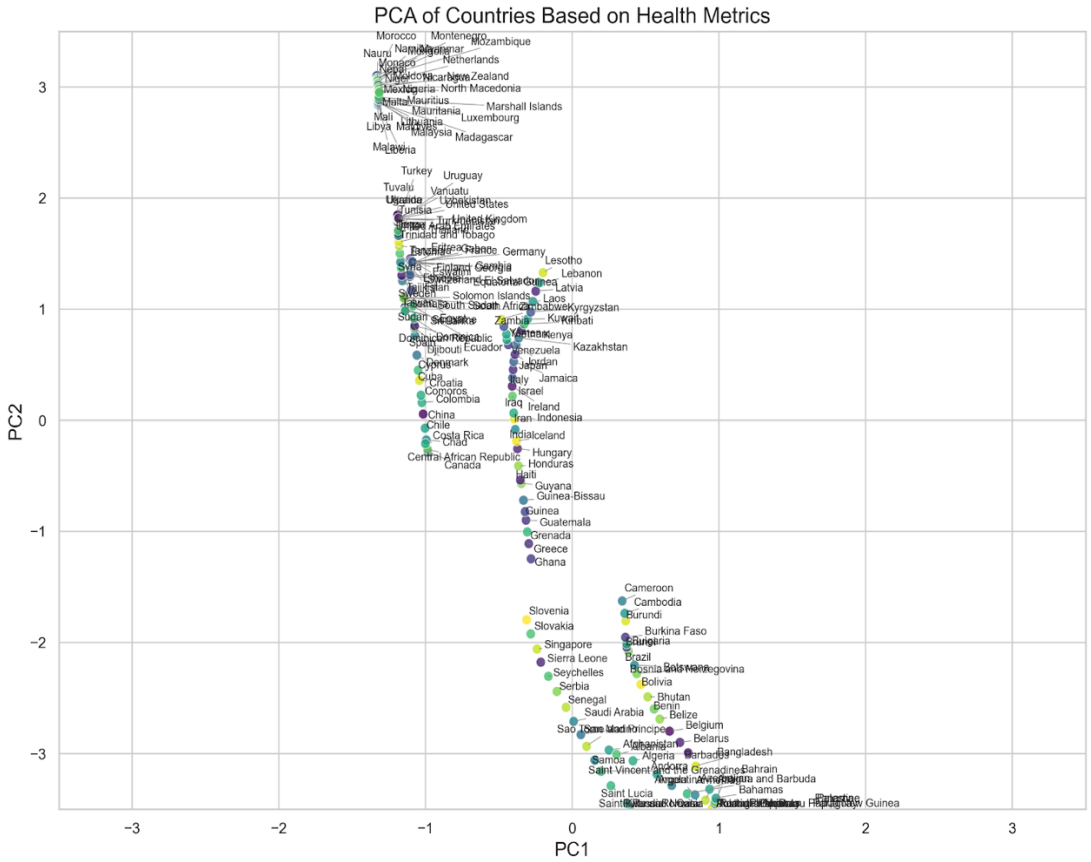


Figure 4. Plot for PCA of Countries Based on Health Metrics

This scatter plot visualizes the distribution of countries based on their health metrics using the first two principal components. The country names are included and adjusted to minimize overlap.

Support Vector Regressor (SVR)

The implementation of GridSearchCV with an RBF kernel yielded optimal parameters of {'C': 10, 'gamma': 1}. This resulted in a test accuracy of 87.27%, with a Test Mean Squared Error (MSE) of 0.0038 and a Test R-squared value of 0.8727. This high R-squared value underscores the model's effectiveness in capturing and explaining the variability in the target variable

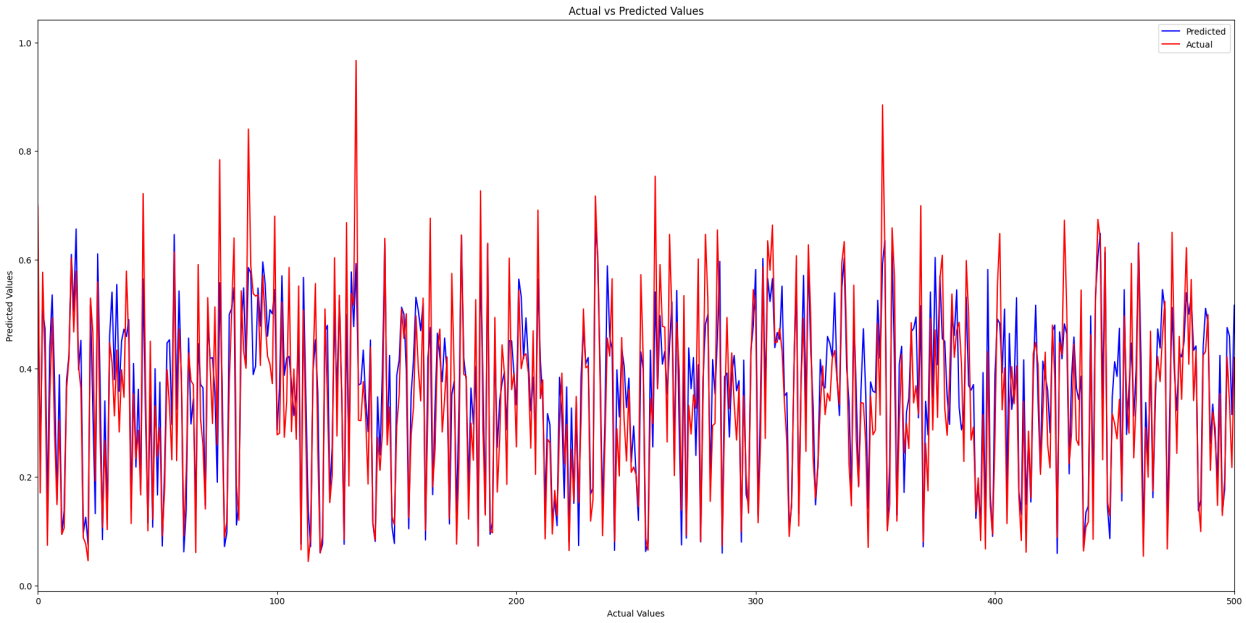


Figure 5. Support Vector Regressor (SVR): Actual vs Predicted Values

The above graph compares the actual mental disorder values (red line) to the predicted values (blue line). Any significant deviation between the two lines suggests areas where the model's predictions are less accurate.

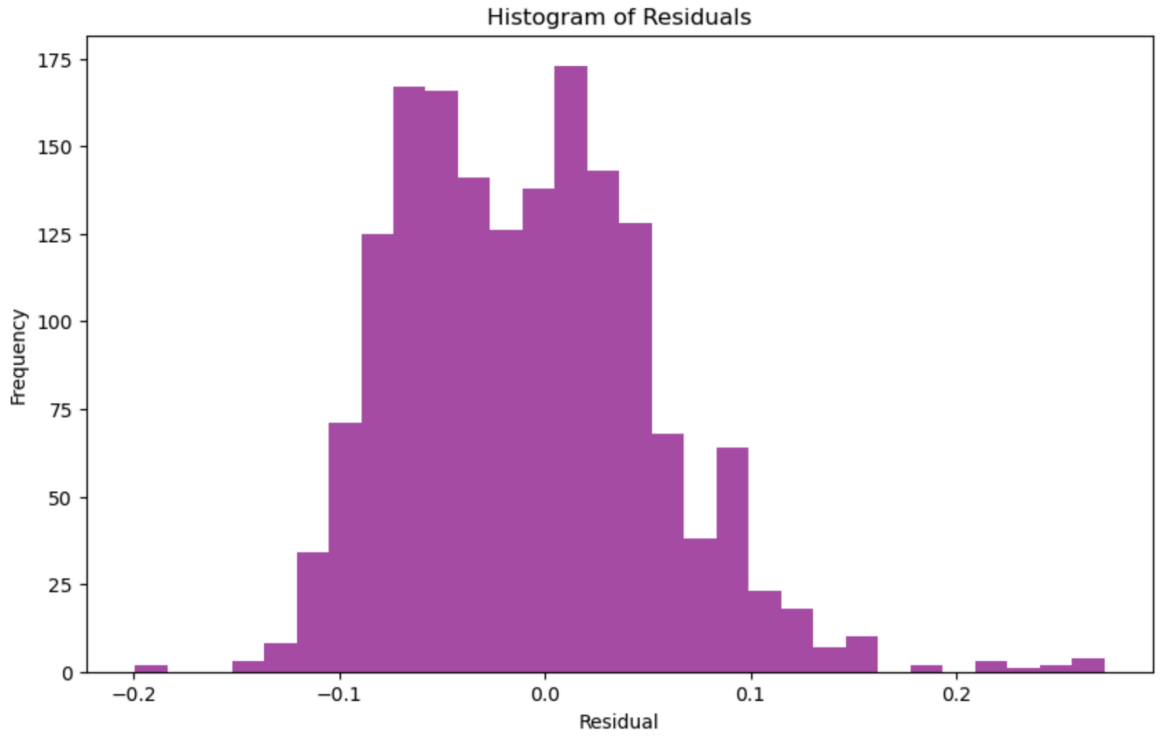


Figure 6. Support Vector Regressor (SVR): Histogram of Residuals

The histogram shows that the residuals, or differences between what the model predicted and the actual values, form a bell-shaped curve centered around zero. This means the model's predictions are mostly accurate, with no clear patterns of error. The spread of the residuals is moderate, indicating that the model's predictions are generally close to the real values, which is a good sign for the model's performance.

Linear Regression

The evaluation metrics indicate that the model exhibits strong predictive performance for mental disorders. The Test Mean Squared Error (MSE) of 0.0039, the squared difference between the actual and predicted values is very small, reflecting high accuracy in the model's predictions. Additionally, the Test R-squared (R^2) value of 0.8709 signifies that approximately 87.09% of the variance in mental disorder incidence is explained by the model using the four principal components derived from the original features. This high R^2 value underscores the model's effectiveness in capturing and explaining the variability in the target variable.

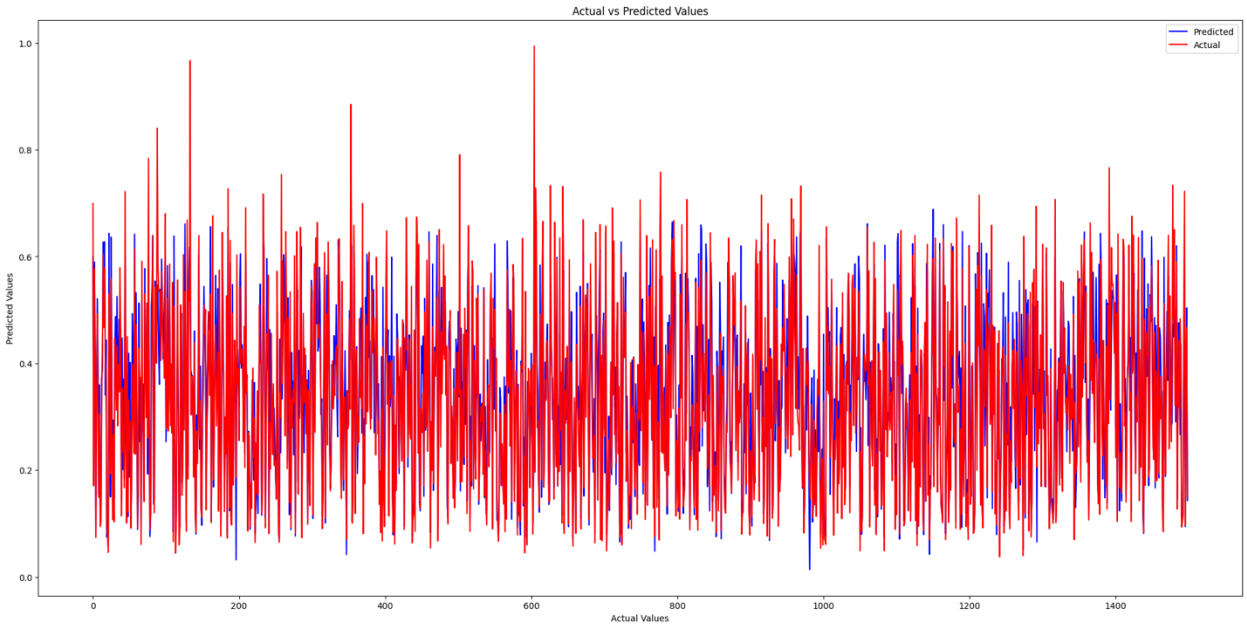


Figure 7. Linear Regression: Actual vs Predicted Values

The blue lines represent the predicted values, while the red lines represent the actual values. Both sets of lines follow a similar pattern, indicating that the model's prediction is close to the actual values.

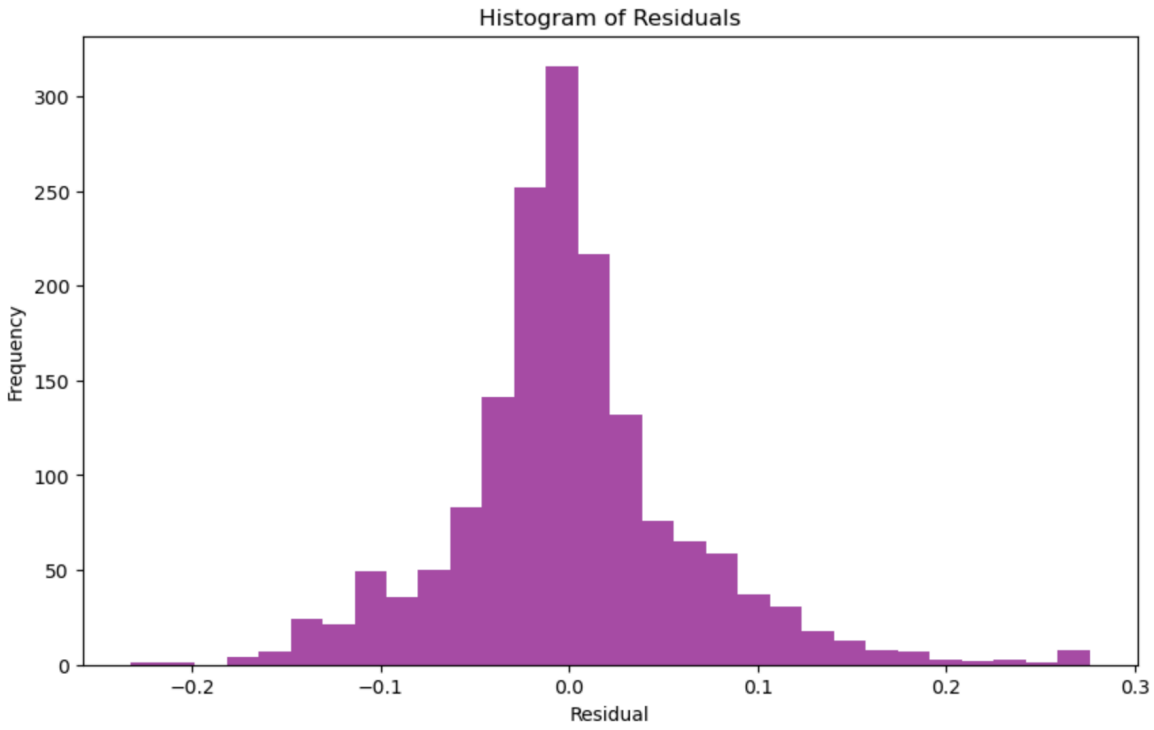


Figure 8. Linear Regression: Histogram of Residuals

The histogram is symmetric around zero, with a slight skew to the right. This means that the residuals are mostly centered around zero, but there are a few more positive outliers than negative ones. The peak of the histogram is around zero, representing that the model's predictions are generally accurate.

Bidirectional Long Short-Term Memory (Bi-BI-LSTM)

The Bi-LSTM model demonstrates strong predictive performance with an R-squared value of 0.9043, indicating it explains approximately 90.43% of the variance in the target variable.

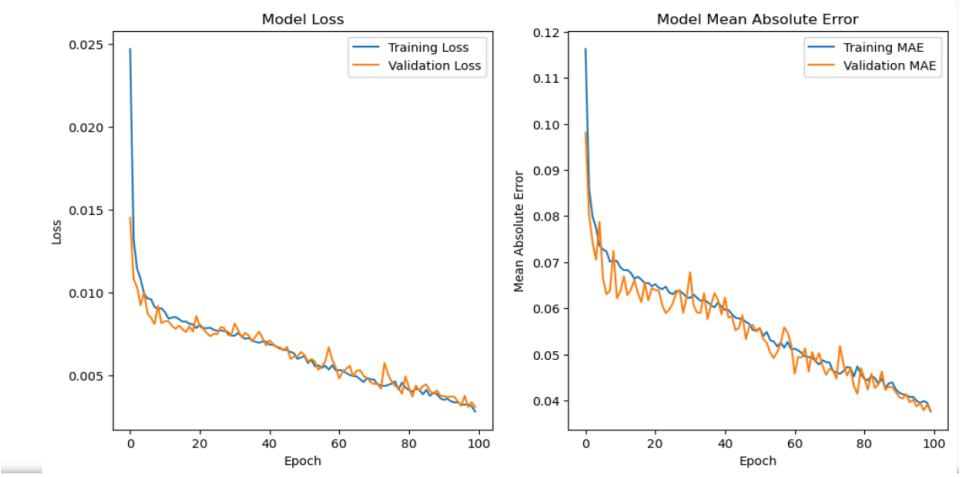


Figure 9. Long Short-Term Memory (BI-LSTM): Convergence Plots

The primary metrics observed during training were the loss and Mean Absolute Error (MAE) for both training and validation datasets. Throughout the 100 epochs, both the training and validation loss decreased significantly, indicating that the model was learning effectively. The MAE also consistently declined, suggesting improved predictive accuracy over time.

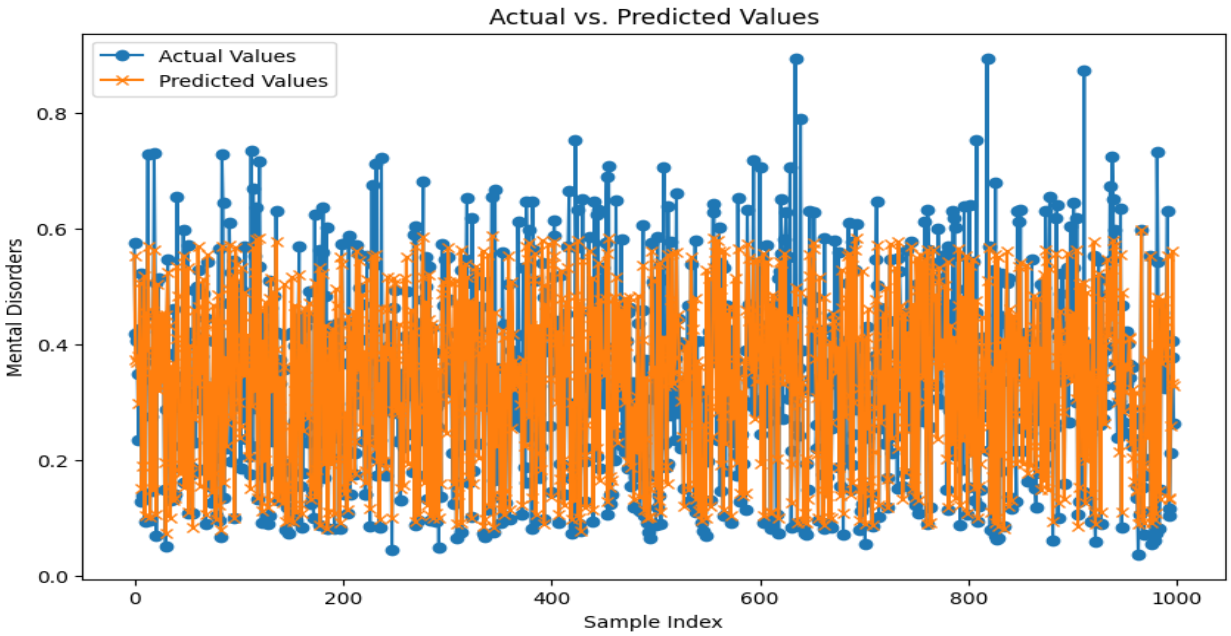


Figure 10. Long Short-Term Memory (BI-LSTM): Actual vs Predicted Values

The second plot compares the actual values of mental disorders against the predicted values from the model across a set of samples. The blue dots represent the actual values, and the orange crosses represent the predicted values. The predicted values closely follow the actual values, although some variability is present.

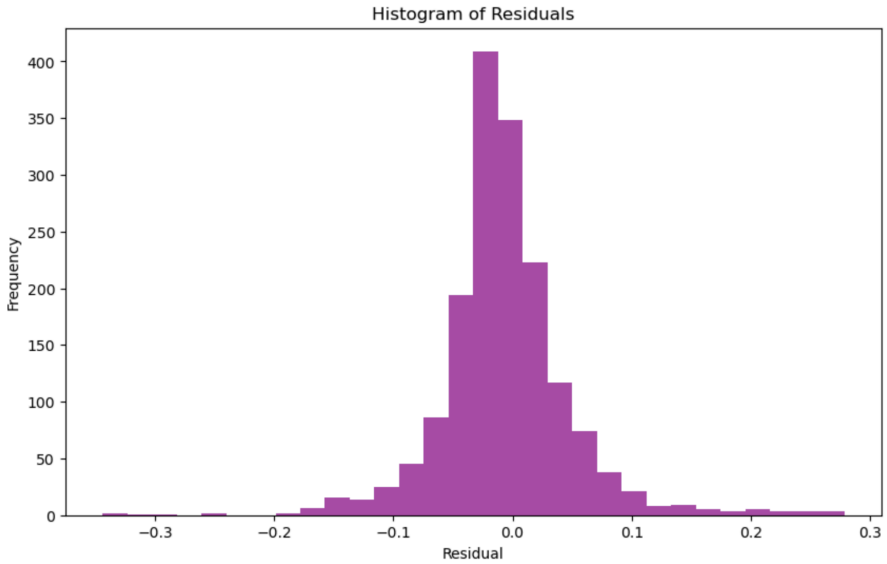


Figure 11. Bidirectional Long Short-Term Memory (Bi-LSTM): Histogram of Residuals

Residuals are the differences between the actual values and the predicted values. The histogram appears to be roughly symmetric and centered around 0, indicating that the residuals are normally distributed. The highest frequency of residuals is close to 0, suggesting that most of the predictions are accurate.

DISCUSSION

This project uses supervised learning methods to analyze two comprehensive datasets about health risk factors and their impacts. The goal is to understand how various factors like diseases, habitat, environmental factors contribute to the overall mental health problem. Through the integration of Principal Component Analysis (PCA), Support Vector Regressor (SVM), Linear Regression and Bidirectional Long Short-Term Memory (Bi-LSTM).

Principal Component Analysis (PCA)

With implementation of PCA on the finalized scaled dataset from Figure 2., the optimal component using the elbow method is selected to be 4, the analysis proceeds with 4 numbers of components and the results are illustrated in Figure 2. Looking at Figure 3. principal component analysis reveals that each of these components explains different sets of variables. PC1 is explained principally by factors of environmental pollution and dietary risks, as significant contributors are air pollution, low fruit intake, household air pollution, secondhand smoke, and other related variables. Chronic diseases and disorders that are significant contributors to PC2 include neurological disorders, musculoskeletal disorders, and skin diseases. The PC3 is related to chronic diseases, mental disorders, and deficiencies in association with essential factors such as neurological disorders, cognitive disorders, vitamin A deficiency, and unsafe water and sanitation. The fourth principal component is primarily related to Substance use, skin conditions, dietary sodium, and body mass index, and substance use disorders, high sodium intake, and high body mass index are the critical contributory elements. In our analysis, we regarded as essential contributors to the principal component all variables with threshold of 0.2 and higher in absolute value.

The PCA plot of countries based on health metrics reveals the most significant patterns in the data by portraying countries onto the first two principal components (PC1 and PC2). Each point represents a country, and its position is determined by its health metrics. Countries near each other forming a cluster on the plot have similar health metrics. For example, Norway, Finland, and Sweden are all clustered together, indicating that they have similar health metrics. Similarly, Angola, Mozambique, and Liberia form another cluster with similar health metrics.

Additionally, from analyzing the figure furthermore it can be stated that the axes PC1 and PC2 capture the most variance in the data, demonstrating how much variation each component accounts for. Moreover, Countries far from the central cluster, at the extremes of the axes, are considered outliers, indicating that their health metrics differ significantly from those of the majority. Labels are adjusted to be more readable, making it easier to identify individual countries within clusters.

Furthermore, directional trends along the axes reveal underlying patterns, such as higher or lower values for specific health metrics in various countries. For instance, Mexico, Brazil, and Argentina, which are near one end of the PC1 axis, might have similar health challenges or strengths to those near the other end. This plot effectively summarizes the similarities and differences in health metrics across countries, providing useful insights for more in-depth analysis.

Support Vector Regressor (SVR)

Figure 5 and Figure 6 demonstrating the actual vs predicted values and Histogram of Residuals, respectively. From Figure 5, the model captures the overall trend of the actual values well. The alignment between the blue and red lines indicates the accuracy of the model's predictions. Closer alignment means better accuracy, while larger deviations indicate areas where the model's predictions are less accurate. Furthermore, from Figure 6 most of the residuals are centered around zero, although they are skewed slightly to the opposing side. This indicates that there is a tendency to overestimate the model. Although many residuals remain small, there are several more significant residuals present, which could indicate possible outliers or points that need some clarification in the model.

Linear Regression

Figure 7 indicates that the predicted values follow the trend of the actual values, but there are noticeable deviations at certain points. This suggests that while the model is effective in capturing the overall trend of the data, there are specific instances where the predictions significantly differ from the actual values. Figure 8 shows the clustering of residuals around zero with a high peak indicates the model performs well for most predictions. The slight right skew suggests that there are some instances where the model unpredicts the actual values.

Bidirectional Long Short-Term Memory (Bi-LSTM)

Figure 9 demonstrates the consistent reduction in both training and validation loss indicates that the model effectively captures the underlying patterns in the data without overfitting. The MAE metric aligns with the loss, providing further confirmation that the model's predictions are accurate and consistent over time. Additionally, the scatter plot of actual vs. predicted values Figure 10, indicates that the model can reliably predict the mental disorder values for the given samples. The close alignment between the actual and predicted values suggests high accuracy.
Figure 11 Histogram of Residuals indicates that the neural network model predicts the target variable with a high degree of accuracy. Most residuals are close to 0, indicating that the predictions are generally very close to the actual values.

CONCLUSION

Mental health disorders constitute a significant global health challenge, impacting millions of individuals and posing substantial socio-economic burdens. Predicting mental disorders using health and environmental factors is crucial for proactive healthcare measures. This study demonstrates that advanced machine learning techniques, specifically Bi-LSTM networks, SVR, and Linear Regression, can effectively model and predict mental disorder trends based on various features. By using data from 1990 to 2019, we created robust models that can identify patterns and provide accurate predictions, contributing valuable insights for addressing mental health issues globally. All three models exhibited strong predictive performance, with high R-squared values and low MSE, indicating their effectiveness in predicting mental disorder values. The Bi-LSTM model slightly outperformed the other models with the highest R-squared value of 0.9043, while the SVR and Linear Regression models also demonstrated robust performance with minimal errors and close alignment between predicted and actual values with R-squared values of 0.8727 and 0.8709 respectively. These findings suggest that each model is a suitable choice for this prediction task, with the Bi-LSTM model being slightly more effective in capturing and explaining the variability in the target variable.

REFERENCES

1. What is Support Vector Machine | IBM
2. Support Vector Machine (SVM) Algorithm - GeeksforGeeks
3. (What Is Linear Regression? | IBM)
4. Understanding Bidirectional LSTM for Sequential Data Processing | by Anishnama | Medium
5. OurWorld Datasets
6. Principal Component Analysis
7. Reduced Learning Rate

Code Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv('./disease-burden-by-risk-factor.csv')

# Define new column names
new_columns = {
    'DALYs that are from all causes attributed to low physical activity, in both sexes aged all ages': 'low_physical_activity',
    'DALYs that are from all causes attributed to non-exclusive breastfeeding, in both sexes aged all ages': 'non_exclusive_breastfeeding',
    'DALYs that are from all causes attributed to air pollution, in both sexes aged all ages': 'air_pollution',
    'DALYs that are from all causes attributed to child wasting, in both sexes aged all ages': 'child_wasting',
```

```
    'DALYs that are from all causes attributed to high systolic blood pressure, in both sexes aged all ages': 'high_systolic_blood_pressure',
    'DALYs that are from all causes attributed to high fasting plasma glucose, in both sexes aged all ages': 'high_fasting_plasma_glucose',
    'DALYs that are from all causes attributed to child stunting, in both sexes aged all ages': 'child_stunting',
    'DALYs that are from all causes attributed to high body-mass index, in both sexes aged all ages': 'high_body_mass_index',
    'DALYs that are from all causes attributed to secondhand smoke, in both sexes aged all ages': 'secondhand_smoke',
    'DALYs that are from all causes attributed to unsafe sanitation, in both sexes aged all ages': 'unsafe_sanitation',
    'DALYs that are from all causes attributed to unsafe water source, in both sexes aged all ages': 'unsafe_water_source',
    'DALYs that are from all causes attributed to diet low in vegetables, in both sexes aged all ages': 'diet_low_in_vegetables',
    'DALYs that are from all causes attributed to diet low in fruits, in both sexes aged all ages': 'diet_low_in_fruits',
    'DALYs that are from all causes attributed to diet high in sodium, in both sexes aged all ages': 'diet_high_in_sodium',
    'DALYs that are from all causes attributed to drug use, in both sexes aged all ages': 'drug_use',
    'DALYs that are from all causes attributed to household air pollution from solid fuels, in both sexes aged all ages': 'household_air_pollution',
    'DALYs that are from all causes attributed to high ldl cholesterol, in both sexes aged all ages': 'high_ldl_cholesterol',
    'DALYs that are from all causes attributed to iron deficiency, in both sexes aged all ages': 'iron_deficiency',
    'DALYs that are from all causes attributed to zinc deficiency, in both sexes aged all ages': 'zinc_deficiency',
    'DALYs that are from all causes attributed to smoking, in both sexes aged all ages': 'smoking',
    'DALYs that are from all causes attributed to vitamin a deficiency, in both sexes aged all ages': 'vitamin_a_deficiency',
    'DALYs that are from all causes attributed to ambient particulate matter pollution, in both sexes aged all ages': 'ambient_particulate_matter_pollution'
}

# Rename the columns
df.rename(columns=new_columns, inplace=True)

# Print the columns to verify
print(df.columns)

# Print the DataFrame
df
```

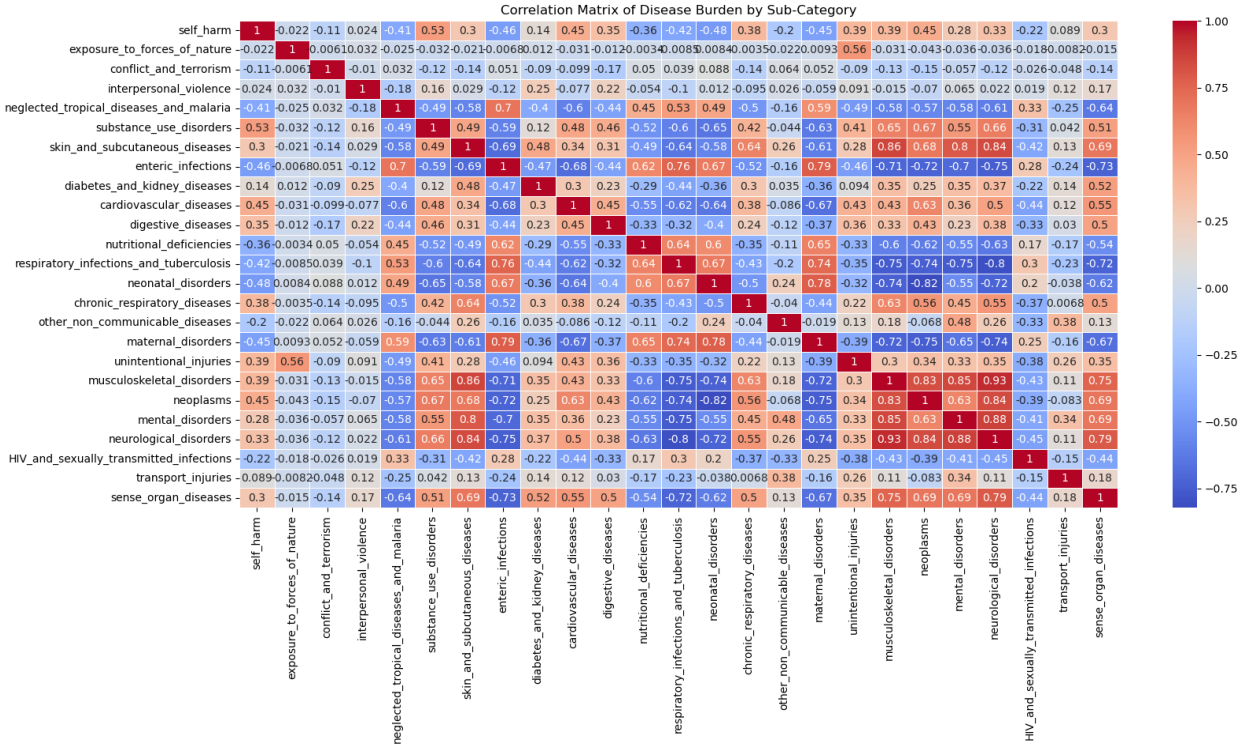
```
Index(['Entity', 'Code', 'Year', 'low_physical_activity',
       'non_exclusive_breastfeeding', 'air_pollution', 'child_wasting',
       'high_systolic_blood_pressure', 'high_fasting_plasma_glucose',
       'child_stunting', 'high_body_mass_index', 'secondhand_smoke',
       'unsafe_sanitation', 'unsafe_water_source', 'diet_low_in_vegetables',
       'diet_low_in_fruits', 'diet_high_in_sodium', 'drug_use',
       'household_air_pollution', 'high_ldl_cholesterol', 'iron_deficiency',
```

```
      'zinc_deficiency', 'smoking', 'vitamin_a_deficiency',
      'ambient_particulate_matter_pollution'],
     dtype='object')

df1 = pd.read_csv('./share-of-total-disease-burden-by-cause.csv')

# Define new column names
new_columns_df1 = {
   'DALYs (Disability-Adjusted Life Years) - Self-harm - Sex: Both - Age: All Ages (Percent)': 'self_harm',
   'DALYs (Disability-Adjusted Life Years) - Exposure to forces of nature - Sex: Both - Age: All Ages (Percent)': 'exposure_to_forces_of_nature',
   'DALYs (Disability-Adjusted Life Years) - Conflict and terrorism - Sex: Both - Age: All Ages (Percent)': 'conflict_and_terrorism',
   'DALYs (Disability-Adjusted Life Years) - Interpersonal violence - Sex: Both - Age: All Ages (Percent)': 'interpersonal_violence',
   'DALYs (Disability-Adjusted Life Years) - Neglected tropical diseases and malaria - Sex: Both - Age: All Ages (Percent)': 'neglected_tropical_diseases_and_malaria',
   'DALYs (Disability-Adjusted Life Years) - Substance use disorders - Sex: Both - Age: All Ages (Percent)': 'substance_use_disorders',
   'DALYs (Disability-Adjusted Life Years) - Skin and subcutaneous diseases - Sex: Both - Age: All Ages (Percent)': 'skin_and_subcutaneous_diseases',
   'DALYs (Disability-Adjusted Life Years) - Enteric infections - Sex: Both - Age: All Ages (Percent)': 'enteric_infections',
   'DALYs (Disability-Adjusted Life Years) - Diabetes and kidney diseases - Sex: Both - Age: All Ages (Percent)': 'diabetes_and_kidney_diseases',
   'DALYs (Disability-Adjusted Life Years) - Cardiovascular diseases - Sex: Both - Age: All Ages (Percent)': 'cardiovascular_diseases',
   'DALYs (Disability-Adjusted Life Years) - Digestive diseases - Sex: Both - Age: All Ages (Percent)': 'digestive_diseases',
   'DALYs (Disability-Adjusted Life Years) - Nutritional deficiencies - Sex: Both - Age: All Ages (Percent)': 'nutritional_deficiencies',
   'DALYs (Disability-Adjusted Life Years) - Respiratory infections and tuberculosis - Sex: Both - Age: All Ages (Percent)': 'respiratory_infections_and_tuberculosis',
   'DALYs (Disability-Adjusted Life Years) - Neonatal disorders - Sex: Both - Age: All Ages (Percent)': 'neonatal_disorders',
   'DALYs (Disability-Adjusted Life Years) - Chronic respiratory diseases - Sex: Both - Age: All Ages (Percent)': 'chronic_respiratory_diseases',
   'DALYs (Disability-Adjusted Life Years) - Other non-communicable diseases - Sex: Both - Age: All Ages (Percent)': 'other_non_communicable_diseases',
   'DALYs (Disability-Adjusted Life Years) - Maternal disorders - Sex: Both - Age: All Ages (Percent)': 'maternal_disorders',
   'DALYs (Disability-Adjusted Life Years) - Unintentional injuries - Sex: Both - Age: All Ages (Percent)': 'unintentional_injuries',
   'DALYs (Disability-Adjusted Life Years) - Musculoskeletal disorders - Sex: Both - Age: All Ages (Percent)': 'musculoskeletal_disorders',
   'DALYs (Disability-Adjusted Life Years) - Neoplasms - Sex: Both - Age: All Ages (Percent)': 'neoplasms',
   'DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)': 'mental_disorders',
   'DALYs (Disability-Adjusted Life Years) - Neurological disorders - Sex: Both - Age: All Ages (Percent)': 'neurological_disorders',
```

```
    'DALYs (Disability-Adjusted Life Years) - HIV/AIDS and sexually transmitted infections - Sex: Both - Age: All Ages (Percent)': 'HIV_and_sexually_transmitted_infections',
    'DALYs (Disability-Adjusted Life Years) - Transport injuries - Sex: Both - Age: All Ages (Percent)': 'transport_injuries',
    'DALYs (Disability-Adjusted Life Years) - Sense organ diseases - Sex: Both - Age: All Ages (Percent)': 'sense_organ_diseases'
}

# Rename the columns
df1.rename(columns=new_columns_df1, inplace=True)

# Print the columns to verify
print(df1.columns)

# Print the DataFrame
df1
```

```
Index(['Entity', 'Code', 'Year', 'self_harm', 'exposure_to_forces_of_nature',
       'conflict_and_terrorism', 'interpersonal_violence',
       'neglected_tropical_diseases_and_malaria', 'substance_use_disorders',
       'skin_and_subcutaneous_diseases', 'enteric_infections',
       'diabetes_and_kidney_diseases', 'cardiovascular_diseases',
       'digestive_diseases', 'nutritional_deficiencies',
       'respiratory_infections_and_tuberculosis', 'neonatal_disorders',
       'chronic_respiratory_diseases', 'other_non_communicable_diseases',
       'maternal_disorders', 'unintentional_injuries',
       'musculoskeletal_disorders', 'neoplasms', 'mental_disorders',
       'neurological_disorders', 'HIV_and_sexually_transmitted_infections',
       'transport_injuries', 'sense_organ_diseases'],
      dtype='object')
```

```
import seaborn as sns
corr_matrix = df1.drop(['Year','Entity','Code'],axis =1).corr()

plt.figure(figsize=(18, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix of Disease Burden by Sub-Category')
plt.show()
```

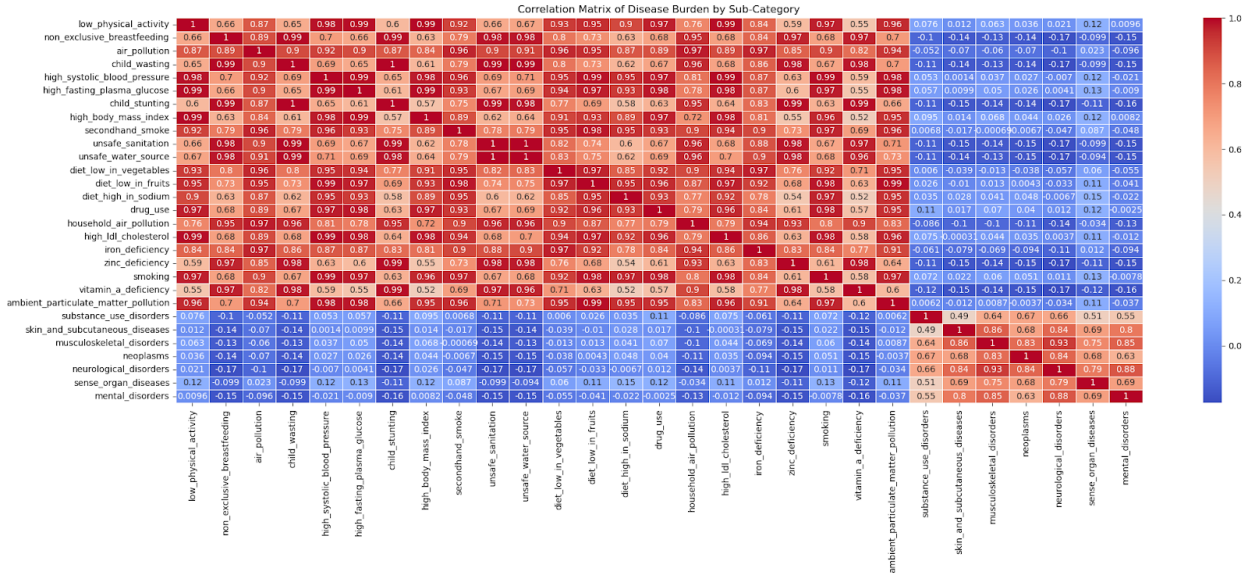



```
target_column = 'mental_disorders'
related_correlations = corr_matrix[target_column]

# Create a list of columns with correlation >= 0.5
high_correlation_columns = related_correlations[related_correlations >= 0.5].index.tolist()

# Remove the target column itself from the list
high_correlation_columns.remove(target_column)

# Print the result
print("Columns with correlation >= 0.5 to '{}':".format(target_column))
print(high_correlation_columns)
```

Columns with correlation >= 0.5 to 'mental_disorders':
['substance_use_disorders', 'skin_and_subcutaneous_diseases', 'musculoskeletal_disorders', 'neoplasms', 'neurological_disorders', 'sense_organ_diseases']

```
def convert_to_actual_dalys(percent_str, total):
    # Convert percent_str to float
    percent = float(percent_str)
    # Perform the calculation
    return (percent / 100) * total


total_population = 100000


for col in df1.columns:
    # Exclude 'Entity', 'Code', and 'Year' columns
```

```
    if col not in ['Entity', 'Code', 'Year']:
        actual_col_name = col
        df1[actual_col_name] = df1[col].apply(convert_to_actual_dalys, total=total_population)

merged_df = pd.merge(df, df1[['Entity','Year','substance_use_disorders', 'skin_and_subcutaneous_diseases', 'musculoskeletal_disorders', 'neoplasms', 'neurological_disorders', 'sense_organ_diseases','mental_disorders']], on=['Year', 'Entity'])

# Display the merged DataFrame
merged_df
print(merged_df.columns)


Index(['Entity', 'Code', 'Year', 'low_physical_activity',
       'non_exclusive_breastfeeding', 'air_pollution', 'child_wasting',
       'high_systolic_blood_pressure', 'high_fasting_plasma_glucose',
       'child_stunting', 'high_body_mass_index', 'secondhand_smoke',
       'unsafe_sanitation', 'unsafe_water_source', 'diet_low_in_vegetables',
       'diet_low_in_fruits', 'diet_high_in_sodium', 'drug_use',
       'household_air_pollution', 'high_ldl_cholesterol', 'iron_deficiency',
       'zinc_deficiency', 'smoking', 'vitamin_a_deficiency',
       'ambient_particulate_matter_pollution', 'substance_use_disorders',
       'skin_and_subcutaneous_diseases', 'musculoskeletal_disorders',
       'neoplasms', 'neurological_disorders', 'sense_organ_diseases',
       'mental_disorders'],
      dtype='object')

merged_df

corr_matrix = merged_df.drop(['Year','Entity','Code'],axis =1).corr()
target_column = 'mental_disorders'
related_correlations = corr_matrix[target_column]


plt.figure(figsize=(25, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix of Disease Burden by Sub-Category')
plt.show()
```




```
missing_values = merged_df.isnull().sum()
print("Missing Values:\n", missing_values)
```

Missing Values:

Entity	0
Code	570
Year	0
low_physical_activity	0
non_exclusive_breastfeeding	0
air_pollution	0
child_wasting	0
high_systolic_blood_pressure	0
high_fasting_plasma_glucose	0
child_stunting	0
high_body_mass_index	0
secondhand_smoke	0
unsafe_sanitation	0
unsafe_water_source	0
diet_low_in_vegetables	0
diet_low_in_fruits	0
diet_high_in_sodium	0
drug_use	0
household_air_pollution	0
high_ldl_cholesterol	0
iron_deficiency	0
zinc_deficiency	0
smoking	0
vitamin_a_deficiency	0
ambient_particulate_matter_pollution	0
substance_use_disorders	0
skin_and_subcutaneous_diseases	0
musculoskeletal_disorders	0
neoplasms	0
neurological_disorders	0

sense_organ_diseases 0
mental_disorders 0
dtype: int64

```
print("\nSummary Statistics:\n", merged_df.describe())
```

```
countries = merged_df['Entity'].unique()
countries.shape
```

array(['Afghanistan', 'African Region (WHO)', 'Albania', 'Algeria',
'American Samoa', 'Andorra', 'Angola', 'Antigua and Barbuda',
'Argentina', 'Armenia', 'Australia', 'Austria', 'Azerbaijan',
'Bahamas', 'Bahrain', 'Bangladesh', 'Barbados', 'Belarus',
'Belgium', 'Belize', 'Benin', 'Bermuda', 'Bhutan', 'Bolivia',
'Bosnia and Herzegovina', 'Botswana', 'Brazil', 'Brunei',
'Bulgaria', 'Burkina Faso', 'Burundi', 'Cambodia', 'Cameroon',
'Canada', 'Cape Verde', 'Central African Republic', 'Chad',
'Chile', 'China', 'Colombia', 'Comoros', 'Congo', 'Cook Islands',
'Costa Rica', "Cote d'Ivoire", 'Croatia', 'Cuba', 'Cyprus',
'Czechia', 'Democratic Republic of Congo', 'Denmark', 'Djibouti',
'Dominica', 'Dominican Republic', 'East Asia & Pacific (WB)',
'East Timor', 'Eastern Mediterranean Region (WHO)', 'Ecuador',
'Egypt', 'El Salvador', 'England', 'Equatorial Guinea', 'Eritrea',
'Estonia', 'Eswatini', 'Ethiopia', 'Europe & Central Asia (WB)',
'European Region (WHO)', 'Fiji', 'Finland', 'France', 'G20',
'Gabon', 'Gambia', 'Georgia', 'Germany', 'Ghana', 'Greece',
'Greenland', 'Grenada', 'Guam', 'Guatemala', 'Guinea',
'Guinea-Bissau', 'Guyana', 'Haiti', 'Honduras', 'Hungary',
'Iceland', 'India', 'Indonesia', 'Iran', 'Iraq', 'Ireland',
'Israel', 'Italy', 'Jamaica', 'Japan', 'Jordan', 'Kazakhstan',
'Kenya', 'Kiribati', 'Kuwait', 'Kyrgyzstan', 'Laos',
'Latin America & Caribbean (WB)', 'Latvia', 'Lebanon', 'Lesotho',
'Liberia', 'Libya', 'Lithuania', 'Luxembourg', 'Madagascar',
'Malawi', 'Malaysia', 'Maldives', 'Mali', 'Malta',
'Marshall Islands', 'Mauritania', 'Mauritius', 'Mexico',
'Micronesia (country)', 'Middle East & North Africa (WB)',
'Moldova', 'Monaco', 'Mongolia', 'Montenegro', 'Morocco',
'Mozambique', 'Myanmar', 'Namibia', 'Nauru', 'Nepal',
'Netherlands', 'New Zealand', 'Nicaragua', 'Niger', 'Nigeria',
'Niue', 'North America (WB)', 'North Korea', 'North Macedonia',
'Northern Ireland', 'Northern Mariana Islands', 'Norway',
'OECD Countries', 'Oman', 'Pakistan', 'Palau', 'Palestine',
'Panama', 'Papua New Guinea', 'Paraguay', 'Peru', 'Philippines',
'Poland', 'Portugal', 'Puerto Rico', 'Qatar',
'Region of the Americas (WHO)', 'Romania', 'Russia', 'Rwanda',
'Saint Kitts and Nevis', 'Saint Lucia',
'Saint Vincent and the Grenadines', 'Samoa', 'San Marino',
'Sao Tome and Principe', 'Saudi Arabia', 'Scotland', 'Senegal',
'Serbia', 'Seychelles', 'Sierra Leone', 'Singapore', 'Slovakia',
'Slovenia', 'Solomon Islands', 'Somalia', 'South Africa',

'South Asia (WB)', 'South Korea', 'South Sudan',
'South-East Asia Region (WHO)', 'Spain', 'Sri Lanka',
'Sub-Saharan Africa (WB)', 'Sudan', 'Suriname', 'Sweden',
'Switzerland', 'Syria', 'Taiwan', 'Tajikistan', 'Tanzania',
'Thailand', 'Togo', 'Tokelau', 'Tonga', 'Trinidad and Tobago',
'Tunisia', 'Turkey', 'Turkmenistan', 'Tuvalu', 'Uganda', 'Ukraine',
'United Arab Emirates', 'United Kingdom', 'United States',
'United States Virgin Islands', 'Uruguay', 'Uzbekistan', 'Vanuatu',
'Venezuela', 'Vietnam', 'Wales', 'Western Pacific Region (WHO)',
'World', 'Yemen', 'Zambia', 'Zimbabwe'], dtype=object)

```
import numpy as np

# List of 195 recognized countries
recognized_countries = [
    "Afghanistan", "Albania", "Algeria", "Andorra", "Angola", "Antigua and Barbuda", "Argentina", "Armenia",
    "Australia", "Austria", "Azerbaijan", "Bahamas", "Bahrain", "Bangladesh", "Barbados", "Belarus", "Belgium",
    "Belize", "Benin", "Bhutan", "Bolivia", "Bosnia and Herzegovina", "Botswana", "Brazil", "Brunei", "Bulgaria",
    "Burkina Faso", "Burundi", "Cabo Verde", "Cambodia", "Cameroon", "Canada", "Central African Republic", "Chad",
    "Chile", "China", "Colombia", "Comoros", "Congo, Democratic Republic of the", "Congo, Republic of the",
    "Costa Rica", "Croatia", "Cuba", "Cyprus", "Czech Republic", "Denmark", "Djibouti", "Dominica", "Dominican Republic",
    "East Timor (Timor-Leste)", "Ecuador", "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea", "Estonia", "Eswatini",
    "Ethiopia", "Fiji", "Finland", "France", "Gabon", "Gambia", "Georgia", "Germany", "Ghana", "Greece", "Grenada",
    "Guatemala", "Guinea", "Guinea-Bissau", "Guyana", "Haiti", "Honduras", "Hungary", "Iceland", "India", "Indonesia",
    "Iran", "Iraq", "Ireland", "Israel", "Italy", "Jamaica", "Japan", "Jordan", "Kazakhstan", "Kenya", "Kiribati",
    "Korea, North", "Korea, South", "Kosovo", "Kuwait", "Kyrgyzstan", "Laos", "Latvia", "Lebanon", "Lesotho", "Liberia",
    "Libya", "Liechtenstein", "Lithuania", "Luxembourg", "Madagascar", "Malawi", "Malaysia", "Maldives", "Mali",
    "Malta", "Marshall Islands", "Mauritania", "Mauritius", "Mexico", "Micronesia", "Moldova", "Monaco", "Mongolia",
    "Montenegro", "Morocco", "Mozambique", "Myanmar", "Namibia", "Nauru", "Nepal", "Netherlands", "New Zealand",
    "Nicaragua", "Niger", "Nigeria", "North Macedonia", "Norway", "Oman", "Pakistan", "Palau", "Palestine", "Panama",
    "Papua New Guinea", "Paraguay", "Peru", "Philippines", "Poland", "Portugal", "Qatar", "Romania", "Russia", "Rwanda",
    "Saint Kitts and Nevis", "Saint Lucia", "Saint Vincent and the Grenadines", "Samoa", "San Marino", "Sao Tome and Principe",
    "Saudi Arabia", "Senegal", "Serbia", "Seychelles", "Sierra Leone", "Singapore", "Slovakia", "Slovenia", "Solomon Islands",
    "Somalia", "South Africa", "South Sudan", "Spain", "Sri Lanka", "Sudan", "Suriname", "Sweden", "Switzerland", "Syria",
```

```
    "Taiwan", "Tajikistan", "Tanzania", "Thailand", "Togo", "Tonga", "Trinidad and Tobago", "Tunisia",
"Turkey", "Turkmenistan",
    "Tuvalu", "Uganda", "Ukraine", "United Arab Emirates", "United Kingdom", "United States",
"Uruguay", "Uzbekistan", "Vanuatu",
    "Vatican City (Holy See)", "Venezuela", "Vietnam", "Yemen", "Zambia", "Zimbabwe"
]


# Filter out non-country entities
filtered_entities = [country for country in countries if country in recognized_countries]

# Print the filtered list
print(filtered_entities)

len(filtered_entities)
```

185

```
recognized_countries_set = set(filtered_entities)

# Filter the dataframe to include only rows where the 'entity' column is in the recognized countries set
merged_df = merged_df[merged_df['Entity'].isin(recognized_countries_set)]
```

Lets start PCA

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


data_to_scale = merged_df.drop(columns=['Year', 'Entity','Code'])

# Scale the data
scaler = StandardScaler()
merged_df_scaled = scaler.fit_transform(data_to_scale)

# Perform PCA

pca = PCA()
pca_out = pca.fit_transform(merged_df_scaled)


pd.DataFrame({'Center': scaler.mean_
              , 'Scale': scaler.scale_}
             , index=data_to_scale.columns)
```

	Center	Scale
low_physical_activity	**6.183756e+04**	**1.931739e+05**

non_exclusive_breastfeeding	1.405859e+05	5.474627e+05
air_pollution	1.287456e+06	5.875304e+06
child_wasting	9.337180e+05	4.022350e+06
high_systolic_blood_pressure	1.013699e+06	3.894411e+06
high_fasting_plasma_glucose	6.231589e+05	2.323884e+06
child_stunting	2.162215e+05	1.033723e+06
high_body_mass_index	5.675718e+05	1.767211e+06
secondhand_smoke	2.069046e+05	9.895219e+05
unsafe_sanitation	4.167442e+05	2.100466e+06
unsafe_water_source	5.797473e+05	2.858931e+06
diet_low_in_vegetables	6.166332e+04	2.466170e+05
diet_low_in_fruits	1.318412e+05	6.228871e+05
diet_high_in_sodium	2.026395e+05	1.340764e+06
drug_use	1.346634e+05	5.809032e+05
household_air_pollution	7.858722e+05	3.737993e+06
high_ldl_cholesterol	4.403610e+05	1.593582e+06

iron_deficiency	**1.611358e+05**	**9.209309e+05**
zinc_deficiency	**4.364368e+03**	**2.376293e+04**
smoking	**9.660387e+05**	**4.215962e+06**
vitamin_a_deficiency	**5.607599e+04**	**2.574250e+05**
ambient_particulate_matter_pollution	**4.874069e+05**	**2.429460e+06**
substance_use_disorders	**1.230150e+03**	**9.888776e+02**
skin_and_subcutaneous_diseases	**1.592032e+03**	**7.186604e+02**
musculoskeletal_disorders	**4.872905e+03**	**3.159057e+03**
neoplasms	**9.285721e+03**	**6.243723e+03**
neurological_disorders	**3.554023e+03**	**1.766960e+03**
sense_organ_diseases	**1.808128e+03**	**7.968354e+02**
mental_disorders	**4.833573e+03**	**2.402449e+03**

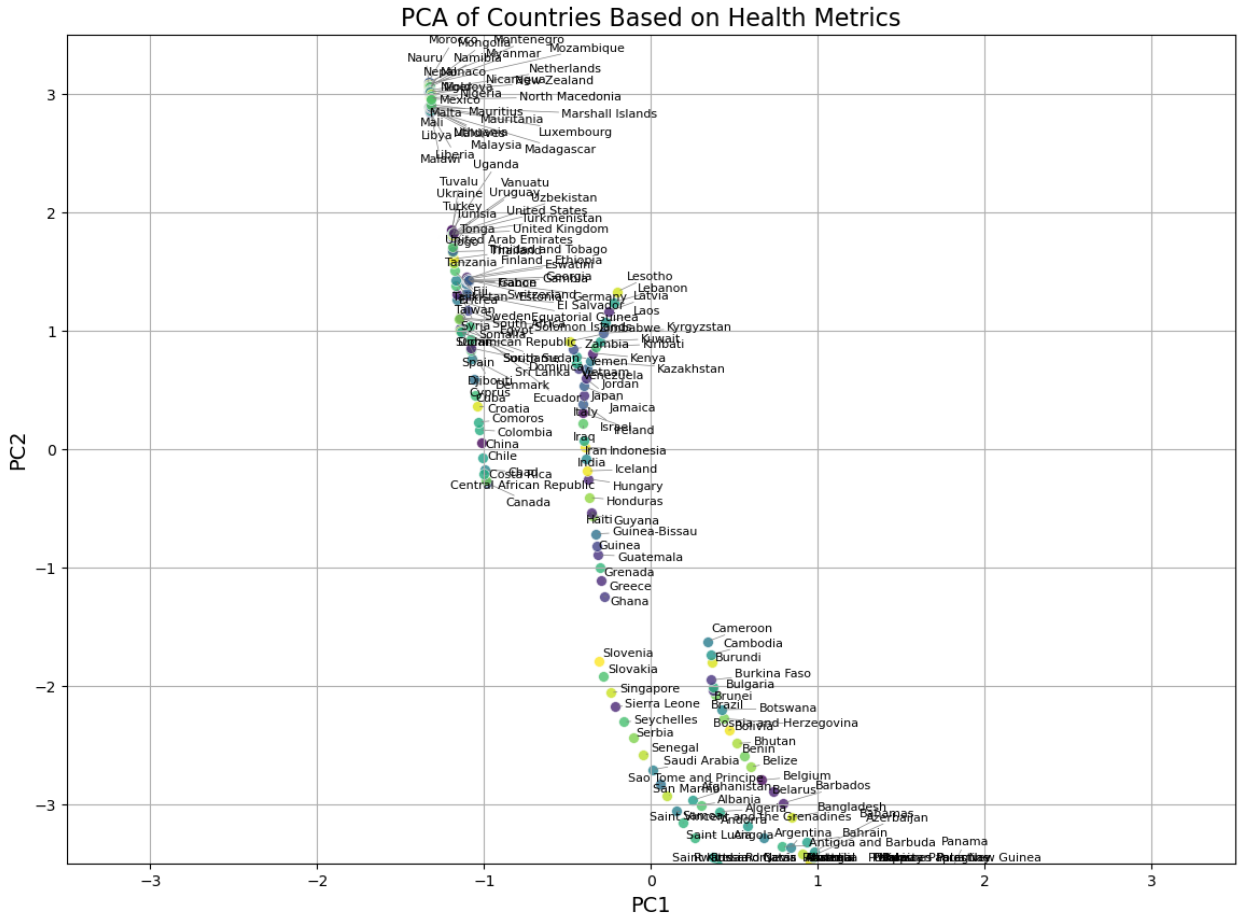
print("Number of Principal Components:", pca.n_components_)

Number of Principal Components: 29

pca.explained_variance_

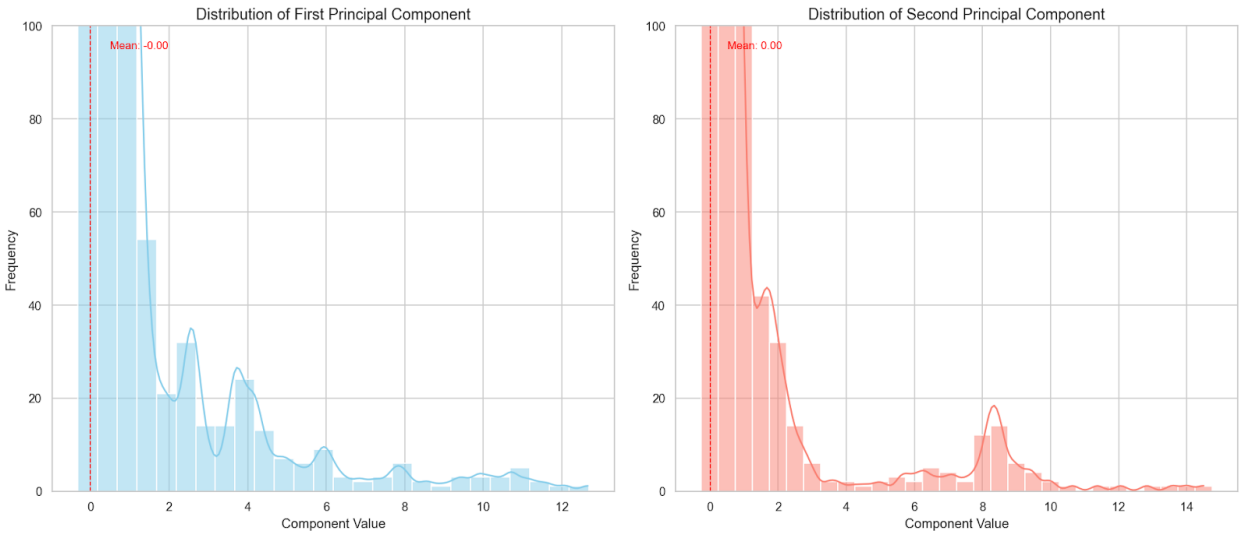
array([1.54999845e+01, 6.53276398e+00, 3.61909967e+00, 7.32560579e-01,
6.25741276e-01, 3.93517665e-01, 3.34616031e-01, 2.91070803e-01,
2.35397005e-01, 1.73817632e-01, 1.35456760e-01, 8.71869269e-02,
8.19874773e-02, 6.25877466e-02, 5.93653146e-02, 4.58567443e-02,
3.21872375e-02, 1.63381244e-02, 1.46354767e-02, 8.10614164e-03,
6.74050510e-03, 4.71839577e-03, 3.51122148e-03, 2.72828411e-03,

```
    2.00595908e-03, 1.77739354e-03, 1.18237924e-03, 2.81399207e-04,
    3.56051580e-06])
components_df = pd.DataFrame(pca.components_.T, index=data_to_scale.columns,
columns=[f'PC{i+1}' for i in range(pca.n_components_)])


from adjustText import adjust_text

np.random.seed(0)
colors = np.random.rand(len(merged_df['Entity'].unique()))

# Create the plot
fig, ax1 = plt.subplots(figsize=(14, 10))

ax1.set_xlim(-3.5, 3.5)
ax1.set_ylim(-3.5, 3.5)

texts = []
# Plot country names for PC1 and PC2
for i, label in enumerate(merged_df['Entity'].unique().tolist()):
    x = pca_out[i, 0]
    y = pca_out[i, 1]
    ax1.scatter(x, y, color=plt.cm.viridis(colors[i]), s=50, alpha=0.8, edgecolor='w', linewidth=0.5)
    texts.append(ax1.text(x, y, label, fontsize=8))

# Use adjustText to minimize overlapping of text labels
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='gray', lw=0.5))

ax1.set_xlabel("PC1", fontsize=14)
ax1.set_ylabel("PC2", fontsize=14)
ax1.set_title("PCA of Countries Based on Health Metrics", fontsize=16)
ax1.grid(True)

plt.savefig('PCA_of_Countries_Health_Metrics.png', dpi=300, bbox_inches='tight')

plt.show()
```

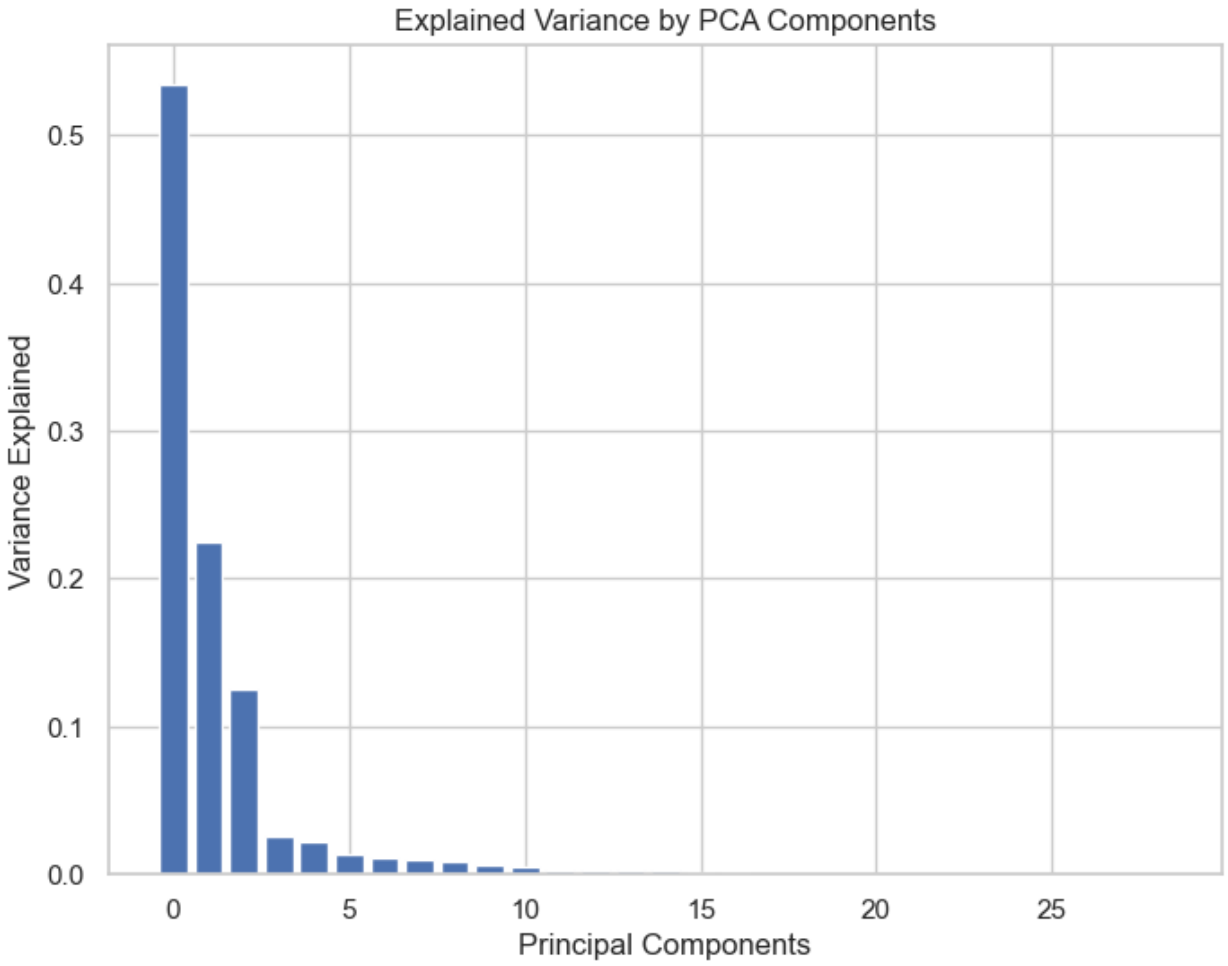



```
merged_df_scaled = pd.DataFrame(merged_df_scaled,columns= data_to_scale.columns)
sns.set(style='whitegrid')
fig, ax = plt.subplots(1, 2, figsize=(16, 7), dpi=120)

sns.histplot(merged_df_scaled[merged_df_scaled.columns[0]], kde=True, color='skyblue', ax=ax[0], binwidth=0.5)
ax[0].set_title('Distribution of First Principal Component', fontsize=14)
ax[0].set_xlabel('Component Value', fontsize=12)
ax[0].set_ylabel('Frequency', fontsize=12)
ax[0].set_ylim(0, 100)  # Adjusted y-axis range

sns.histplot(merged_df_scaled[merged_df_scaled.columns[1]], kde=True, color='salmon', ax=ax[1], binwidth=0.5)
ax[1].set_title('Distribution of Second Principal Component', fontsize=14)
ax[1].set_xlabel('Component Value', fontsize=12)
ax[1].set_ylabel('Frequency', fontsize=12)
ax[1].set_ylim(0, 100)  # Adjusted y-axis range

# Adding a vertical line for the mean
mean_val_0 = merged_df_scaled[merged_df_scaled.columns[0]].mean()
mean_val_1 = merged_df_scaled[merged_df_scaled.columns[1]].mean()
```

```
ax[0].axvline(mean_val_0, color='red', linestyle='dashed', linewidth=1)
ax[1].axvline(mean_val_1, color='red', linestyle='dashed', linewidth=1)

# Add text annotation for mean
ax[0].text(mean_val_0 + 0.5, 95, f'Mean: {mean_val_0:.2f}', color = 'red', fontsize=10)
ax[1].text(mean_val_1 + 0.5, 95, f'Mean: {mean_val_1:.2f}', color = 'red', fontsize=10)

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




```
# Plotting the explained variance by PCA
plt.figure(figsize=(8, 6))
plt.bar(range(len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
plt.xlabel('Principal Components')
plt.ylabel('Variance Explained')
plt.title('Explained Variance by PCA Components')
plt.show()
```

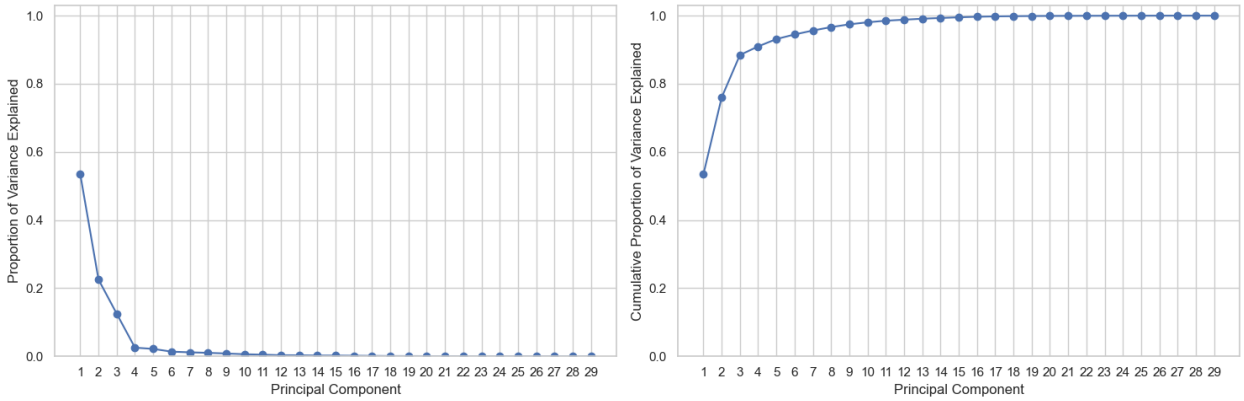



```
pca.explained_variance_ratio_
```

```
array([5.34385920e-01, 2.25227135e-01, 1.24774054e-01, 2.52561582e-02,
       2.15733976e-02, 1.35671297e-02, 1.15364048e-02, 1.00351158e-02,
       8.11567557e-03, 5.99263152e-03, 4.67008115e-03, 3.00590405e-03,
       2.82664499e-03, 2.15780929e-03, 2.04671097e-03, 1.58098213e-03,
       1.10970476e-03, 5.63282090e-04, 5.04580679e-04, 2.79471761e-04,
       2.32389331e-04, 1.62673986e-04, 1.21054787e-04, 9.40618112e-05,
       6.91585394e-05, 6.12783892e-05, 4.07643518e-05, 9.70167258e-06,
       1.22754285e-07])
```

```
# Plot of proportion of variance explained
fig, ax = plt.subplots(1, 2, figsize=(15, 5))

# Plot of proportion of variance explained
ax[0].plot(range(1, 30), pca.explained_variance_ratio_, marker='o')
ax[0].set_xlabel('Principal Component')
ax[0].set_ylabel('Proportion of Variance Explained')
ax[0].set_ylim(0, 1.03)
ax[0].set_xticks(range(1, 30))

# Plot of cumulative proportion of variance explained
```

```
ax[1].plot(range(1, 30), np.cumsum(pca.explained_variance_ratio_), marker='o')
ax[1].set_xlabel('Principal Component')
ax[1].set_ylabel('Cumulative Proportion of Variance Explained')
ax[1].set_ylim(0, 1.03)
ax[1].set_xticks(range(1, 30))

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




```
numeric_columns = merged_df_scaled.select_dtypes(include=['float64', 'int64']).columns
merged_numeric = merged_df_scaled[numeric_columns]

# Standardize the numeric data
scaler = StandardScaler()
merged_numeric_scaled = scaler.fit_transform(merged_numeric)

# Perform PCA to reduce to 4 components
pca = PCA(n_components=4)
pca_out = pca.fit_transform(merged_numeric_scaled)

# Print PCA components' loadings
loadings = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i in range(4)], index=numeric_columns)
print("PCA Loadings:")
print(loadings)
```

PCA Loadings:

	PC1	PC2	PC3	PC4
low_physical_activity	0.212415	0.145885	-0.118299	0.148719
non_exclusive_breastfeeding	0.198540	-0.157623	0.172540	-0.070970
air_pollution	0.249433	-0.004436	0.010712	-0.141176
child_wasting	0.203092	-0.165662	0.209707	-0.032775
high_systolic_blood_pressure	0.224138	0.133343	-0.155266	-0.008427
high_fasting_plasma_glucose	0.227395	0.129165	-0.125603	0.042588
child_stunting	0.193075	-0.173807	0.230871	-0.024464
high_body_mass_index	0.197867	0.165229	-0.168983	0.247438
secondhand_smoke	0.231702	0.081320	-0.097590	-0.213588
unsafe_sanitation	0.200048	-0.162379	0.230774	0.045156
unsafe_water_source	0.202230	-0.159259	0.225930	0.053575

diet_low_in_vegetables 0.223073 -0.003001 0.061173 0.199293
diet_low_in_fruits 0.237983 0.091141 -0.098777 -0.049549
diet_high_in_sodium 0.179092 0.152992 -0.227137 -0.321276
drug_use 0.200096 0.138842 -0.136886 0.133722
household_air_pollution 0.236732 -0.074069 0.099677 -0.143585
high_ldl_cholesterol 0.221231 0.128952 -0.130133 0.169875
iron_deficiency 0.222299 -0.059561 0.128716 0.096307
zinc_deficiency 0.179360 -0.170202 0.241040 0.061468
smoking 0.212048 0.153815 -0.172452 -0.102329
vitamin_a_deficiency 0.178420 -0.188677 0.228016 0.018895
ambient_particulate_matter_pollution 0.230421 0.097758 -0.120554 -0.113654
substance_use_disorders -0.009187 0.253340 0.152362 0.655319
skin_and_subcutaneous_diseases -0.026409 0.286467 0.254842 -0.308608
musculoskeletal_disorders -0.020727 0.316480 0.269477 -0.066616
neoplasms -0.022955 0.290043 0.219899 0.113176
neurological_disorders -0.034357 0.315475 0.279734 -0.045609
sense_organ_diseases 0.003371 0.302272 0.197044 -0.087096
mental_disorders -0.033414 0.283241 0.262180 -0.190953

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
df = pd.read_csv('./Final_data.csv')

#df = df[df['Year'] < 2017]
#years = df['Year'].astype(str)  # Convert year to string to ensure it's treated as a categorical variable
#dummies = pd.get_dummies(years, prefix='year', drop_first=True)
#df = pd.concat([df, dummies], axis=1)
df.columns
```

```
Index(['Entity', 'Code_x', 'Year', 'low_physical_activity',
       'non_exclusive_breastfeeding', 'air_pollution', 'child_wasting',
       'high_systolic_blood_pressure', 'high_fasting_plasma_glucose',
       'child_stunting', 'high_body_mass_index', 'secondhand_smoke',
       'unsafe_sanitation', 'unsafe_water_source', 'diet_low_in_vegetables',
       'diet_low_in_fruits', 'diet_high_in_sodium', 'drug_use',
       'household_air_pollution', 'high_ldl_cholesterol', 'iron_deficiency',
       'zinc_deficiency', 'smoking', 'vitamin_a_deficiency',
       'ambient_particulate_matter_pollution', 'Code_y',
       'substance_use_disorders', 'skin_and_subcutaneous_diseases',
       'musculoskeletal_disorders', 'neoplasms', 'neurological_disorders',
       'sense_organ_diseases', 'mental_disorders'],
      dtype='object')
```

```
features = [
```

```
    'low_physical_activity', 'non_exclusive_breastfeeding', 'air_pollution',
    'child_wasting', 'high_systolic_blood_pressure', 'high_fasting_plasma_glucose',
    'high_body_mass_index', 'secondhand_smoke', 'unsafe_sanitation', 'unsafe_water_source',
    'diet_low_in_vegetables', 'diet_low_in_fruits', 'diet_high_in_sodium', 'drug_use',
    'household_air_pollution', 'high_ldl_cholesterol', 'iron_deficiency', 'zinc_deficiency',
    'smoking', 'vitamin_a_deficiency', 'ambient_particulate_matter_pollution',
    'substance_use_disorders', 'skin_and_subcutaneous_diseases', 'musculoskeletal_disorders',
    'neoplasms', 'neurological_disorders', 'sense_organ_diseases', 'child_stunting']

target = 'mental_disorders'

# Normalize the dataset
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
df[target] = scaler.fit_transform(df[[target]])
df

pca = PCA(n_components=4)  # Select number of components
merged_df_pca = pca.fit_transform(df.drop(columns=['Entity','Year','Code_x','Code_y']))
merged_df_pca
```

```
array([[-0.64612718, -0.0248205 , -0.04948753,  0.03328347],
       [-0.65036896, -0.02247287, -0.04492745,  0.03350559],
       [-0.65257246, -0.01691882, -0.03583051,  0.0351256 ],
       ...,
       [-0.45180058, -0.06040674, -0.05563566,  0.00464654],
       [-0.42997334, -0.05967361, -0.0546987 ,  0.00425658],
       [-0.41675952, -0.05889445, -0.0550508 ,  0.00457749]])
```

```
column_names = [f'PC{i}' for i in range(1, merged_df_pca.shape[1] + 1)]

# Create a new DataFrame
pca_df = pd.DataFrame(data=merged_df_pca, columns=column_names)
pca_df
```

	PC1	PC2	PC3	PC4
0	-0.646127	-0.024821	-0.049488	0.033283
1	-0.650369	-0.022473	-0.044927	0.033506
2	-0.652572	-0.016919	-0.035831	0.035126
3	-0.661566	-0.008751	-0.028581	0.042535
4	-0.673319	-0.004180	-0.025247	0.046511

	PC1	PC2	PC3	PC4
...	...	...	...	...
5545	-0.486504	-0.061376	-0.057477	0.005150
5546	-0.468311	-0.060805	-0.056375	0.004515
5547	-0.451801	-0.060407	-0.055636	0.004647
5548	-0.429973	-0.059674	-0.054699	0.004257
5549	-0.416760	-0.058894	-0.055051	0.004577

5550 rows × 4 columns

```
X = pca_df.values  # Features (PC components)
y = df['mental_disorders'].values.ravel()  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
param_grid_rbf = {'C': [0.01, 0.1, 1, 10,50,100], 'gamma': [0.01, 0.1, 1, 10,50,100]}

# Initialize GridSearchCV with RBF kernel
grid_search_rbf = GridSearchCV(SVR(kernel='rbf', verbose=True), param_grid_rbf,
scoring='neg_mean_squared_error', cv=10, n_jobs=-1)

# Fit the GridSearchCV on the training data
grid_search_rbf.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params_rbf = grid_search_rbf.best_params_
best_estimator_rbf = grid_search_rbf.best_estimator_

print("Best parameters for RBF kernel:", best_params_rbf)

# Evaluate the best RBF model on the test set
accuracy_rbf = best_estimator_rbf.score(X_test, y_test)
print(f"Accuracy with the best RBF model: {accuracy_rbf * 100:.2f}%")
```

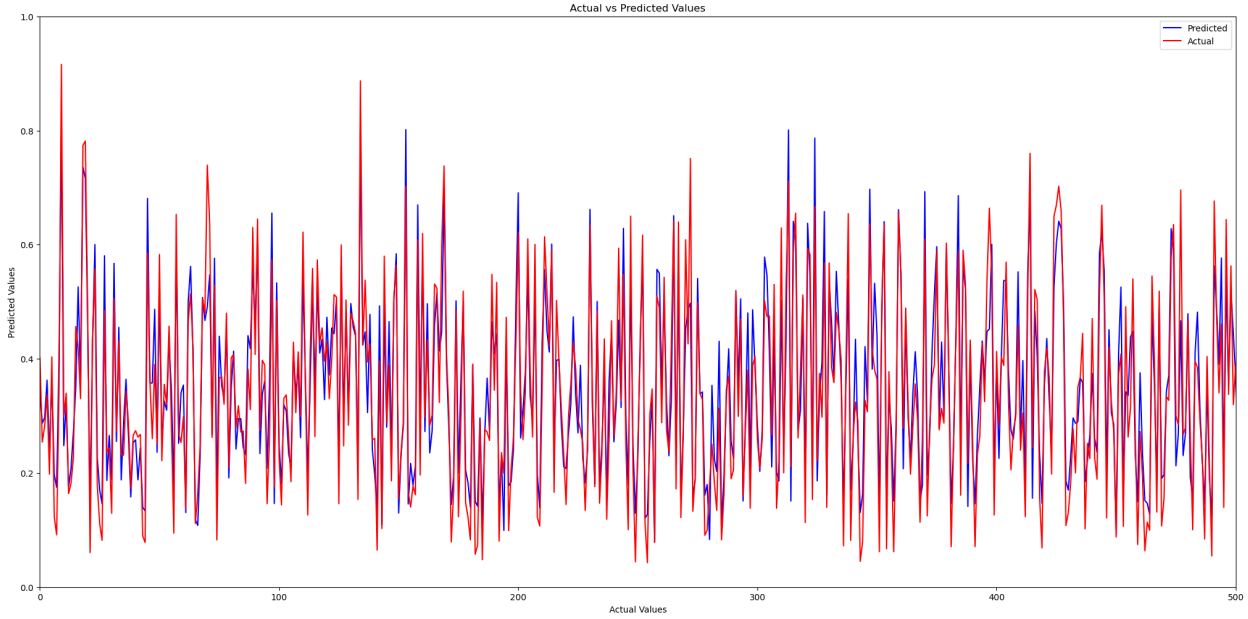
```
Best parameters for RBF kernel: {'C': 50, 'gamma': 10}
Accuracy with the best RBF model: 88.75%
```

```
model = SVR(kernel = 'rbf',C=10,gamma=1)

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)
```

```
# Evaluate the model
test_mse = mean_squared_error(y_test, y_pred)
test_r2 = r2_score(y_test, y_pred)
print(f'Test Mean Squared Error: {test_mse}')
print(f'Test R-squared: {test_r2}')
```

Test Mean Squared Error: 0.0038995784534656534
Test R-squared: 0.8727087538040207

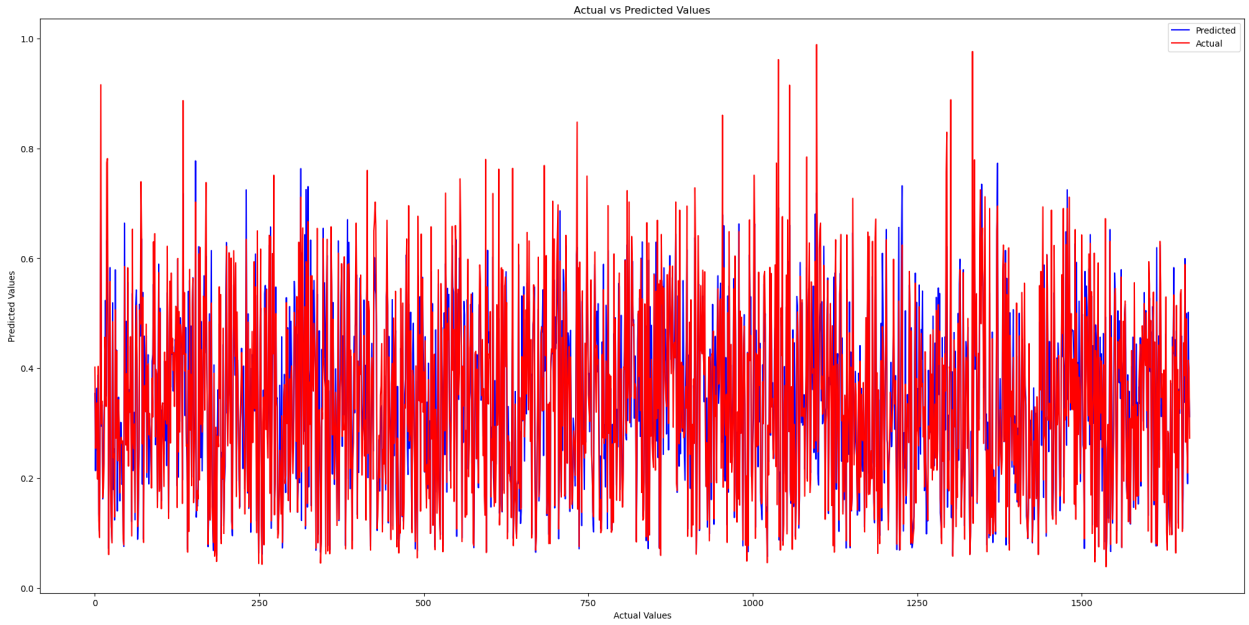
```
import matplotlib.pyplot as plt

# Plot actual vs predicted
plt.figure(figsize=(25, 12))  # Enlarge the plot
plt.plot(y_pred, color='blue', label='Predicted')
plt.plot(y_test, color='red', label='Actual')  # Plotting the line y=x for reference
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.legend()

# Set xlim and ylim
plt.xlim(0, 500)  # Adjust the values as needed
plt.ylim(0, 1)  # Adjust the values as needed
plt.show()
```

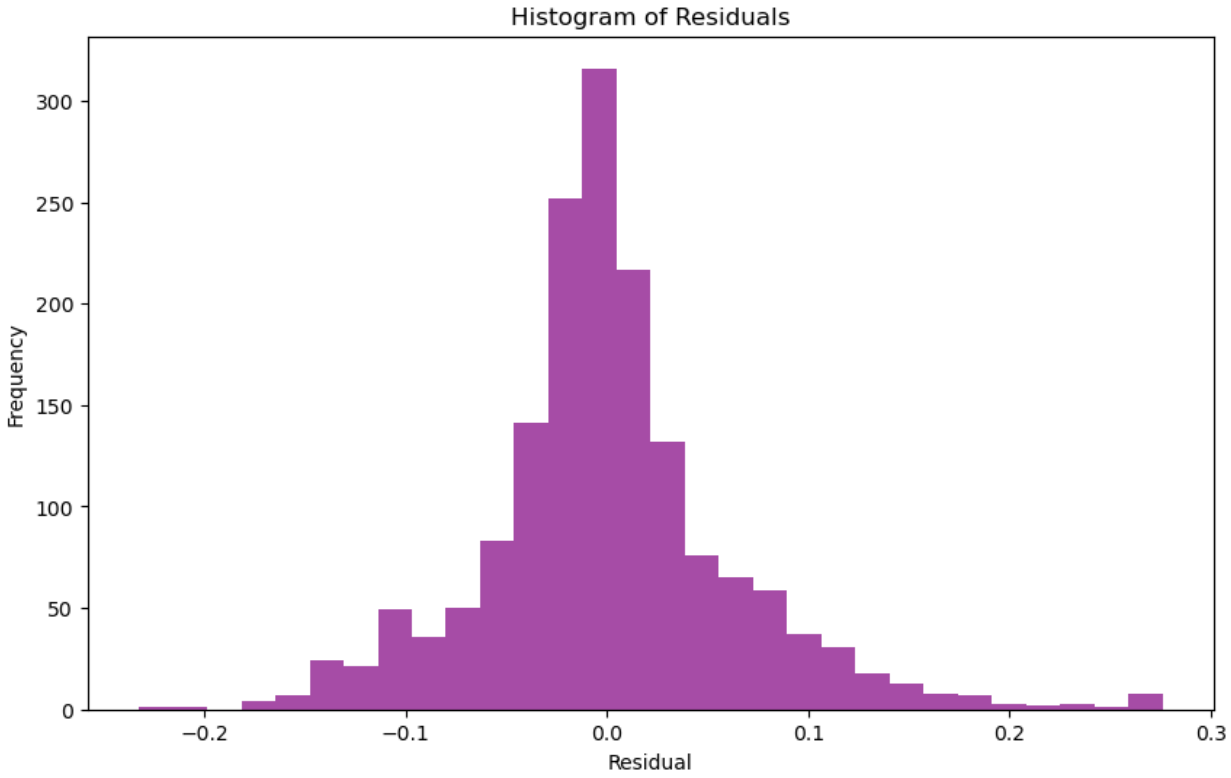



```
residuals = y_test - y_pred.flatten()
# Plot histogram of residuals
plt.figure(figsize=(10, 6))
plt.hist(residuals, bins=30, color='purple', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.show()
```

```
target = 'mental_disorders'

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

# Model Evaluation
y_pred = lm_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

plt.figure(figsize=(25, 12))  # Enlarge the plot
plt.plot(y_pred, color='blue', label='Predicted')
plt.plot(y_test, color='red', label='Actual')  # Plotting the line y=x for reference
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.legend()

# Plotting residuals
residuals = y_test - y_pred.flatten()
# Plot histogram of residuals
plt.figure(figsize=(10, 6))
plt.hist(residuals, bins=30, color='purple', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.show()
```

Mean Squared Error: 0.003954179689872241
R-squared: 0.8709264433545774





Lstm

from sklearn.model_selection import train_test_split

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Dropout
from keras.callbacks import ModelCheckpoint, EarlyStopping,ReduceLROnPlateau
import numpy as np
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Dropout

# Assuming merged_numeric_scaled is the standardized feature data
# Extract the target variable before performing PCA
target_column = 'mental_disorders'
y = df[target_column].values
# Drop the target column from the feature data
X = df.drop(columns=[target_column,'Entity','Code_x','Code_y']).values
# Perform PCA to reduce to 4 components
pca = PCA(n_components=4)
pca_out = pca.fit_transform(X)
# Function to create sequences
def create_sequences(data, target, n_timesteps):
    sequences = []
    targets = []
    for i in range(len(data) - n_timesteps + 1):
        seq = data[i:i + n_timesteps]
        sequences.append(seq)
        targets.append(target[i + n_timesteps - 1])
    return np.array(sequences), np.array(targets)

# Assuming each sample should be a sequence of 10 timesteps
n_timesteps = 10
n_features = pca_out.shape[1]
# Create sequences
X_seq, y_seq = create_sequences(pca_out, y, n_timesteps)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.3, random_state=42)
# Print the number of features
print(n_features)
# Create the RNN model with Bidirectional LSTM layers for regression task
model2 = Sequential()
# Bidirectional LSTM layer with Dropout regularization
model2.add(Bidirectional(LSTM(units=64, return_sequences=True, input_shape=(n_timesteps,
n_features))))
model2.add(Dropout(0.3))
# Second Bidirectional LSTM layer
model2.add(Bidirectional(LSTM(units=64, return_sequences=True)))
model2.add(Dropout(0.3))
# Third Bidirectional LSTM layer
model2.add(Bidirectional(LSTM(units=64)))
model2.add(Dropout(0.3))
# Output layer for regression task
model2.add(Dense(units=1))
# Compile the model with rmsprop optimizer
model2.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
# Summary of the model
model2.summary()
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# Reduce learning rate on plateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)
# Train the model
history = model2.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test),
callbacks=[early_stopping, reduce_lr])
# Plot training & validation metrics
plt.figure(figsize=(12, 6))
# Plot loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
# Plot mean absolute error
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model Mean Absolute Error')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.show()
# Make predictions
y_pred = model2.predict(X_test)
# Plot actual vs. predicted values using a line plot
plt.figure(figsize=(10, 6))
plt.plot(range(len(y_test)), y_test, label='Actual Values', linestyle='-', marker='o')
plt.plot(range(len(y_test)), y_pred, label='Predicted Values', linestyle='-', marker='x')
plt.title('Actual vs. Predicted Values')
plt.xlabel('Sample Index')
plt.ylabel('Mental Disorders')
plt.legend()
plt.show()
```

```
/Applications/Anaconda/anaconda3/lib/python3.11/site-packages/keras/src/layer
s/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argum
ent to a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	?	0 (unbuilt)

dropout (Dropout)	?	0 (unbuilt)
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0 (unbuilt)
bidirectional_2 (Bidirectional)	?	0 (unbuilt)
dropout_2 (Dropout)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.0424 - mae:
0.1527 - val_loss: 0.0145 - val_mae: 0.0981 - learning_rate: 0.0010
Epoch 2/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0138 - mae:
0.0875 - val_loss: 0.0108 - val_mae: 0.0802 - learning_rate: 0.0010
Epoch 3/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0120 - mae:
0.0813 - val_loss: 0.0103 - val_mae: 0.0744 - learning_rate: 0.0010
Epoch 4/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 0.0110 - mae:
0.0779 - val_loss: 0.0092 - val_mae: 0.0705 - learning_rate: 0.0010
Epoch 5/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 0.0099 - mae:
0.0742 - val_loss: 0.0100 - val_mae: 0.0787 - learning_rate: 0.0010
Epoch 6/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 0.0101 - mae:
0.0755 - val_loss: 0.0087 - val_mae: 0.0665 - learning_rate: 0.0010
Epoch 7/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 0.0097 - mae:
0.0729 - val_loss: 0.0084 - val_mae: 0.0630 - learning_rate: 0.0010
Epoch 8/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 0.0091 - mae:
0.0701 - val_loss: 0.0081 - val_mae: 0.0638 - learning_rate: 0.0010
Epoch 9/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0089 - mae:
0.0708 - val_loss: 0.0092 - val_mae: 0.0724 - learning_rate: 0.0010
Epoch 10/100
```

```
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0092 - mae:
0.0712 - val_loss: 0.0081 - val_mae: 0.0621 - learning_rate: 0.0010
Epoch 11/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0087 - mae:
0.0688 - val_loss: 0.0082 - val_mae: 0.0637 - learning_rate: 0.0010
Epoch 12/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0084 - mae:
0.0681 - val_loss: 0.0082 - val_mae: 0.0669 - learning_rate: 0.0010
Epoch 13/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0086 - mae:
0.0693 - val_loss: 0.0080 - val_mae: 0.0629 - learning_rate: 0.0010
Epoch 14/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0088 - mae:
0.0685 - val_loss: 0.0078 - val_mae: 0.0640 - learning_rate: 0.0010
Epoch 15/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0085 - mae:
0.0670 - val_loss: 0.0080 - val_mae: 0.0661 - learning_rate: 0.0010
Epoch 16/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0084 - mae:
0.0675 - val_loss: 0.0078 - val_mae: 0.0633 - learning_rate: 0.0010
Epoch 17/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0083 - mae:
0.0668 - val_loss: 0.0076 - val_mae: 0.0613 - learning_rate: 0.0010
Epoch 18/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0079 - mae:
0.0650 - val_loss: 0.0079 - val_mae: 0.0655 - learning_rate: 0.0010
Epoch 19/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0083 - mae:
0.0663 - val_loss: 0.0076 - val_mae: 0.0617 - learning_rate: 0.0010
Epoch 20/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0077 - mae:
0.0641 - val_loss: 0.0086 - val_mae: 0.0643 - learning_rate: 0.0010
Epoch 21/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0081 - mae:
0.0652 - val_loss: 0.0079 - val_mae: 0.0640 - learning_rate: 0.0010
Epoch 22/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0077 - mae:
0.0636 - val_loss: 0.0078 - val_mae: 0.0638 - learning_rate: 0.0010
Epoch 23/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0081 - mae:
0.0655 - val_loss: 0.0075 - val_mae: 0.0607 - learning_rate: 0.0010
Epoch 24/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0076 - mae:
0.0625 - val_loss: 0.0073 - val_mae: 0.0589 - learning_rate: 0.0010
Epoch 25/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0074 - mae:
0.0615 - val_loss: 0.0075 - val_mae: 0.0597 - learning_rate: 0.0010
Epoch 26/100
122/122 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0082 - mae:
0.0646 - val_loss: 0.0075 - val_mae: 0.0610 - learning_rate: 0.0010
Epoch 27/100
```

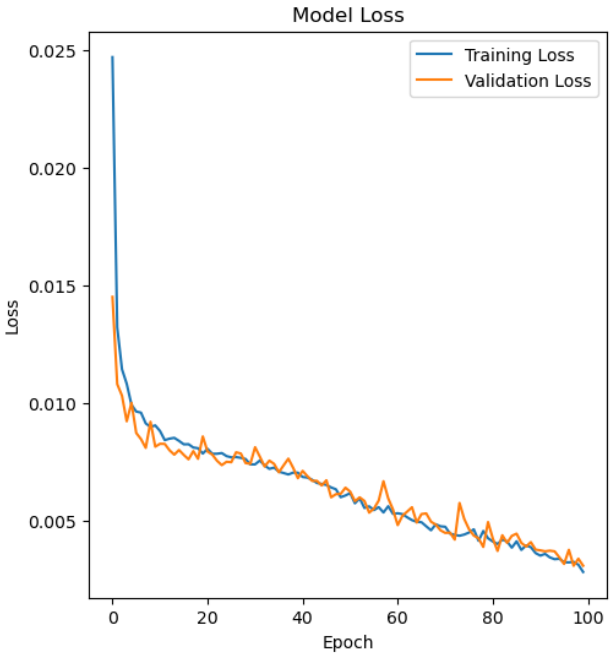
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0078 - mae: 0.0643 - val_loss: 0.0079 - val_mae: 0.0633 - learning_rate: 0.0010
Epoch 28/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0077 - mae: 0.0639 - val_loss: 0.0078 - val_mae: 0.0640 - learning_rate: 0.0010
Epoch 29/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0078 - mae: 0.0641 - val_loss: 0.0074 - val_mae: 0.0590 - learning_rate: 0.0010
Epoch 30/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0076 - mae: 0.0631 - val_loss: 0.0074 - val_mae: 0.0625 - learning_rate: 0.0010
Epoch 31/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0075 - mae: 0.0631 - val_loss: 0.0081 - val_mae: 0.0678 - learning_rate: 0.0010
Epoch 32/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0075 - mae: 0.0641 - val_loss: 0.0077 - val_mae: 0.0606 - learning_rate: 0.0010
Epoch 33/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0077 - mae: 0.0636 - val_loss: 0.0073 - val_mae: 0.0591 - learning_rate: 0.0010
Epoch 34/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0071 - mae: 0.0614 - val_loss: 0.0075 - val_mae: 0.0590 - learning_rate: 0.0010
Epoch 35/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0069 - mae: 0.0595 - val_loss: 0.0074 - val_mae: 0.0633 - learning_rate: 0.0010
Epoch 36/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0069 - mae: 0.0612 - val_loss: 0.0070 - val_mae: 0.0576 - learning_rate: 0.0010
Epoch 37/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0071 - mae: 0.0611 - val_loss: 0.0073 - val_mae: 0.0608 - learning_rate: 0.0010
Epoch 38/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0068 - mae: 0.0601 - val_loss: 0.0076 - val_mae: 0.0633 - learning_rate: 0.0010
Epoch 39/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0069 - mae: 0.0614 - val_loss: 0.0072 - val_mae: 0.0618 - learning_rate: 0.0010
Epoch 40/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0074 - mae: 0.0618 - val_loss: 0.0068 - val_mae: 0.0587 - learning_rate: 0.0010
Epoch 41/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0069 - mae: 0.0602 - val_loss: 0.0071 - val_mae: 0.0623 - learning_rate: 0.0010
Epoch 42/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0071 - mae: 0.0603 - val_loss: 0.0069 - val_mae: 0.0579 - learning_rate: 0.0010
Epoch 43/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0065 - mae: 0.0579 - val_loss: 0.0067 - val_mae: 0.0583 - learning_rate: 0.0010
Epoch 44/100

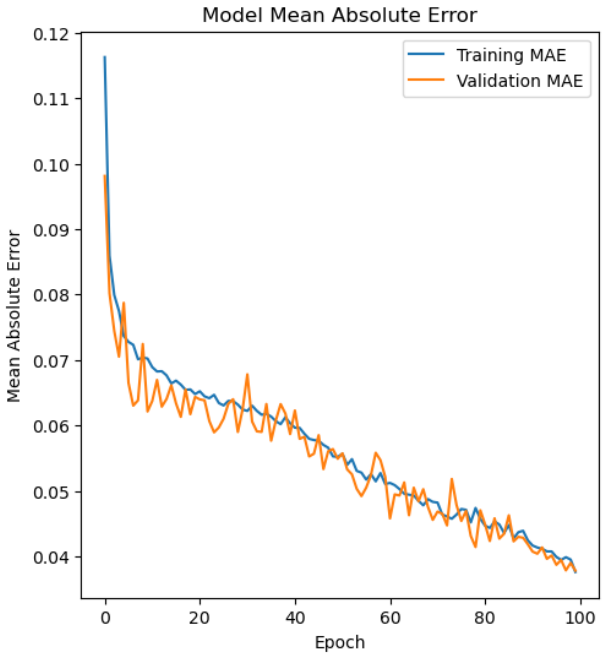
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0064 - mae: 0.0567 - val_loss: 0.0067 - val_mae: 0.0552 - learning_rate: 0.0010
Epoch 45/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0069 - mae: 0.0593 - val_loss: 0.0065 - val_mae: 0.0557 - learning_rate: 0.0010
Epoch 46/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0065 - mae: 0.0576 - val_loss: 0.0067 - val_mae: 0.0585 - learning_rate: 0.0010
Epoch 47/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0066 - mae: 0.0574 - val_loss: 0.0060 - val_mae: 0.0533 - learning_rate: 0.0010
Epoch 48/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0063 - mae: 0.0567 - val_loss: 0.0061 - val_mae: 0.0561 - learning_rate: 0.0010
Epoch 49/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 18ms/step - loss: 0.0059 - mae: 0.0555 - val_loss: 0.0061 - val_mae: 0.0564 - learning_rate: 0.0010
Epoch 50/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 20ms/step - loss: 0.0061 - mae: 0.0552 - val_loss: 0.0064 - val_mae: 0.0549 - learning_rate: 0.0010
Epoch 51/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0059 - mae: 0.0544 - val_loss: 0.0062 - val_mae: 0.0557 - learning_rate: 0.0010
Epoch 52/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0057 - mae: 0.0547 - val_loss: 0.0058 - val_mae: 0.0533 - learning_rate: 0.0010
Epoch 53/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0059 - mae: 0.0546 - val_loss: 0.0060 - val_mae: 0.0525 - learning_rate: 0.0010
Epoch 54/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0057 - mae: 0.0535 - val_loss: 0.0058 - val_mae: 0.0503 - learning_rate: 0.0010
Epoch 55/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0061 - mae: 0.0546 - val_loss: 0.0053 - val_mae: 0.0492 - learning_rate: 0.0010
Epoch 56/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0058 - mae: 0.0527 - val_loss: 0.0055 - val_mae: 0.0505 - learning_rate: 0.0010
Epoch 57/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0054 - mae: 0.0523 - val_loss: 0.0058 - val_mae: 0.0524 - learning_rate: 0.0010
Epoch 58/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0050 - mae: 0.0508 - val_loss: 0.0067 - val_mae: 0.0558 - learning_rate: 0.0010
Epoch 59/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0059 - mae: 0.0541 - val_loss: 0.0059 - val_mae: 0.0547 - learning_rate: 0.0010
Epoch 60/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0053 - mae: 0.0517 - val_loss: 0.0055 - val_mae: 0.0521 - learning_rate: 0.0010
Epoch 61/100

122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0056 - mae: 0.0527 - val_loss: 0.0048 - val_mae: 0.0458 - learning_rate: 0.0010
Epoch 62/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0050 - mae: 0.0501 - val_loss: 0.0052 - val_mae: 0.0495 - learning_rate: 0.0010
Epoch 63/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0051 - mae: 0.0505 - val_loss: 0.0054 - val_mae: 0.0493 - learning_rate: 0.0010
Epoch 64/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0052 - mae: 0.0506 - val_loss: 0.0056 - val_mae: 0.0513 - learning_rate: 0.0010
Epoch 65/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0052 - mae: 0.0513 - val_loss: 0.0049 - val_mae: 0.0463 - learning_rate: 0.0010
Epoch 66/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0047 - mae: 0.0486 - val_loss: 0.0053 - val_mae: 0.0505 - learning_rate: 0.0010
Epoch 67/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0047 - mae: 0.0489 - val_loss: 0.0053 - val_mae: 0.0484 - learning_rate: 0.0010
Epoch 68/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0045 - mae: 0.0475 - val_loss: 0.0049 - val_mae: 0.0502 - learning_rate: 0.0010
Epoch 69/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0048 - mae: 0.0485 - val_loss: 0.0048 - val_mae: 0.0475 - learning_rate: 0.0010
Epoch 70/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0048 - mae: 0.0486 - val_loss: 0.0046 - val_mae: 0.0456 - learning_rate: 0.0010
Epoch 71/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0046 - mae: 0.0480 - val_loss: 0.0045 - val_mae: 0.0468 - learning_rate: 0.0010
Epoch 72/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0041 - mae: 0.0444 - val_loss: 0.0045 - val_mae: 0.0465 - learning_rate: 0.0010
Epoch 73/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0044 - mae: 0.0458 - val_loss: 0.0042 - val_mae: 0.0447 - learning_rate: 0.0010
Epoch 74/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0045 - mae: 0.0465 - val_loss: 0.0057 - val_mae: 0.0518 - learning_rate: 0.0010
Epoch 75/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0042 - mae: 0.0458 - val_loss: 0.0051 - val_mae: 0.0478 - learning_rate: 0.0010
Epoch 76/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0047 - mae: 0.0488 - val_loss: 0.0046 - val_mae: 0.0454 - learning_rate: 0.0010
Epoch 77/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0048 - mae: 0.0470 - val_loss: 0.0043 - val_mae: 0.0469 - learning_rate: 0.0010
Epoch 78/100

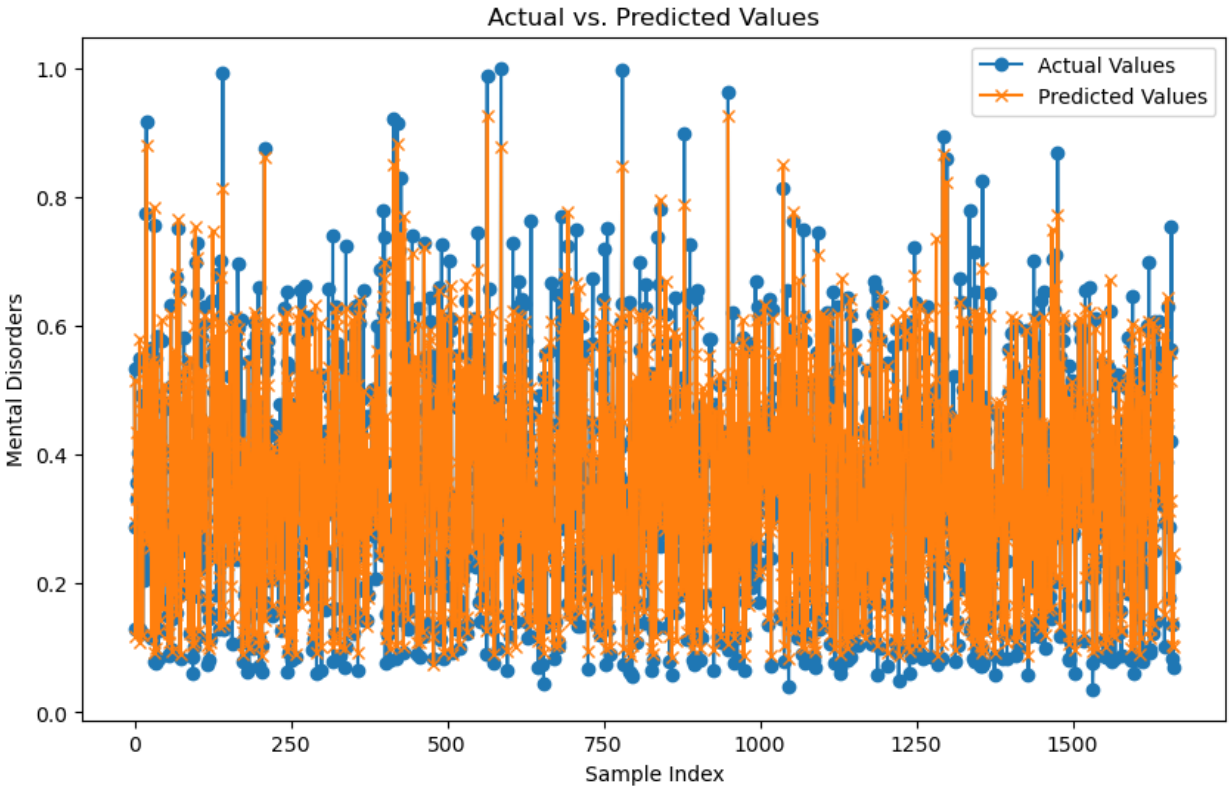
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0040 - mae: 0.0452 - val_loss: 0.0042 - val_mae: 0.0431 - learning_rate: 0.0010
Epoch 79/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0047 - mae: 0.0484 - val_loss: 0.0039 - val_mae: 0.0414 - learning_rate: 0.0010
Epoch 80/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0044 - mae: 0.0468 - val_loss: 0.0049 - val_mae: 0.0470 - learning_rate: 0.0010
Epoch 81/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0042 - mae: 0.0455 - val_loss: 0.0042 - val_mae: 0.0449 - learning_rate: 0.0010
Epoch 82/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0042 - mae: 0.0456 - val_loss: 0.0037 - val_mae: 0.0423 - learning_rate: 0.0010
Epoch 83/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0039 - mae: 0.0442 - val_loss: 0.0044 - val_mae: 0.0458 - learning_rate: 0.0010
Epoch 84/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0041 - mae: 0.0445 - val_loss: 0.0040 - val_mae: 0.0427 - learning_rate: 0.0010
Epoch 85/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0038 - mae: 0.0433 - val_loss: 0.0043 - val_mae: 0.0434 - learning_rate: 0.0010
Epoch 86/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0041 - mae: 0.0446 - val_loss: 0.0044 - val_mae: 0.0463 - learning_rate: 0.0010
Epoch 87/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0039 - mae: 0.0437 - val_loss: 0.0040 - val_mae: 0.0423 - learning_rate: 0.0010
Epoch 88/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0037 - mae: 0.0426 - val_loss: 0.0039 - val_mae: 0.0430 - learning_rate: 0.0010
Epoch 89/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0037 - mae: 0.0429 - val_loss: 0.0041 - val_mae: 0.0428 - learning_rate: 0.0010
Epoch 90/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0035 - mae: 0.0425 - val_loss: 0.0037 - val_mae: 0.0418 - learning_rate: 0.0010
Epoch 91/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0034 - mae: 0.0417 - val_loss: 0.0037 - val_mae: 0.0407 - learning_rate: 0.0010
Epoch 92/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0035 - mae: 0.0410 - val_loss: 0.0037 - val_mae: 0.0404 - learning_rate: 0.0010
Epoch 93/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0033 - mae: 0.0403 - val_loss: 0.0037 - val_mae: 0.0413 - learning_rate: 0.0010
Epoch 94/100
122/122 ━━━━━━━━━━━━━━━━━━━━ **2s** 19ms/step - loss: 0.0035 - mae: 0.0413 - val_loss: 0.0037 - val_mae: 0.0396 - learning_rate: 0.0010
Epoch 95/100

```
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0034 - mae:
0.0405 - val_loss: 0.0034 - val_mae: 0.0401 - learning_rate: 0.0010
Epoch 96/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0035 - mae:
0.0411 - val_loss: 0.0031 - val_mae: 0.0387 - learning_rate: 0.0010
Epoch 97/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0031 - mae:
0.0389 - val_loss: 0.0037 - val_mae: 0.0394 - learning_rate: 0.0010
Epoch 98/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0034 - mae:
0.0406 - val_loss: 0.0031 - val_mae: 0.0378 - learning_rate: 0.0010
Epoch 99/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0031 - mae:
0.0399 - val_loss: 0.0034 - val_mae: 0.0390 - learning_rate: 0.0010
Epoch 100/100
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0030 - mae:
0.0384 - val_loss: 0.0031 - val_mae: 0.0378 - learning_rate: 0.0010
```






```
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
```




```
residuals = y_test - y_pred.flatten()
# Plot histogram of residuals
plt.figure(figsize=(10, 6))
plt.hist(residuals, bins=30, color='purple', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.show()
```

```
r2 = r2_score(y_test, y_pred)
print(f'R-squared: {r2}')
```

R-squared: 0.9043033169059241