**Utilising Economic Development and Population Size to Predict Plastics mismanagement.**

Data:

1. https://ourworldindata.org/plastic-pollution
2. https://data.worldbank.org/indicator/NY.GDP.MKTP.CD?view=chart

Contents

[**Utilising Economic Development and Population Size to Predict Plastics mismanagement.** 2](#_Toc150377618)

[**1.** **Introduction** 2](#_Toc150377619)

[**2.** **Models** 3](#_Toc150377620)

[**2.1 Multiple linear regression analysis** 3](#_Toc150377621)

[**2.1.1 Problem statement** 3](#_Toc150377622)

[**2.2 Machine learning algorithms** 4](#_Toc150377623)

[**2.3 Performance metrics** 12](#_Toc150377624)

[**2.4 The dataset** 13](#_Toc150377625)

[**3.** **Data Pre-processing and Results of Data Exploration** 13](#_Toc150377626)

[**4.** **Baselinegression results** 19](#_Toc150377627)

[**5.** **Visualization** 29](#_Toc150377628)

[**5. Conclusion** 29](#_Toc150377629)

[**References/Bibliography** 30](#_Toc150377630)

[**Code/Appendix:** 31](#_Toc150377631)

# **Utilising Economic Development and Population Size to Predict Plastics mismanagement.**

**Summary**

Plastic pollution has become a global menace. Predicting plastic mismanagement accurately will aid assist in decision-making and formulating policies to mitigate this growing threat. I utilise economic development and population to predict plastic mismanagement in this report. My results show that Gradient Boosting Machines with Hyperparameter Turning had relatively higher performance and predictive accuracy when compared to other models used in the report.

## **Introduction**

In the modern era, the proliferation of plastics has become synonymous with industrial growth and economic prosperity (World Economic Forum, 2016). Plastic is so useful to almost all industries because of it is high durability, versatility, lightweight and low-cost, resulting in significant increase in its production and usage over the years (Hopewell et al., 2009). This growth brings with it a significant environmental challenge – inappropriately disposed plastic (also termed plastic mismanagement or waste pollution). As economic development increases (measured by GDP; e.g., see Konstantinidou and Scherer, 2022), so does the massive increase in environmental pollution (Stern, 2017) and most especially, plastic waste (Jambeck et al., 2015). As highlighted by Tsiamis et al. (2018), in the United States for example, while generated solid waste between 1960 and 2013 increased by 188%, plastic pollution grew by 8,238%. In a related study, Pankaj (2015) show that population size is associated with increase in plastic waste generation. Supporting this study, Jambeck et al. (2015) note that plastic waste (including mismanagement) will continue to increase with growth in population.

Plastic mismanagement has significant negative impact on climate change (Cabernard et al., 2022). The growing problem of plastic mismanagement has adversely influenced biodiversity, animal welfare, food security, marine life, loss of income and human health (Kershaw, 2016). Plastic has become now one of the most common contaminants on the earth today and poses serious ecological and financial issues (Derraik, 2002). At present, plastic mismanagement is an issue that threatens the United Nations Sustainable Development Goals (SDGs), inclusive of climate change targets (Walter, 2021).

Existing approaches primarily focusing on either the economic benefits of plastic production or its environmental repercussions in isolation (e.g., Hawkins et al., 2013) and e-waste pollution in general (e.g., Boubellouta & Kusch-Brandt, 2021); however, little is known about how economic development and population size predicts plastic mismanagement. Accurate forecasting will help countries promote more focused programmes meant for waste reduction and advance pro-environmental behaviour whilst also raising awareness of environmental issues. Thus, this report explores the utilisation of economic development and population size to predict plastic waste mismanagement.

## **Models**

### **2.1 Multiple linear regression analysis**

In this report, I employ multiple regressions analysis, an extension of linear regression to predict the plastic mismanagement by utilising economic development and population size data. Although, linear regression could be very beneficial, it comes with its set of challenges, primarily if unbiased variables are enormously correlated. This can distort interpretations (O'Brien, 2007). Additionally, the presence of outliers or non-linear relationships may necessitate extra complex modelling strategies (Hastie et al., 2009). This only emphasizes the need for rigorous data pre-processing and model validation (Dietz & Stern, 1995).

### **2.1.1 Problem statement**

The regression line I used for the evaluating the relationship, which, in its most basic form, is to regress function *f,* such that:

(1)

Where represent the explanatory variables (GDP, population size) of the dataset while is the dependent variable (plastic mismanagement) used for the prediction. GDP is used as a measure for economic development (e.g., see Konstantinidou & Scherer, 2022).

### **2.2 Machine learning algorithms**

For the report, I compare the baseline model with some Machine Learning (ML) techniques used in prediction. Machine Learning methodologies can be broadly classified as: supervised, unsupervised, and reinforcement learning (Bishop, 2006). For this report, my work consists mostly of supervised learning because it is task-driven and commonly used for forecasting. Supervised learning is the most common approach and involves training models on labelled data, enabling them to make predictions or classifications based on input features (Hart et al., 2001). Unsupervised learning - on the other hand - deals with uncovering hidden structures or patterns within datasets while reinforcement learning is a dynamic approach where agents learn by interacting with an environment; they receive feedback in the form of rewards or penalties (Hastie et al., 2009; Sutton and Barto, 2018).

* + 1. **Supervised machine learning algorithms**

In this section, I describe the supervised predictive modelling and algorithms techniques used in the report.



Figure 1 below shows the code snippet. In addition, the steps involved for improvement using polynomial regression -Polynomial features allow linear regression to model non-linear relationships by adding higher-degree terms of the input features- are:

1. Set the degree of the polynomial (degree = 2).
2. Create a pipeline that includes PolynomialFeatures (degree) and LinearRegression (polyreg).
3. Fit the model (polyreg) on the training data (X\_train, y\_train).
4. Make predictions on both the training and test sets (y\_pred\_train\_poly and y\_pred\_test\_poly) using the polynomial regression model.
5. Evaluate the model.

While Figure 1 below shows the code snippets for the linear regression, Figure 2 shows the snippet for improvement using polynomial regression

A screenshot of a computer program

Description automatically generated

Fig. 1. Linear regression code snippet

A screenshot of a computer program

Description automatically generated

Fig. 2. Linear regression with polynomial features code snippet

1. Gradient Boosting Machine (GBM): GBM is mostly employed to solving regression problems and big data mining challenges. Key advantages are efficiency, less data pre-processing, robustness to less clean data, flexibility, predictive performance, and interpretability (Zhang and Jung, 2020). Also, If the relationship between the features and the target variable is non-linear, Gradient Boosting can capture this complexity through the sequential building of trees and most importantly It offers insights into feature importance, which can be valuable for understanding which factors contribute most to plastic mismanagement. The key steps I followed are I initialise the regressor, train the model, obtain the importance ratings for every attribute, make prediction, and evaluate the model. See figure 3. Next, I optimised the model via hyperparameter tuning (In the param\_grid for the GradientBoostingRegressor, n\_estimators defines the number of sequential trees to build (more trees can capture complex patterns but may overfit), learning\_rate determines the contribution of each tree (lower rates require more trees but can generalize better), and max\_depth sets the depth of each tree (deeper trees can model complex relationships at the risk of overfitting)) using GridSearchCV to get the best variables as follows: (i) Define hyperparameters and values, (ii) Initialize, (iii) Fit the model, searching for the best hyperparameters, and (iv) Extract the best hyperparameters and associated R-squared score. See Figure 4.

A computer screen shot of a program

Description automatically generatedFig. 3. GBM code snippet

A computer screen shot of a program code

Description automatically generated

Fig. 4: Code Snippet for GBM with hyperparameter tuning.

1. Support Vector Regression (SVR): SVR is a non-linear ML model based on statistical learning theory known for enhancing prediction accuracy (Gunn, 1998). However, it is less responsive to minor estimation errors and outliers due to its loss function (Vapnik, 1995). SVR can model non-linear relationships using kernel functions, which can be particularly useful if there are complex, non-linear interactions between features. The steps I followed are initialisation of the regressor, training the model, obtaining importance ratings for every attribute, making prediction and evaluate the model. See figure 5. In addition, I optimised with Polynomial Feature engineering. See figure 6.

A computer screen shot of text

Description automatically generated

Fig. 5: SVR code snippet

A screenshot of a computer program

Description automatically generatedFig. 6: SVR optimisation using polynomial feature engineering code snippet.

1. Neural Networks for Regression NNR): Neural Networks can model highly complex and non-linear relationships, which might be present in the data, with a considerable amount of data (which we have), a Neural Network can learn deep representations of the data, potentially leading to more accurate predictions. Neural networks also help to prevent incorrect prediction caused by noise, lack of variables and data insufficiency (Kang and kang, 2023). However, one of its limitations is that it requires considerable computation and large memory usage to make prediction (Leroux et al., 2020). The steps I followed are: Normalise the regressor, define the model, compile the model, train, evaluate, optimise (it includes manually adjusting hyperparameters to evaluate performance like learning rate, epochs, and activation functions -In neural network tuning, the learning rate controls how much the model weights are updated during training, with a lower rate potentially leading to better generalization but slower convergence, while a higher rate speeds up training but risks overshooting the minimum loss. Epochs define the number of times the entire training dataset is passed forward and backward through the neural network, with too few causing underfitting and too many possibly leading to overfitting. Activation functions determine the output of neurons, influencing the network's ability to capture non-linear patterns, with common choices like ReLU offering fast computation and sigmoid or tanh handling binary classification effectively-) and regularise one model with dropout layers and L1/L2 regularization. See figure 7a, 7b, 7c and 7d.

A screen shot of a computer code

Description automatically generated

Fig. 7a: Normalising the features for NNR code snippet.

A computer screen with text on it

Description automatically generated

Fig. 7b: Defining NNR code snippet.

A screen shot of a computer program

Description automatically generated

Fig. 7c: Predicting NNR code snippet.

A screen shot of a computer program

Description automatically generated

Fig. 7d: NNR optimisation

### **2.3 Performance metrics**

I employ four performance metrics to evaluate the accuracy of various models used it predicting plastic mismanagement. The four-evaluation metrics are: Mean Absolute Error (MAE) [Eq. 2], Mean Squared Error (MSE) [Eq. 3], Root Mean Square Error (RMSE) [Eq. 4] and Coefficient of Determination () [Eq. 5] (Thongthammachart et al., 2022). The MAE, MSE and RMSE reflect the error magnitude of the models so that lower values indicate higher accuracy of the prediction results. Besides, ranges from 0 to 1 and a higher value indicates a higher precision of a model to accurately predict values. The use of MAE, MSE, and RMSE provides a rounded approach to understanding the accuracy of predictions. Different error metrics can give insights into the type of errors a model is making. For example, RMSE is more sensitive to outliers than MAE.

The formulas of the four metrics are as follows:

(2)

(3)

(4)

(5)

where, the total number of samples in the dataset, represents the actual values, denotes the predicted values and represents the average of predicted values. is the absolute difference between actual and predicted values.

### **2.4 The dataset**

To carry out the prediction, I collected plastics mismanagement data from Our World in Data database[[1]](#footnote-1). The sample consist of 168 countries from year 2000 to 2019. The dataset has incomplete record for some countries. Since the data has yearly total of plastics mismanagement, and I used extrapolation to complete the missing data. I merge the information with data collected from the World Bank development indicators database (GDP and population size)[[2]](#footnote-2). The sample give a total of 3360 observations and 3 variables across 168 countries.

## **Data Pre-processing and Results of Data Exploration**

Loading Data from Excel Sheets:

I started by loading GDP, population, and plastic pollution data from different sheets of an Excel file. Each dataset was loaded into separate Pandas dataframes (gdp\_df, population\_df, plastic\_df). This was necessary because the data was segmented across multiple sheets. (See Fig 0a). Next, I inspected each dataframe using the info() and head() methods. This step was crucial for me to get an initial understanding of the data, including the format, column names, data types, and a peek at the actual data. I then created a function named clean\_and\_melt to clean and reshape the dataframes. In this function, I set new column names based on the third row, dropped unnecessary rows, reset the index, and melted the dataframes to transform them into a long format. This helped in standardizing the data structure for analysis. See Fig 0b. I applied the clean\_and\_melt function to each dataframe, resulting in gdp\_melted, population\_melted, and plastic\_melted. This step ensured that all datasets were in a consistent format.

I then merged these cleaned dataframes based on common columns. This merging was crucial for combining different data sources into a single dataset for a comprehensive analysis. See Fig 0c.

After merging, I removed duplicate or unnecessary columns and sorted the data by 'Country' and 'Year'. I also cleaned the 'Year' column by dropping NaN values and converting it to an integer type for consistency. See Fig 0d.

I conducted basic data exploration using the describe() method to understand the statistics of the dataset. I then focused on specific columns like 'Mismanaged Plastic', 'GDP', and 'Population', calculating their median values and summarizing their statistics.

Addressing missing values was a key part of my process. I calculated the number of missing values, then imputed or dropped them based on the dataset's needs. I also rechecked the dataset to ensure no missing values remained. See Fig 0e.

The discovery of outliers in the data, as indicated by high standard deviations and medians, led me to consider potential anomalies in data collection or unique characteristics of certain data points. To manage these outliers and reduce skewness, I applied a log transformation to the data, making it more amenable to linear modelling. See Fig 0f.

Finally, I analysed the correlation heatmap to understand the relationships between different variables, particularly focusing on how GDP and population size might influence plastic mismanagement. See Fig. 9.

In essence, my data preprocessing journey involved a series of meticulous steps to ensure that the data was accurately loaded, cleaned, merged, and transformed, setting a strong foundation for the subsequent analysis. See Fig 0

A screen shot of a computer code

Description automatically generated

Fig 0a: Loading Data from Excel Sheets.

A computer screen with colorful text

Description automatically generated

Fig 0b: Cleaning and Reshaping Data with a Custom Function.

A computer screen shot of text

Description automatically generated

Fig 0c: Merging the Dataframes.

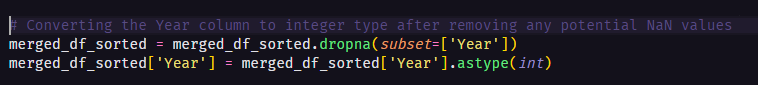


Fig 0d: Further Cleaning and Organizing.

A screenshot of a computer program

Description automatically generated

Fig 0e: Handling Missing Values.

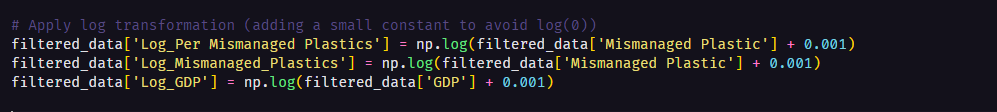


Fig 0f: Log Transformation.

After cleaning the data and using python to extract the necessary data, I initiate the data exploration. I loaded the dataset into Python and utilized the describe method to gain a comprehensive understanding of its basic statistics. Table 1 shows the summary statistics of the data employed. The dataset contained multiple columns, but most notably, it provided information on GDP, Population Size and Mismanaged Plastics. The mean population of the sample is 38.2 million (standard deviation = 144.2 million), whilst the mean GDP is $372.2 billion (standard deviation = $1,488.6 billion). The standard deviation and median of the data are very high, suggesting outliers. Figure 1 shows the distribution of each variable. The presence of these outliers raised questions regarding data collection or the unique nature of certain entities (See Figure 8a below). I decided to retain the outliers for a comprehensive analysis, but their impact was mitigated by applying a log transformation of the data. This transformation not only reduced the outliers and skewness in the dataset but also made it more suitable for linear modelling (See Figure 8b). Figure 9 shows the correlation heatmap. The outcome revealed a strong positive correlation, with the log-GDP and log-population explaining some of the variance in plastics mismanagement. However, this also underscored a critical realization: while log-GDP and log-population are significant determinants, a portion of the variability in plastic mismanagement is influenced by other factors not present in the dataset.



A graph with a red line

Description automatically generatedFig 8a: Distribution of GDP, Mismanaged Plastics and Population Size before transformation

A graph of a graph

Description automatically generatedFig 8b. Distribution of GDP, Mismanaged Plastics and Population Size after transformation

A red squares with white numbers

Description automatically generated

Fig. 9. Correlation heatmap

## **Baseline regression results**

Table 2 presents the baseline regression results based on two explanatory variables. To elucidate the multiple regression method, I first look at the coefficients (β) of the explanatory variables and their statistical significance (t-statistics or p-value). The validation results for the performance of linear regression modelling are displayed in Figure 10. Overall, the results corroborate a significantly positive relationship between log-GDP (β=0.2386, t=31.96), Log-Population (β=0.7025, t=95.70) and plastics mismanagement. In addition, the results in Table 3 (part a) show that the R^2 is 0.922, suggesting that the linear regression model conducted in this study can effectively predict plastics mismanagement with high reliability. 

A graph showing a line of plastic

Description automatically generated with medium confidence

Fig. 10 Log-Mismanaged plastics(t)=f(log-Population, log-GDP)



### **4.1 Predicting plastics mismanagement accuracy using ML models.**

Given the positive statistically significant relation between log-GDP, Log-Population and plastic mismanagement, in this section, I take further step to discuss the predictive accuracy of various estimation models using ML techniques (see Section 2 above). I compare with the baseline regression model and select the best model for prediction. Part B of Table 3 above shows the results of the predictive accuracy of the linear model and each comparison model.

To begin I'm selecting 'Log\_GDP' and 'Log\_Population' as my model's features and 'Log\_Per Mismanaged Plastics' as the target, selecting 'Log\_GDP' and 'Log\_Population' as features is a strategic choice because they are likely to have a significant relationship with the target variable 'Log\_Per Mismanaged Plastics'. Good feature selection is key to improving model performance as it ensures that the predictors have a strong explanatory power. Then I split my dataset into 80% for training and 20% for testing to validate my model's performance. The choice to split the dataset into 80% for training and 20% for testing is a standard approach in machine learning that helps in validating the model's performance on unseen data, which is essential for understanding its generalizability.

In the linear regression model, is 0.933, indicating that the model provides a high and fitting accuracy in predicting plastics mismanagement. In addition, the MAE, MSE and RMSE values are 0.464, 0.420 and 0.647, respectively. The results demonstrate that the linear model can predict plastic waste. For linear regression with polynomial features, is 0.940, while the MAE(0.449) MSE(0.386) and RMSE(0.621) values are lower when compared with linear regression model, implying that linear regression results can be further improved with polynomial features. The improvement in performance with the inclusion of polynomial features in the linear regression model suggests that the relationship between the predictors and the target variable is not purely linear. Polynomial features can model the curvature in data, providing a better fit.

Examining the numerical results of all the other ML models used in this report (See Table 3), for example, GBM prediction accuracy () without the process of hyperparameters tuning yield 0.955. The model forecast accuracy is further enhanced with the process of hyperparameters tuning that yields relatively higher predictive accuracy (=0.964) among all model candidates. In term of using MAE, MSE and RMSE to validate the forecast accuracy, the results in columns (1) – (3) of Table 3 show that GBM with hyperparameter tuning values yield the lowest error among all model candidates, demonstrating that the ML outperforms other models in terms of predicting plastic mismanagement. The significant increase in performance of GBM after hyperparameter tuning indicates that optimizing the model parameters can lead to a substantial improvement in the model's ability to predict accurately. Hyperparameter tuning helps in customizing the model to the specifics of the data.

My results are in agreement with those reported by Chen et al. (2022). Overall, I show that Gradient Boosting Machines with Hyperparameter Turning is the best model to use predict plastic mismanagement using log-GDP and log-Population in this report.

The evaluation results of Linear Regression, Linear Regression with Polynomial Features, Gradient Boosting Machines (GBM), Gradient Boosting Machines (GBM) With Hyperparameter Tuning, Support Vector Regression (SVR), SVR optimised with polynomial feature engineering, Neural Networks Model, Neural Networks for Regression (Optimised) and Optimal Neural Network (Regularization) ML algorithms for predicting plastics mismanagement are presented below in Figures 11 to 19, respectively. Figure 20 shows the geographical distribution visualization of model’s residuals.

A graph showing a blue and red line

Description automatically generated

**Fig. 11a: Evaluation results of the linear regression ML**

A graph of different colored lines

Description automatically generated with medium confidence

**Fig. 11b: Evaluation results of the linear regression ML scatter plot and residuals**

A graph with a line graph

Description automatically generated with medium confidence

**Fig. 12a: Evaluation results of the linear regression with polynomial features**

A graph of data and a graph of data

Description automatically generated

**Fig. 12b: Evaluation results of the linear regression ML scatter plot and residuals**

A graph of red dots

Description automatically generated

**Fig. 13: Evaluation results using GBM**

A diagram with red dots

Description automatically generated

**Fig. 14 Evaluation results using GBM with Hyperparameter Tuning**

A diagram of a plot

Description automatically generated

**Fig. 15 Evaluation results using SVR**

A graph with red dots

Description automatically generated

**Fig. 16 Evaluation results using SVR with Polynomial Feature Engineering**

A diagram of a graph

Description automatically generated with medium confidence

**Fig. 17 Evaluation results using Neural Networks for Model**

A comparison of graphs with different colors

Description automatically generated with medium confidence

**Fig. 18 Evaluation results using Neural Networks Regression (Optimised)**

A graph of different values

Description automatically generated with medium confidence

**Fig. 19 Evaluation results using Neural Networks Regression (Regularisation)**

For Geographical distribution of model residuals, I am calculating residuals, merging them with a world map, and then plotting the residuals geographically to analyse prediction accuracy. See Fig. 20.

A map of the world

Description automatically generated

**Fig. 20. Geographical distribution visualization**

### **4.2 Results and Discussion**

In evaluating the performance of the models used in my study, "Utilising Economic Development and Population Size to Predict Plastics Mismanagement," I realized that several factors contributed to the varied success rates. The Gradient Boosting Machine (GBM), for instance, outperformed others largely due to its ability to handle complex, non-linear relationships in the data, a feature crucial in environmental modelling. Its performance was further enhanced by meticulous hyperparameter tuning, which optimized the model's learning process. I also noticed that the models' accuracy varied with the quality of data preprocessing; models performed better with well-curated data, highlighting the importance of thorough data cleaning and transformation. Additionally, the incorporation of cross-validation methods provided insights into the models' robustness, ensuring that the performance wasn't just a result of overfitting to a particular dataset. Through these evaluations, I gained a deeper understanding of how the choice of algorithm, data quality, and validation techniques collectively influence the predictive power and reliability of the models in this environmental context.

## **Visualization**

In my visualization, I've decided to use three plots to thoroughly evaluate the performance of my regression model.

The first plot, "True vs. Predicted Values," is where I assess how well my model's predictions match the actual values. Each point represents an observation from my dataset. By plotting the true values against the predicted ones, I aim for the points to align closely with the diagonal line. This alignment would indicate a high accuracy of my model's predictions.

Moving to the second plot, the "Residuals Plot," I investigate the residuals, which are the differences between the true values and my model's predictions. I plot these residuals against the predicted values to check for randomness. My goal here is to see no discernible pattern; a random dispersion of points suggests that my model is making errors evenly across all levels of prediction.

The third plot is the "Residuals Distribution." Here, I'm looking at how the residuals are distributed in terms of frequency. Ideally, I'm aiming for a normal distribution, represented by the bell curve. This normality would support the assumption that my model's errors are also normally distributed, which is a fundamental consideration for many statistical modelling techniques.

I use these three plots collectively to diagnose any issues with my model, like whether there are any patterns in the residuals that I need to address, or if there are outliers affecting my predictions. These visualizations are crucial for me to ensure that my model is robust and reliable before I proceed with using it for further analysis or making decisions based on its predictions.

## **. Conclusion**

It is a known fact that plastic pollution has become a global problem. Predicting plastic mismanagement accurately will assist in decision-making and formulating policies to mitigate this growing threat. In this report, I utilise economic development and population to predict plastic mismanagement. I show that both explanatory variables have high reliability in predicting plastic pollution. In addition, I explored the predictive accuracy using various MLs algorithms and I show that Gradient Boosting Machines with Hyperparameter Turning had relatively higher performance and predictive accuracy of compared to other models. In sum, while each algorithm has its strengths and weaknesses, the choice of model should be guided by the problem at hand, the nature of the data, and available resources.

# **References/Bibliography**

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4(4)). New York: springer.

Boubellouta, B., & Kusch-Brandt, S. (2021). Relationship between economic growth and mismanaged e-waste: Panel data evidence from 27 EU countries analyzed under the Kuznets curve hypothesis. *Waste Management*, 120, 85-97.

Cabernard, L., Pfister, S., Oberschelp, C., & Hellweg, S. (2022). Growing environmental footprint of plastics driven by coal combustion. *Nature Sustainability*, 5(2), 139-148.

Chen, H., Shen, Z., Wang, L., Qi, C., & Tian, Y. (2022). Prediction of undrained failure envelopes of skirted circular foundations using gradient boosting machine algorithm. *Ocean Engineering*, *258*, 111767.

Derraik, J. G. (2002). The pollution of the marine environment by plastic debris: a review. *Marine pollution bulletin*, *44*(9), 842-852.

Dietz, T., & Stern, P. C. (2002). Exploring new tools for environmental protection. *New tools for environmental protection: education, information, and voluntary measures*, Vol. 3(8).

‌Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*. <https://arxiv.org/abs/1702.08608>

Gunn, S. R. (1998). Support vector machines for classification and regression. *ISIS technical report*, 14(1), 5-16.

Hart, P. E., Stork, D. G., & Duda, R. O. (2001). *Pattern classification*. Wiley. <https://www.researchgate.net/publication/228058014_Pattern_Classification>

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (2nd ed.). Springer.

Hawkins, T. R., Singh, B., Majeau‐Bettez, G., & Strømman, A. H. (2013). Comparative environmental life cycle assessment of conventional and electric vehicles. *Journal of industrial ecology*, 17(1), 53-64.

Hopewell, J., Dvorak, R., & Kosior, E. (2009). Plastics recycling: challenges and opportunities. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364, 2115-2126.

Jambeck, J. R., Geyer, R., Wilcox, C., Siegler, T. R., Perryman, M., Andrady, A., ... & Law, K. L. (2015). Plastic waste inputs from land into the ocean. *Science*, *347*(6223), 768-771.

Kang, M., & Kang, S. (2023). Surrogate approach to uncertainty quantification of neural networks for regression. *Applied Soft Computing*, 139, 110234.

Kershaw, P. J. (2016). Marine plastic debris and microplastics–Global lessons and research to inspire action and guide policy change.

Leroux, S., Vankeirsbilck, B., Verbelen, T., Simoens, P., & Dhoedt, B. (2020). Training binary neural networks with knowledge transfer. *Neurocomputing*, 396, 534-541.

‌O’Brien, R.M. (2007). A Caution Regarding Rules of Thumb for Variance Inflation Factors. *Quality & Quantity*, 41(5), 673–690.

Pankaj, V. P. (2015). Sustainable model of Plastic waste management. *International Journal of ChemTech Research*, *7*(1), 440-458.

Ritchie, H., Samborska, V., Roser, M. (2023). Plastic Pollution. Our world in Data. <https://ourworldindata.org/plastic-pollution>

Rock, N. M. S. (1987). ROBUST: An interactive FORTRAN-77 package for exploratory data analysis using parametric, robust and nonparametric location and scale estimates, data transformations, normality tests, and outlier assessment. *Computers & Geosciences*, *13*(5), 463-494.

Roe, B. P., Yang, H. J., Zhu, J., Liu, Y., Stancu, I., & McGregor, G. (2005). Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2-3), 577-584.

Sutton, R.S., & Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. MIT Press. <https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>

Thongthammachart, T., Araki, S., Shimadera, H., Matsuo, T., & Kondo, A. (2022). Incorporating Light Gradient Boosting Machine to land use regression model for estimating NO2 and PM2. 5 levels in Kansai region, Japan. *Environmental Modelling and Software*, *155*, 105447.

Tsiamis, K., Zenetos, A., Deriu, I., Gervasini, E., & Cardoso, A. C. (2018). The native distribution range of the European marine non-indigenous species. *Aquatic Invasions*, *13*(2).

World Bank (2023). World Development Indicators. <https://data.worldbank.org/indicator/NY.GDP.MKTP.CD?view=chart>

Zhang, Z., & Jung, C. (2020). GBDT-MO: Gradient-boosted decision trees for multiple outputs. *IEEE transactions on neural networks and learning systems*, *32*(7), 3156-3167.

# **Code/Appendix:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817  818  819  820  821  822  823  824  825  826  827  828  829  830  831  832  833  834  835  836  837  838  839  840  841  842  843  844  845  846  847  848  849  850  851  852  853  854  855  856  857  858  859  860  861  862  863  864  865  866  867  868  869  870  871  872  873  874  875  876  877  878  879  880  881  882  883  884  885  886  887  888  889  890  891  892  893  894  895  896  897  898  899  900  901  902  903  904  905  906  907  908  909  910  911  912  913  914  915  916  917  918  919  920  921  922  923  924  925  926  927  928  929  930  931  932  933  934  935  936  937  938  939  940  941  942  943  944  945  946  947  948  949  950  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968  969  970  971  972  973  974  975  976  977  978  979  980  981  982  983  984  985  986  987  988  989  990  991  992  993  994  995  996  997  998  999  1000  1001  1002  1003  1004  1005  1006  1007  1008  1009  1010  1011  1012  1013  1014  1015  1016  1017  1018  1019  1020  1021  1022  1023  1024  1025  1026  1027  1028  1029  1030  1031  1032  1033  1034  1035  1036  1037  1038  1039  1040  1041  1042  1043  1044  1045  1046  1047  1048  1049  1050  1051  1052  1053  1054  1055  1056  1057  1058  1059  1060  1061  1062  1063  1064  1065  1066  1067  1068  1069  1070  1071  1072  1073  1074  1075  1076  1077  1078  1079  1080  1081  1082 | **import** **pandas** **as** **pd**  **import** **numpy** **as** **np**  **import** **geopandas** **as** **gpd**  **import** **statsmodels.api** **as** **sm**  **from** **sklearn.model\_selection** **import** train\_test\_split, GridSearchCV, cross\_val\_score, cross\_validate  **from** **sklearn.linear\_model** **import** LinearRegression  **from** **sklearn.tree** **import** DecisionTreeRegressor  **from** **sklearn.ensemble** **import** GradientBoostingRegressor  **from** **sklearn.svm** **import** SVR  **from** **sklearn.neural\_network** **import** MLPRegressor  **from** **sklearn.preprocessing** **import** StandardScaler, PolynomialFeatures  **import** **matplotlib.pyplot** **as** **plt**  **import** **seaborn** **as** **sns**  **from** **matplotlib** **import** cm  **from** **mpl\_toolkits.mplot3d** **import** Axes3D  **from** **sklearn.tree** **import** plot\_tree  **from** **sklearn.metrics** **import** mean\_absolute\_error, mean\_squared\_error, r2\_score, make\_scorer  **import** **tensorflow** **as** **tf**  **from** **tensorflow.keras.models** **import** Sequential  **from** **tensorflow.keras.layers** **import** Dense, Dropout  **from** **tensorflow.keras.activations** **import** relu, tanh, sigmoid  **from** **tensorflow.keras.regularizers** **import** l1\_l2  **from** **sklearn.pipeline** **import** Pipeline, make\_pipeline  %matplotlib inline  **import** **warnings**  warnings.filterwarnings('ignore')  # Set plot style  sns.set\_style('whitegrid')  # Load the data from the provided Excel file  file\_path = "C:**\\**CoventryMl**\\**11215866-NDA-s1**\\**Coursework\_1**\\**datasets**\\**Plastic Pollution\_population\_data\_20.10.2023.xlsx"  # Loading GDP and Population data from the other sheets in the Excel file  gdp\_df = pd.read\_excel(file\_path, sheet\_name=**1**)  population\_df = pd.read\_excel(file\_path, sheet\_name=**2**)  plastic\_df = pd.read\_excel(file\_path, sheet\_name=**0**)  # Displaying basic info and the first few rows of the GDP, Population, and Plastic dataframes  gdp\_df.info(), gdp\_df.head(), population\_df.info(), population\_df.head(), plastic\_df.info(), plastic\_df.head()  # Function to clean and melt the dataframes  # Modifying the function to handle different column names for country and year columns  **def** **clean\_and\_melt**(df, value\_var, country\_col, code\_col, drop\_rows=**3**):  # Setting new column names based on the content of the third row  df.columns = df.iloc[**2**]  df = df.drop([**0**, **1**, **2**])  df.reset\_index(drop=True, inplace=True)    # Melting the dataframe  df\_melted = df.melt(id\_vars=['ss', country\_col, code\_col], var\_name='Year', value\_name=value\_var)  **return** df\_melted  # Applying the modified function to each dataframe  gdp\_melted = clean\_and\_melt(gdp\_df, 'GDP', 'Country Namedd', 'Country Code')  population\_melted = clean\_and\_melt(population\_df, 'Population', 'Country Namedd', 'Country Code')  # Applying the modified function to the plastic pollution dataframe with corrected column names  plastic\_melted = clean\_and\_melt(plastic\_df, 'Mismanaged Plastic', 'Country', 'Code')  # Displaying the first few rows of each melted dataframe  gdp\_melted.head(), population\_melted.head(), plastic\_melted.head()  # Merging the three melted dataframes based on common columns  merged\_df = plastic\_melted.merge(gdp\_melted, left\_on=['ss', 'Country', 'Code', 'Year'],  right\_on=['ss', 'Country Namedd', 'Country Code', 'Year'])  merged\_df = merged\_df.merge(population\_melted, left\_on=['ss', 'Country', 'Code', 'Year'],  right\_on=['ss', 'Country Namedd', 'Country Code', 'Year'])  # Dropping duplicated or unnecessary columns  columns\_to\_drop = ['Country Namedd\_x', 'Country Code\_x', 'Country Namedd\_y', 'Country Code\_y']  merged\_df = merged\_df.drop(columns=columns\_to\_drop)  # Sorting the dataframe by Country and Year  merged\_df\_sorted = merged\_df.sort\_values(by=['Country', 'Year']).reset\_index(drop=True)  # Displaying the first few rows of the merged dataframe  merged\_df\_sorted.head()  # Converting the Year column to integer type after removing any potential NaN values  merged\_df\_sorted = merged\_df\_sorted.dropna(subset=['Year'])  merged\_df\_sorted['Year'] = merged\_df\_sorted['Year'].astype(int)  # Displaying the first few rows of the updated dataframe  merged\_df\_sorted.head(**30**)  # Saving the sorted and merged data to a CSV file  file\_path\_csv = 'C:\CoventryMl**\\**11215866-NDA-s1\Coursework\_1\datasetsCleaned\_Plastic\_Pollution\_Data.csv'  merged\_df\_sorted.to\_csv(file\_path\_csv, index=False)  file\_path\_csv  # Load the first dataset  df\_per\_capita\_vs\_gdp = pd.read\_csv('C:**\\**CoventryMl**\\**11215866-NDA-s1**\\**Coursework\_1**\\**datasetsCleaned\_Plastic\_Pollution\_Data.csv')  # Display the first few rows of the dataset  df\_per\_capita\_vs\_gdp.head()  # Set the display format to be float format  pd.options.display.float\_format = '{:.0f}'.format  df\_selected\_columns = df\_per\_capita\_vs\_gdp[['Mismanaged Plastic', 'GDP', 'Population']]  # Getting a summary of the selected columns  summary\_selected\_columns = df\_selected\_columns.describe()  # Calculating the median of the selected columns  median\_selected\_columns = df\_selected\_columns.median()  summary\_selected\_columns, median\_selected\_columns  # Initialize the figure  plt.figure(figsize=(**18**, **6**))  # Plotting histograms  plt.subplot(**1**, **3**, **1**)  sns.histplot(data=df\_per\_capita\_vs\_gdp, x='Mismanaged Plastic', bins=**30**, color='skyblue', kde=True)  plt.title('Distribution of Mismanaged Plastics (tons)', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**, **3**, **2**)  sns.histplot(data=df\_per\_capita\_vs\_gdp, x='GDP', bins=**30**, color='salmon', kde=True)  plt.title('Distribution of GDP', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**, **3**, **3**)  sns.histplot(data=df\_per\_capita\_vs\_gdp, x='Population', bins=**30**, color='orange', kde=True)  plt.title('Distribution of Population', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  # Displaying the plot  plt.show()  # Calculate number of missing values and percentage of missing values for each column  missing\_values = pd.DataFrame(df\_per\_capita\_vs\_gdp.isnull().sum(), columns=['Missing Values'])  missing\_values['% of Total Values'] = (df\_per\_capita\_vs\_gdp.isnull().sum() / len(df\_per\_capita\_vs\_gdp)) \* **100**  missing\_values.sort\_values(by='% of Total Values', ascending=False)  # Filter rows  filtered\_data = df\_per\_capita\_vs\_gdp[df\_per\_capita\_vs\_gdp['Mismanaged Plastic'].notnull()]  # Calculate the number and percentage of missing values in the filtered dataset  missing\_data\_filtered = pd.DataFrame(filtered\_data.isnull().sum(), columns=['Missing Values'])  missing\_data\_filtered['% of Total Values'] = (filtered\_data.isnull().sum() / len(filtered\_data)) \* **100**  missing\_data\_filtered.sort\_values(by='% of Total Values', ascending=False)  # Create a subset by dropping rows with missing GDP values  data\_subset\_no\_missing\_gdp = filtered\_data.dropna(subset=['GDP'])  # Impute missing values in the main dataset  filtered\_data['GDP'].fillna(filtered\_data['GDP'].median(), inplace=True)  # Check if there are any remaining missing values in the main dataset  missing\_data\_final = pd.DataFrame(filtered\_data.isnull().sum(), columns=['Missing Values'])  missing\_data\_final['% of Total Values'] = (filtered\_data.isnull().sum() / len(filtered\_data)) \* **100**  missing\_data\_final, data\_subset\_no\_missing\_gdp.shape  # Display data information including data types and non-null counts  data\_info = filtered\_data.info()  # Display summary statistics for numerical columns  data\_summary = filtered\_data.describe()  data\_info, data\_summary  # Apply log transformation (adding a small constant to avoid log(0))  filtered\_data['Log\_Per Mismanaged Plastics'] = np.log(filtered\_data['Mismanaged Plastic'] + **0.001**)  filtered\_data['Log\_Mismanaged\_Plastics'] = np.log(filtered\_data['Mismanaged Plastic'] + **0.001**)  filtered\_data['Log\_GDP'] = np.log(filtered\_data['GDP'] + **0.001**)  filtered\_data['Log\_Population'] = np.log(filtered\_data['Population'] + **0.001**)  # Visualize the transformed data  plt.figure(figsize=(**18**, **6**))  # Histogram for log-transformed  plt.subplot(**1**, **3**, **1**)  sns.histplot(data=filtered\_data, x='Log\_Per Mismanaged Plastics', bins=**30**, color='skyblue', kde=True)  plt.title('Distribution of Log Mismanaged Plastics (tons)', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  # Histogram for log-transformed 'GDP per capita'  plt.subplot(**1**, **3**, **2**)  sns.histplot(data=filtered\_data, x='Log\_GDP', bins=**30**, color='salmon', kde=True)  plt.title('Distribution of Log GDP',fontsize=**18**, fontname="Times New Roman", fontweight="bold")  # Histogram for log-transformed 'GDP per capita'  plt.subplot(**1**, **3**, **3**)  sns.histplot(data=filtered\_data, x='Log\_Population', bins=**30**, color='orange', kde=True)  plt.title('Distribution of Log Population',fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  # Calculate the correlation matrix for the log-transformed variables  correlation\_matrix = filtered\_data[['Log\_Mismanaged\_Plastics', 'Log\_GDP', 'Log\_Population']].corr()  # Visualize the correlation using a heatmap  plt.figure(figsize=(**8**, **6**))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', vmin=-**1**, vmax=**1**)  plt.title('Correlation Heatmap', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.show()  # Extracting the independent and dependent variables  X\_clean\_test = filtered\_data[['Log\_GDP','Log\_Population']]  y\_clean\_test = filtered\_data['Log\_Per Mismanaged Plastics']  # Adding a constant to the model (intercept)  X\_clean\_test\_with\_const = sm.add\_constant(X\_clean\_test)  # Fitting the OLS model  model = sm.OLS(y\_clean\_test, X\_clean\_test\_with\_const).fit()  # Getting a summary of the regression  model\_summary = model.summary()  **print**(model\_summary)  # Fitting the linear regression model  model = LinearRegression()  model.fit(X\_clean\_test, y\_clean\_test)  # Predicting the dependent variable values  y\_pred = model.predict(X\_clean\_test)  # Calculating R^2 and RMSE  mae = mean\_absolute\_error(y\_clean\_test, y\_pred)  mse = mean\_squared\_error(y\_clean\_test, y\_pred)  r2 = r2\_score(y\_clean\_test, y\_pred)  rmse = np.sqrt(mean\_squared\_error(y\_clean\_test, y\_pred))  mae, mse, r2, rmse  plt.figure(figsize=(**15**, **6**))  scatter = plt.scatter(y\_clean\_test, y\_pred, c=np.abs(y\_clean\_test-y\_pred), cmap='viridis')  plt.colorbar(scatter, label='Absolute Error')  plt.plot([**0**, max(y\_clean\_test)], [**0**, max(y\_clean\_test)], color='red') # Diagonal line starting from (0,0)  plt.title('Log Mismanaged Plastics (tons) = f(Log Population, Log GDP)', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.xlabel('Log Population & Log GDP', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Log Mismanaged Plastics (tons)', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.legend(['Data Points', 'Regression Line'])  plt.xlim(left=**0**)  plt.ylim(bottom=**0**)  plt.tight\_layout()  plt.show()  # Creating a 3D plot to visualize the regression plane  fig = plt.figure(figsize=(**18**, **6**))  ax = fig.add\_subplot(**111**, projection='3d')  # Creating a meshgrid for the plane  x1, x2 = np.meshgrid(np.linspace(X\_clean\_test['Log\_GDP'].min(), X\_clean\_test['Log\_GDP'].max(), **100**),  np.linspace(X\_clean\_test['Log\_Population'].min(), X\_clean\_test['Log\_Population'].max(), **100**))  # Predicting values using the model to find the coordinates for the Z axis  Z = model.predict(np.c\_[x1.ravel(), x2.ravel()])  Z = Z.reshape(x1.shape)  # Plotting the regression plane  ax.plot\_surface(x1, x2, Z, color='None', alpha=**0.5**)  # Scatter plot of the original data  scatter = ax.scatter(X\_clean\_test['Log\_GDP'], X\_clean\_test['Log\_Population'], y\_clean\_test,  c=y\_clean\_test, cmap='viridis')  cbar = fig.colorbar(scatter)  cbar.set\_label('Log\_Per Mismanaged Plastics')  # Setting labels  ax.set\_xlabel('Log\_GDP', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  ax.set\_ylabel('Log\_Population', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  ax.set\_zlabel('Log\_Mismanaged\_Plastics', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  ax.set\_title('3D Visualization of Linear Regression', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.show()  #### Linear Regression.  # Features and Target  X = filtered\_data[['Log\_GDP','Log\_Population']]  y = filtered\_data['Log\_Per Mismanaged Plastics']  # Splitting the data into training and testing sets (80% train, 20% test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=**0.2**, random\_state=**42**)  # Initialize and train the Linear Regression model  lr = LinearRegression()  lr.fit(X\_train, y\_train)  # Predictions  y\_pred\_train = lr.predict(X\_train)  y\_pred\_test = lr.predict(X\_test)  # Getting the coefficient of the model  coefficient = lr.coef\_  # Evaluation  mae = mean\_absolute\_error(y\_test, y\_pred\_test)  mse = mean\_squared\_error(y\_test, y\_pred\_test)  r2 = r2\_score(y\_test, y\_pred\_test)  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test))  # Printing the coefficient and intercept  **print**(f'Coefficient: {coefficient}')  mae, mse, r2, rmse  ##  # Scatter plot of actual vs predicted values  plt.figure(figsize=(**18**, **6**))  # Scatter plot for training data  plt.scatter(y\_train, y\_pred\_train, color='blue', alpha=**0.5**, label='Train')  # Scatter plot for testing data  plt.scatter(y\_test, y\_pred\_test, color='red', alpha=**0.5**, label='Test')  # Plotting the line for perfect prediction  min\_val = min(min(y\_train), min(y\_test))  max\_val = max(max(y\_train), max(y\_test))  plt.plot([min\_val, max\_val], [min\_val, max\_val], color='black', linestyle='--')  plt.xlabel('Actual Log(Mismanaged Plastics (tons))',fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Log(Mismanaged Plastics (tons))', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Actual vs Predicted Log(Mismanaged Plastics (tons))',fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.legend()  plt.grid(True)  plt.show()  # Calculating residuals for training and testing data  residuals\_train = y\_train - y\_pred\_train  residuals\_test = y\_test - y\_pred\_test  # Function to create a colored residual plot with a lowess line  **def** **colored\_residual\_plot**(predictions, residuals, title):  # Creating a scatter plot with a color map  scatter = plt.scatter(predictions, residuals, c=residuals, cmap='viridis')  plt.colorbar(scatter).set\_label('Residual Value')    # Adding a lowess line  sns.regplot(x=predictions, y=residuals, lowess=True, scatter=False, color='red', line\_kws={'lw': **1**})    # Adding labels and title  plt.xlabel('Predicted Values',fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals',fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title(title, fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.figure(figsize=(**18**, **6**))  # Creating the residual plot using seaborn for training data  # Creating a colored residual plot for testing data  plt.subplot(**1**, **2**, **1**)  colored\_residual\_plot(y\_pred\_test, residuals\_test, 'Residual Plot (Testing Data)')  # Residuals Distribution  plt.subplot(**1**, **2**, **2**)  plt.hist(residuals\_test, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  # Displaying the plots  plt.tight\_layout()  plt.show()  ## Polynomial features  # Degree of the polynomial. You can modify this value to have more complex polynomials  degree = **2**  # Creating a model that includes polynomial features  polyreg = make\_pipeline(PolynomialFeatures(degree),LinearRegression())  # Fitting the model  polyreg.fit(X\_train, y\_train)  # Making predictions  y\_pred\_train\_poly = polyreg.predict(X\_train)  y\_pred\_test\_poly = polyreg.predict(X\_test)  # Evaluating the model  mae\_poly = mean\_absolute\_error(y\_test, y\_pred\_test\_poly)  mse\_poly = mean\_squared\_error(y\_test, y\_pred\_test\_poly)  r2\_poly = r2\_score(y\_test, y\_pred\_test\_poly)  rmse\_poly = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test\_poly))  mae\_poly, mse\_poly, r2\_poly, rmse\_poly  # Visualizing the results  residuals= y\_test-y\_pred\_test\_poly  plt.figure(figsize=(**18**, **6**))  plt.scatter(y\_test, y\_test, color='red', label='True values', alpha=**0.6**, s=**50**, edgecolor='white')  plt.scatter(y\_test, y\_pred\_test\_poly, color='blue', label='Predicted values', alpha=**0.6**, s=**50**, edgecolor='white')  plt.title('Polynomial Regression', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.legend()  plt.show()  plt.figure(figsize=(**18**, **6**))  # Creating a colored residual plot for testing data  plt.subplot(**1**, **2**, **1**)  colored\_residual\_plot(y\_pred\_test\_poly, residuals, 'Residual Plot (Testing Data)')  plt.subplot(**1**,**2**,**2**)  plt.hist(residuals, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  #### Gradient Boosting regressor  # Initialize the Gradient Boosting regressor  gbr = GradientBoostingRegressor(random\_state=**42**)  # Train the model  gbr.fit(X\_train, y\_train)  # Get feature importances  feature\_importances = gbr.feature\_importances\_  # Predictions  y\_pred\_gbr\_test = gbr.predict(X\_test)  # Evaluation  mae\_gbr = mean\_absolute\_error(y\_test, y\_pred\_gbr\_test)  mse\_gbr = mean\_squared\_error(y\_test, y\_pred\_gbr\_test)  r2\_gbr = r2\_score(y\_test, y\_pred\_gbr\_test)  rmse\_gbr = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_gbr\_test))  mae\_gbr, mse\_gbr, r2\_gbr, rmse\_gbr  # Visualization of True vs. Predicted Values and Residuals  plt.figure(figsize=(**18**, **6**))  # True vs. Predicted Values  plt.subplot(**1**, **3**, **1**)  plt.scatter(y\_test, y\_pred\_gbr\_test, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=**2**)  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  # Residuals Plot  plt.subplot(**1**, **3**, **2**)  residuals = y\_test - y\_pred\_gbr\_test  plt.scatter(y\_pred\_gbr\_test, residuals, color='red', alpha=**0.6**)  plt.axhline(**0**, color='black', linestyle='--', lw=**2**)  plt.xlabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Plot', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**,**3**,**3**)  plt.hist(residuals, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  True vs. Predicted Values (Left):  The blue points represent the predicted values against the true values.  The dotted line represents the ideal scenario where every prediction matches the true value.  The closer the blue points are to this line, the better the prediction.  Residuals Plot (Right):  The red points represent the residuals (differences between predicted **and** true values) **for** each predicted value.  The horizontal black dashed line represents a residual of zero.  Ideally, we'd like the residuals to be randomly scattered around the horizontal axis, which would indicate that the model's errors are random.  # Define the hyperparameters and their possible values  param\_grid = {  'n\_estimators': [**50**, **100**, **150**],  'learning\_rate': [**0.01**, **0.05**, **0.1**, **0.5**],  'max\_depth': [**3**, **4**, **5**]  }  # Initialize GridSearchCV with cross-validation  grid\_search = GridSearchCV(GradientBoostingRegressor(random\_state=**42**),  param\_grid,  cv=**5**,  scoring='r2',  verbose=**1**)  # Fit the model  grid\_search.fit(X\_train, y\_train)  # Extract best hyperparameters from the grid search  best\_params = grid\_search.best\_params\_  best\_score = grid\_search.best\_score\_  best\_params, best\_score  # Train the Gradient Boosting regressor with best hyperparameters  gbr\_best = GradientBoostingRegressor(\*\*best\_params, random\_state=**42**)  gbr\_best.fit(X\_train, y\_train)  # Predictions on the test set  y\_pred\_gbr\_best\_test = gbr\_best.predict(X\_test)  # Evaluation  mae\_gbr\_best = mean\_absolute\_error(y\_test, y\_pred\_gbr\_best\_test)  mse\_gbr\_best = mean\_squared\_error(y\_test, y\_pred\_gbr\_best\_test)  r2\_gbr\_best = r2\_score(y\_test, y\_pred\_gbr\_best\_test)  rmse\_gbr\_best = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_gbr\_best\_test))  mae\_gbr\_best, mse\_gbr\_best, r2\_gbr\_best, rmse\_gbr\_best  # Visualization of True vs. Predicted Values and Residuals  plt.figure(figsize=(**15**, **6**))  # True vs. Predicted Values  plt.subplot(**1**, **3**, **1**)  plt.scatter(y\_test, y\_pred\_gbr\_best\_test, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k-', lw=**2**, color= 'orange')  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  # Residuals Plot  plt.subplot(**1**, **3**, **2**)  residuals\_best = y\_test - y\_pred\_gbr\_best\_test  plt.scatter(y\_pred\_gbr\_best\_test, residuals\_best, color='red', alpha=**0.6**)  plt.axhline(**0**, color='black', linestyle='--', lw=**2**)  plt.xlabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Plot', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**,**3**,**3**)  plt.hist(residuals\_best, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  True vs. Predicted Values (Left):  The model's predictions are closer to the ideal line, indicating better alignment with the true values.  Residuals Plot (Right):  The residuals are more randomly scattered around the horizontal axis, suggesting the model errors are more random **and** less biased.  #### SVR  # Initialize the SVR  svr = SVR()  # Train the model  svr.fit(X\_train, y\_train)  # Predictions on the test set  y\_pred\_svr\_test = svr.predict(X\_test)  # Evaluation  mae\_svr = mean\_absolute\_error(y\_test, y\_pred\_svr\_test)  mse\_svr = mean\_squared\_error(y\_test, y\_pred\_svr\_test)  r2\_svr = r2\_score(y\_test, y\_pred\_svr\_test)  rmse\_svr = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_svr\_test))  mae\_svr, mse\_svr, r2\_svr, rmse\_svr  # Visualization  plt.figure(figsize=(**18**, **6**))  # True vs. Predicted Values  plt.subplot(**1**, **4**, **1**)  plt.scatter(y\_test, y\_pred\_svr\_test, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.colorbar().set\_label('True Values')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=**2**)  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  # Residuals Distribution  plt.subplot(**1**, **3**, **2**)  residuals\_svr = y\_test - y\_pred\_svr\_test  plt.scatter(y\_pred\_svr\_test, residuals\_svr, color='red', alpha=**0.6**)  plt.axhline(**0**, color='black', linestyle='--', lw=**2**)  plt.xlabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Plot', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**, **3**, **3**)  plt.hist(residuals\_svr, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  ### Feature engineering  # Creating polynomial features  poly = PolynomialFeatures(degree=**2**)  X\_train\_poly = poly.fit\_transform(X\_train)  X\_test\_poly = poly.transform(X\_test)  # Initialize the SVR  svr = SVR()  # Train the model  svr.fit(X\_train\_poly, y\_train)  # Predictions on the test set  y\_pred\_svr\_test = svr.predict(X\_test\_poly)  # Evaluation  mae\_svr = mean\_absolute\_error(y\_test, y\_pred\_svr\_test)  mse\_svr = mean\_squared\_error(y\_test, y\_pred\_svr\_test)  r2\_svr = r2\_score(y\_test, y\_pred\_svr\_test)  rmse\_svr = np.sqrt(mse\_svr)  mae\_svr, mse\_svr, r2\_svr, rmse\_svr  # Visualization  plt.figure(figsize=(**18**, **6**))  # True vs. Predicted Values  plt.subplot(**1**, **3**, **1**)  plt.scatter(y\_test, y\_pred\_svr\_test, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.colorbar().set\_label('True Values')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=**2**)  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  # Residuals Distribution  plt.subplot(**1**, **3**, **2**)  residuals\_svr = y\_test - y\_pred\_svr\_test  plt.scatter(y\_pred\_svr\_test, residuals\_svr, color='red', alpha=**0.6**)  plt.axhline(**0**, color='black', linestyle='--', lw=**2**)  plt.xlabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Plot', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**, **3**, **3**)  plt.hist(residuals\_svr, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  #### Neural Net  # Extracting GDP per capita and per capita plastic waste values  #X = df\_per\_capita\_vs\_gdp[['GDP']]  #y = df\_per\_capita\_vs\_gdp['Mismanaged Plastics']  X = filtered\_data[['Log\_GDP', 'Log\_Population']]  y = filtered\_data['Log\_Per Mismanaged Plastics']  # Splitting data into training and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=**0.2**, random\_state=**42**)  # Creating a neural network model with scaled input  model = Pipeline(steps=[('scaler', StandardScaler()),  ('classifier', MLPRegressor(hidden\_layer\_sizes=(**50**, **50**), max\_iter=**500**, random\_state=**42**))])  # Training the model  model.fit(X\_train, y\_train)  # Predicting the test set results  y\_pred = model.predict(X\_test)  # Evaluating the model R^2  test\_score = model.score(X\_test, y\_test)  # Calculate the Mean Squared Error (test loss)  test\_loss = mean\_squared\_error(y\_test, y\_pred)  # Calculate the Mean Absolute Error  test\_mae = mean\_absolute\_error(y\_test, y\_pred)  # Calculate the Root Mean Squared Error (RMSE)  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  test\_score, test\_loss, test\_mae, rmse  # Visualization  plt.figure(figsize=(**18**, **6**))  # True vs. Predicted Values  plt.subplot(**1**, **3**, **1**)  plt.scatter(y\_test, y\_pred, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.colorbar().set\_label('True Values')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=**2**)  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  # Residuals Distribution  plt.subplot(**1**, **3**, **2**)  residuals = y\_test - y\_pred  plt.scatter(y\_pred, residuals, color='red', alpha=**0.6**)  plt.axhline(**0**, color='black', linestyle='--', lw=**2**)  plt.xlabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Plot', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.subplot(**1**, **3**, **3**)  plt.hist(residuals, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  # Normalizing the features and target variable  scaler\_X = StandardScaler().fit(X\_train)  scaler\_y = StandardScaler().fit(y\_train.values.reshape(-**1**, **1**))  X\_train\_normalized = scaler\_X.transform(X\_train)  y\_train\_normalized = scaler\_y.transform(y\_train.values.reshape(-**1**, **1**))  X\_test\_normalized = scaler\_X.transform(X\_test)  y\_test\_normalized = scaler\_y.transform(y\_test.values.reshape(-**1**, **1**))  # Define the neural network model  model = Sequential([  Dense(**64**, activation='relu', input\_shape=(X\_train\_normalized.shape[**1**],)),  Dense(**32**, activation='relu'),  Dense(**1**)  ])  # Compile the model  model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['mae'])  history = model.fit(X\_train\_normalized, y\_train\_normalized, epochs=**50**, validation\_split=**0.2**, verbose=**0**)  # Evaluate the model on the test set  test\_loss, test\_mae = model.evaluate(X\_test\_normalized, y\_test\_normalized)  test\_loss, test\_mae  # Predictions using the neural network on the test set  y\_pred\_nn = model.predict(X\_test\_normalized)  y\_pred\_nn\_denormalized = scaler\_y.inverse\_transform(y\_pred\_nn)  # Ensuring y\_test is a numpy array for consistent arithmetic operations  y\_test\_array = y\_test.to\_numpy().reshape(-**1**, **1**)  r2 = r2\_score(y\_test\_array, y\_pred\_nn\_denormalized)  mse = mean\_squared\_error(y\_test\_array, y\_pred\_nn\_denormalized)  rmse = np.sqrt(mse)  mae = mean\_absolute\_error(y\_test\_array, y\_pred\_nn\_denormalized)  # Now you can print or return all the evaluation metrics  r2, mse, rmse, mae  residuals\_nn = y\_test\_array - y\_pred\_nn\_denormalized  # Visualization  plt.figure(figsize=(**20**, **6**))  # Training History  plt.subplot(**1**, **4**, **1**)  plt.plot(history.history['loss'], label='Training Loss', color='blue')  plt.plot(history.history['val\_loss'], label='Validation Loss', color='orange')  plt.xlabel('Epoch', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Mean Squared Error', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Training History', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.legend()  # True vs. Predicted Values  plt.subplot(**1**, **4**, **2**)  plt.scatter(y\_test, y\_pred\_nn\_denormalized, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.plot([min(y\_test), max(y\_test)],  [min(y\_test), max(y\_test)], color='red')  plt.colorbar().set\_label('True Values')  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Subplot for Residuals  plt.subplot(**1**, **4**, **3**)  plt.scatter(y\_test, residuals\_nn, c=y\_test, cmap='CMRmap', alpha=**0.7**)  plt.axhline(y=**0**, color='red', linestyle='--') # Adding a horizontal line at 0  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Residuals Distribution  plt.subplot(**1**, **4**, **4**)  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.hist(residuals\_nn, bins=**20**, color='skyblue', edgecolor='black')  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  #### Fine tuning  Learning rate  # Given our constraints, we'll manually adjust hyperparameters and evaluate the model's performance  # First, we'll adjust the learning rate  learning\_rates = [**0.01**, **0.001**, **0.0001**, **0.00001**]  results = {}  **for** lr **in** learning\_rates:  # Define the model with modified learning rate  model\_tuned = Sequential([  Dense(**64**, activation='relu', input\_shape=(X\_train\_normalized.shape[**1**],)),  Dense(**32**, activation='relu'),  Dense(**1**)  ])    # Compile the model with adjusted learning rate  model\_tuned.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=lr), loss='mean\_squared\_error', metrics=['mae'])  # Train the model (using a reduced number of epochs for quick iterations)  model\_tuned.fit(X\_train\_normalized, y\_train\_normalized, epochs=**50**, validation\_split=**0.2**, verbose=**0**)  # Evaluate the model on the test set  test\_loss\_tuned, test\_mae\_tuned = model\_tuned.evaluate(X\_test\_normalized, scaler\_y.transform(y\_test.values.reshape(-**1**, **1**)), verbose=**0**)    results[lr] = (test\_loss\_tuned, test\_mae\_tuned)  results  Epochs  epochs = [**50**, **100**, **250**, **500**]  results = {}  **for** epoch **in** epochs:  model\_tuned1 = Sequential([  Dense(**64**, activation='relu', input\_shape=(X\_train\_normalized.shape[**1**],)),  Dense(**32**, activation='relu'),  Dense(**1**)  ])  model\_tuned1.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=**0.001**),  loss='mean\_squared\_error', metrics=['mae'])  # Train with different epochs  model\_tuned1.fit(X\_train\_normalized, y\_train\_normalized,  epochs=epoch, validation\_split=**0.2**, verbose=**0**)  # Evaluate  test\_loss\_tuned, test\_mae\_tuned = model\_tuned1.evaluate(  X\_test\_normalized,  scaler\_y.transform(y\_test.values.reshape(-**1**, **1**)), verbose=**0**)  # Record results  results[epoch] = (test\_loss\_tuned, test\_mae\_tuned)  results  #### Activations  ReLU **is** a popular choice due to its simplicity **and** effectiveness. However, **in** some cases, other functions like Tanh **or** Sigmoid might work better.  activations = ['relu', 'tanh', 'sigmoid']  results = {}  **for** activation **in** activations:  model2 = Sequential([  Dense(**64**, activation=activation, input\_shape=(X\_train.shape[**1**],)),  Dense(**32**, activation=activation),  Dense(**1**)  ])  model2.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=**0.001**),  loss='mean\_squared\_error', metrics=['mae'])  # Train with different epochs  model2.fit(X\_train\_normalized, y\_train\_normalized,  epochs=**500**, validation\_split=**0.2**, verbose=**0**)  # Evaluate  test\_loss\_tuned, test\_mae\_tuned = model2.evaluate(  X\_test\_normalized,  scaler\_y.transform(y\_test.values.reshape(-**1**, **1**)), verbose=**0**)  # Record results  results[activation] = (test\_loss\_tuned, test\_mae\_tuned)  results  # Define the neural network model  best\_model = Sequential([  Dense(**64**, activation='relu', input\_shape=(X\_train\_normalized.shape[**1**],)),  Dense(**32**, activation='relu'),  Dense(**1**)  ])  # Compile the model  best\_model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=**0.001**), loss='mean\_squared\_error', metrics=['mae'])  # Train the model  history = best\_model.fit(X\_train\_normalized, y\_train\_normalized, epochs=**500**, validation\_split=**0.2**, verbose=**0**)  # Evaluate the model on the test set  test\_loss, test\_mae = best\_model.evaluate(X\_test\_normalized, scaler\_y.transform(y\_test.values.reshape(-**1**, **1**)))  test\_loss, test\_mae  # Predictions using the neural network on the test set  y\_pred\_nn = best\_model.predict(X\_test\_normalized)  # Predictions using the neural network on the test set  y\_pred\_nn\_denormalized = scaler\_y.inverse\_transform(y\_pred\_nn)  # Ensuring y\_test is a numpy array for consistent arithmetic operations  y\_test\_array = y\_test.to\_numpy().reshape(-**1**, **1**)  r2 = r2\_score(y\_test\_array, y\_pred\_nn\_denormalized)  mse = mean\_squared\_error(y\_test\_array, y\_pred\_nn\_denormalized)  rmse = np.sqrt(mse)  mae = mean\_absolute\_error(y\_test\_array, y\_pred\_nn\_denormalized)  # Now you can print or return all the evaluation metrics  r2, mse, rmse, mae  residuals\_nn = y\_test\_array - y\_pred\_nn\_denormalized  # Visualization  plt.figure(figsize=(**18**, **6**))  # Training History  plt.subplot(**1**, **4**, **1**)  plt.plot(history.history['loss'], label='Training Loss', color='blue')  plt.plot(history.history['val\_loss'], label='Validation Loss', color='orange')  plt.xlabel('Epoch', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Mean Squared Error', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Training History', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.legend()  # True vs. Predicted Values  plt.subplot(**1**, **4**, **2**)  plt.scatter(y\_test, y\_pred\_nn\_denormalized, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.plot([min(y\_test), max(y\_test)],  [min(y\_test), max(y\_test)], color='red')  plt.colorbar().set\_label('True Values')  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Subplot for Residuals  plt.subplot(**1**, **4**, **3**)  plt.scatter(y\_test, residuals\_nn, c=y\_test, cmap='CMRmap', alpha=**0.7**)  plt.axhline(y=**0**, color='red', linestyle='--') # Adding a horizontal line at 0  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Residuals Distribution  plt.subplot(**1**, **4**, **4**)  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.hist(residuals\_nn, bins=**20**, color='skyblue', edgecolor='black')  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  #### Trying Regulaization  model\_regularized = Sequential([  Dense(**64**, activation='relu', kernel\_regularizer=l1\_l2(l1=**0.01**, l2=**0.01**), input\_shape=(X\_train\_normalized.shape[**1**],)),  Dropout(**0.3**), # Dropout layer with 30% dropout rate  Dense(**32**, activation='relu', kernel\_regularizer=l1\_l2(l1=**0.01**, l2=**0.01**)),  Dropout(**0.3**), # Another dropout layer  Dense(**1**)  ])  # Compile the model  model\_regularized.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=**0.001**), loss='mean\_squared\_error', metrics=['mae'])  # Train the model  history = model\_regularized.fit(X\_train\_normalized, y\_train\_normalized, epochs=**500**, validation\_split=**0.2**, verbose=**0**)  # Evaluate the model on the test set  test\_loss, test\_mae = model\_regularized.evaluate(X\_test\_normalized, scaler\_y.transform(y\_test.values.reshape(-**1**, **1**)))  test\_loss, test\_mae  # Predictions using the neural network on the test set  y\_pred\_nn = model\_regularized.predict(X\_test\_normalized)  # Predictions using the neural network on the test set  y\_pred\_nn\_denormalized = scaler\_y.inverse\_transform(y\_pred\_nn)  # Ensuring y\_test is a numpy array for consistent arithmetic operations  y\_test\_array = y\_test.to\_numpy().reshape(-**1**, **1**)  r2 = r2\_score(y\_test\_array, y\_pred\_nn\_denormalized)  mse = mean\_squared\_error(y\_test\_array, y\_pred\_nn\_denormalized)  rmse = np.sqrt(mse)  mae = mean\_absolute\_error(y\_test\_array, y\_pred\_nn\_denormalized)  # Now you can print or return all the evaluation metrics  r2, mse, rmse, mae  # Visualization  plt.figure(figsize=(**20**, **6**))  # Training History  plt.subplot(**1**, **4**, **1**)  plt.plot(history.history['loss'], label='Training Loss', color='blue')  plt.plot(history.history['val\_loss'], label='Validation Loss', color='orange')  plt.xlabel('Epoch', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Mean Squared Error', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Training History', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.legend()  # True vs. Predicted Values  plt.subplot(**1**, **4**, **2**)  plt.scatter(y\_test, y\_pred\_nn\_denormalized, c=y\_test, cmap='viridis', alpha=**0.7**)  plt.plot([min(y\_test), max(y\_test)],  [min(y\_test), max(y\_test)], color='red')  plt.colorbar().set\_label('True Values')  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Predicted Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('True vs. Predicted Values', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Subplot for Residuals  plt.subplot(**1**, **4**, **3**)  plt.scatter(y\_test, residuals\_nn, c=residuals\_nn, cmap='plasma', alpha=**0.7**)  plt.axhline(y=**0**, color='red', linestyle='--') # Adding a horizontal line at 0  plt.xlabel('True Values', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.grid(True)  # Residuals Distribution  plt.subplot(**1**, **4**, **4**)  plt.hist(residuals\_nn, bins=**20**, color='skyblue', edgecolor='black')  plt.axvline(**0**, color='red', linestyle='--', lw=**2**)  plt.xlabel('Residuals', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.ylabel('Frequency', fontsize=**18**, fontname="Times New Roman", fontweight="bold")  plt.title('Residuals Distribution', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.tight\_layout()  plt.show()  # Extracting clean GDP per capita and per capita plastic waste values  data\_cleaned = filtered\_data.dropna(subset=['Log\_GDP', 'Log\_Population',  'Log\_Per Mismanaged Plastics'])  X\_cleaned = filtered\_data[['Log\_GDP', 'Log\_Population']].values  y\_cleaned = filtered\_data['Log\_Per Mismanaged Plastics'].values  # Load the world map  world = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_lowres'))  # Print first 5 rows  **print**(X\_cleaned[:**5**])  # Predictions using the neural network on the entire dataset  scaler\_X = StandardScaler().fit(X\_cleaned)  y\_pred\_all = best\_model.predict(scaler\_X.transform(X\_cleaned))  # Compute residuals for each country  data\_cleaned['residuals'] = scaler\_y.inverse\_transform(y\_cleaned.reshape(-**1**, **1**)) - scaler\_y.inverse\_transform(y\_pred\_all)  # Merge the world map with the residuals  merged = world.set\_index('name').join(data\_cleaned.set\_index('Country'))  # Plotting  fig, ax = plt.subplots(**1**, **1**, figsize=(**15**, **10**))  world.boundary.plot(ax=ax, linewidth=**1**)  merged.plot(column='residuals', ax=ax, legend=True, cmap='coolwarm',  legend\_kwds={'label': 'Residuals (Difference between True and Predicted Values)'})  plt.title('Geographical Distribution of Model Residuals', fontsize=**25**, fontname="Times New Roman", fontweight="bold")  plt.show() |

1. See the link for details: <https://ourworldindata.org/plastic-pollution> [↑](#footnote-ref-1)
2. See the link: <https://data.worldbank.org/indicator/NY.GDP.MKTP.CD?view=chart> [↑](#footnote-ref-2)