**Analysing the Evolution of the Irish Population: A Data Analytics Perspective (1926-2023)**

**MSC Data Analytics CA1**

*Author: Simone Finelli*

*Student ID: sba22524*

*E-mail: sba22524@student.cct.ie*

*MSc in Data Analytics*

*Programming for DA, Statistics for Data Analytics, Machine Learning for Data Analysis, Data Preparation & Visualisation*

*David Gonzalez, Marina Iantorno, Muhammad Iqbal, David McQuaid*

*CCT College*

*Dublin, Ireland*

**Abstract**

This report presents a comprehensive analysis of the Irish population from 1926 to 2023, focusing on its age and sex distribution. The study aims to understand how the age composition of the population has evolved over time and to explore the dynamics of new births on an annual basis. Furthermore, this research provides population forecasts to gain insights into how the population is likely to change in the future.

The aging population is a global demographic challenge, and Ireland is no exception. By examining the age distribution over nearly a century, this report assesses whether the Irish population is getting older and the implications of this trend. A closer look at annual birth rates will shed light on the patterns and fluctuations in new births, allowing us to better understand the factors influencing demographic change.

In addition, this report employs data analytics techniques to forecast future population numbers, providing valuable insights for policymakers and stakeholders in Ireland. Understanding the future population trends is crucial for planning healthcare, social services, and infrastructure to meet the needs of a changing society.

Through data visualization and statistical analysis, this report offers a data-driven exploration of the Irish population's past, present, and future, making it a valuable resource for anyone interested in demographic trends and population projections.

*Keywords: CRISP-DM, Price Level Indices, Purchasing Power Parities, Random Forest, XGBoost, Neural Network*

**Table of Contents**

[Statistics – Dataset summarisation 5](#_Toc150104434)

[Statistics – Discrete distributions 9](#_Toc150104435)

[Statistics – Normal distribution 11](#_Toc150104436)

[Modeling 14](#_Toc150104437)

[Evaluation 35](#_Toc150104438)

[Deployment 37](#_Toc150104439)

[Reference List 38](#_Toc150104440)

**Table of Figures**

Figure 1. PLI head [7](#_heading=h.2et92p0)

Figure 2. PLI shape and info  [8](#_heading=h.2et92p0)

Figure 3. PLI unique values of columns [8](#_heading=h.2et92p0)

Figure 4. PLI description [8](#_heading=h.2et92p0)

Figure 5. PLI dataset Price Level Indices over time for countries  [9](#_heading=h.2et92p0)

Figure 6. PPP head [10](#_heading=h.2et92p0)

Figure 7. PPP pivot and head  [10](#_heading=h.2et92p0)

Figure 8. PPP shape and info  [11](#_heading=h.2et92p0)

Figure 9. PPP description  [11](#_heading=h.2et92p0)

Figure 10. PPP dataset null values  [11](#_heading=h.2et92p0)

Figure 11. PPP columns renaming [12](#_heading=h.2et92p0)

Figure 12. df dataset creation from PPP and PLI outer merge  [12](#_heading=h.2et92p0)

Figure 13. df drop of null values  [12](#_heading=h.2et92p0)

Figure 14. Mapping location integers  [13](#_heading=h.2et92p0)

Figure 15. Scaling the dataset df  [13](#_heading=h.2et92p0)

Figure 16. Creating our ‘X’ and ‘y’ labels  [14](#_heading=h.2et92p0)

Figure 17. Food Value over time. See the location nine value in the top right  [15](#_heading=h.2et92p0)

Figure 18. Alcohol Value over time. See the location nine value in the top right  [15](#_heading=h.2et92p0)

Figure 19. Dropping line 270 from df  [16](#_heading=h.2et92p0)

Figure 20. Feature ‘A0103’ after removing the outlier  [16](#_heading=h.2et92p0)

Figure 21. ‘y’ label visualisations for Austria and Belgium  [17](#_heading=h.2et92p0)

Figure 22. ‘y’ label visualisations of the value for each country over time  [17](#_heading=h.2et92p0)

Figure 23. Train Test Split [1](#_heading=h.2et92p0)9

Figure 24. Simple Random Forest training [1](#_heading=h.2et92p0)9

Figure 25. Simple Random Forest scoring [1](#_heading=h.2et92p0)9

Figure 26. Simple Random Forest visualization code20

Figure 27. Simple Random Forest score visualization20

Figure 28. Previous Random Forest score result21

Figure 29. Random Forest training with scaling22

Figure 30. Random Forest scoring with scaling22

Figure 31. Scaled Random Forest score visualization 23

Figure 32. Random Forest parameters to tune24

Figure 33. Random Forest best parameters  [2](#_heading=h.2et92p0)4

Figure 34. Best Random Forest score visualization 25

Figure 35. Random Forest score comparison 25

Figure 36. XGBoost libraries 26

Figure 37. XGBoost train test split 26

Figure 38. Simple XGBoost training 26

Figure 39. Simple XGBoost score visualization 27

Figure 40. Simple XGBoost score 27

Figure 41. Simple XGBoost MSE score visualisation 28

Figure 42. XGBoost train test split 28

Figure 43. XGBoost comparison with initial model 28

Figure 44. XGBoost parameters to tune 29

Figure 45. XGBoost tuning 29

Figure 46. XGBoost GridSearchCV best parameters 29

Figure 47. Best XGBoost GridSearchCV score 30

Figure 48. Best XGBoost GridSearchCV RMSE 30

Figure 49. XGBoost l1 optimization 31

Figure 50. XGBoost l2 optimization 31

Figure 51. XGBoost l1 MSE score 32

Figure 52. XGBoost models comparison 32

Figure 53. Simple Neural Network training 33

Figure 54. Simple Neural Network loss change over epochs 34

Figure 55. Simple Neural Network MSE change over epochs 34

Figure 56. Neural Network parameters to tune 35

Figure 57. Neural Network code for hyperparameter tuning 36

Figure 58. Neural Network best activation function 36

Figure 59. Best Neural Network loss change over epochs 38

## Statistics – Dataset summarisation

The data we are working contains data regarding the population of Ireland divided by Year, Age and Sex. This can be easily visualized with a quick exploration of the dataset:

A screenshot of a computer

Description automatically generated

After analysing our dataset, we can recall the following key information:

* We don’t have null values, but we are missing data for years between 1926 and 1996.
* “STATIC Label” and “UNIT” columns can be dropped as they only contain a single unique value.

At this point we couldn’t proceed with a descriptive statistic of the dataset as it contained redundant data given “All Ages” vs other ages and “Both sexes” vs “Male and Female” data. For this reason, we processed multiple filtered versions of the dataset to obtain relevant information.

First thing we want to double check if the sum of the “Male” and “Female” population for each year is equal to the one of “Both Sexes”.

A graph of a number of people

Description automatically generated

A graph of a number of people

Description automatically generated with medium confidence

Even though there is no null value in the dataset, there might be some wrong data for specific years. To spot this kind of error the previous two visualizations were built to confirm that no error is present.

Now that we know that there is no error, we can gain descriptive information for male and female population:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

To better compare this data, a boxplot was built:

A diagram of a couple of blue and orange squares

Description automatically generated

Looking at numbers and plot, we can say that from 1926 to 2023 (remember that we have missing years) the average female population has been higher that the male one with very similar min, 25% and 50% with respect to the male population but visibly higher 75% and max values.

We can thus say that, observing the data available to use, Ireland has a slightly higher female population. This may also relate to a higher longer life expectancy for women.

Instead of just looking at the population sex over years, we now want to get some information regarding the population age. This was possible thanks to a manipulation of the “Single Year of Age” column:

A screenshot of a computer screen

Description automatically generated

To have a good understanding of how the population age is changing over time, we want to compute:

* Mean. It is the average age of the population; if is increasing over time, it may suggest that the population is aging, and the older generations are becoming a larger proportion of the population.
* Median. It is the middle age in a population when ages are sorted from youngest to oldest; if is increasing, it suggests that the population is aging, as the midpoint of ages is moving toward older individuals.
* Mode. It is the age that appears most frequently in the population; if it is increasing, it could indicate that there is a growing cohort of people at a specific age, which may be due to factors like increased birth rates.
* Standard deviation. It is a measure of the spread or dispersion of ages in the population, if it is high it indicates greater variability in ages, which may imply a more diverse age structure in the population.

With the data available we obtained the following data:

* The mean is 34.95 years.
* The median is 30 years.
* The mode is 43 years.
* The standard deviation is 22.41 years.

From this data we can say that, looking at the standard deviation, the population of Ireland appears to have a moderately diverse age structure, with a mix of younger and older individuals.

The presence of a mode at 43 years suggests the existence of a significant age cohort, which may be attributed to specific demographic events or trends.

One last question we want to answer is: Ireland population is getting older?

To be able to answer this question I also computed the age mean for few years in the last decade and those are the results I got:

* Mean age in 2013 is 36.07 years.
* Mean age in 2018 is 37.25 years.
* Mean age in 2023 is 38.61 years.

This does demonstrate that the population is getting older with a higher life expectancy.

Let’s also visualize this:

A graph of a number of people

Description automatically generated

The graph shows the normalized population for different years by age. We can see that the age in the last decade (2013 to 2023) is becoming more and more skewed, moving from left to right as age is increasing in the last years.

## Statistics – Discrete distributions

The Poisson distribution is often used to model count data, such as the number of events or occurrences within a fixed interval. As we have data on population growth, we can model it as a Poisson distribution. Each year represents a fixed interval, and the number of births can be counted as events within that interval.

Given previous results we got regarding the increasing mean age over the last decade, we want to better understand if it only due to a higher life expectancy or also due a decreasing number of births.

To configure the Poisson distribution, we want to answer the following problem:

*In Ireland there are, in average, several births a year. If we watch Ireland any random year from the last decade, what is the probability of having new births population less than the average?*

The problem is so configured:

* Number of elements -> Number of births
* Specific place -> In Ireland
* Time frame -> Per Year

The first thing we calculate is the mean (λ) of the distribution, that is the average of new births from 1926 to 2023.

Once we got λ we derived that:

* The probability, in any random year from 1926 to 2023, to have less than λ new births is 50%
* The probability, in any random year from last decade, to have less than λ new births is 95%

This can also be seen from below graph:

A diagram of a normal distribution

Description automatically generated

Given that λ is 61764, the probability to have that same number of births from last decade is very low – indicating that natality is decreasing with respect to what it used to be.

If we further increase our dataset over several decades, we will notice that the mean of new births for a specific year would become closer and closer to the overall mean of new births. This means that with a larger dataset the two distributions plotted in before graph would come closer due to the law of large numbers.

Let’s try now to analyse the same problem with a different perspective while using a binomial distribution:

*If we randomly choose 500 people what is the probability to encounter at least 10 newborns?*

We start to solve the problem for 2023:

* Number of people -> Number of elements
* Newborns -> Characteristic or attribute
* Within 500 people -> Limit

For 2023 we got a 4% probability, but we also plotted a graph to show the results for other years available:

A graph of a number of years

Description automatically generated

Inherently, last decade shows a significant drop in this probability with a YoY result not promising.

If in this case we are to increase the dataset, the distribution of encountering newborns would approximate to a normal distribution due to the Central Limit Theorem. This means that we would have a relatively consistent probability of encountering newborns across years, even though it may be a small percentage.

## Statistics – Normal distribution

The variable we want to analyse as a normal distribution is the value of population over time.

Before analysing the normal distribution, we must make sure that the variable is normally distributed. To check this, we will use two methods:

* Q-Q Plot
* Shapiro-Wilk Normality Test

A Q-Q plot compares the distribution of the data to a normal distribution by plotting the quantiles of the data against the quantiles of the normal distribution. If the data is normally distributed, the points on the plot will form a straight line.

This is the result we got for our variable:

A graph of a graph with blue dots

Description automatically generated

The graph clearly shows that the quantiles try to form a straight line.

To double check the distribution we look at the Shapiro-Wilk Normality Test. This test measures the difference between the observed distribution and the expected normal distribution. It produces a p-value that, if less than a significance level, rejects the null hypothesis and conclude that the data is not normally distributed. For our study, we configure the significance level at 0.05 (means we are willing to accept a 5% chance of rejecting the null hypothesis when it's true).

From previous graph, we can see that p-value is slightly higher than the significance level, thus we can conclude that the variable is normally distributed.

If we plot the graph of the probability dense function, we can easily see that a curved bell shape is formed, confirming again the normal distribution:

A graph with a line

Description automatically generated

If we want to explain what it means to have a normally distributed data for the population over time variable, we can say:

* The symmetry suggests that fluctuations of population over time is balanced, meaning that if there is a year were the population is above average there is also one year where it is below.
* There is no high dispersion as the standard deviation value is not big with respect to the mean. This means that there are no years with extremely low or high population.

This analysis allows us to say that the population is growing naturally without any disruptive event creating crazy spikes or deeps of population. Still this analysis doesn't take into consideration the fact that the population is growing because of aging previously analysed.

The variable used for the discrete distributions is the Newborn population over time and, according to Q-Q Plot and Shapiro-Wilk Normality Test, it does follow a normal distribution and could have been used for our analysis:

A graph with blue dots

Description automatically generated

## Modeling

In this phase of the CRISP-DM methodology, we would ideally build and assess various models based on several different modeling techniques. (Nick Hotz, 2022)

The machine learning model which we opted for this dataset are listed below:

* Random Forest
* XGBoost
* Neural Network

The goal is to create different models which will be trained, tested, and evaluated based on various scoring techniques. Hyperparameters tuning will be implemented thanks to cross-validation technique to extract the most performing model that could eventually be deployed in production.

For this specific regression problem, R-squared (R²) and Mean Squared Error (MSE) are the metrics used to evaluate the model performance:

* Mean squared error (MSE) measures the average squared difference between the predicted and actual values of the target variable. It is a measure of the model's accuracy, with lower values indicating better performance.
* R-squared (R²) measures the proportion of variance in the target variable that is explained by the model. It ranges from 0 to 1, with higher values indicating better performance.

**Random Forest – Training with no optimizations**

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. (Chaya Bakshi, 2020)

In the following pages, we will proceed to describe the implementation of the Random Forest Regression model and improvements made from the last submitted report.

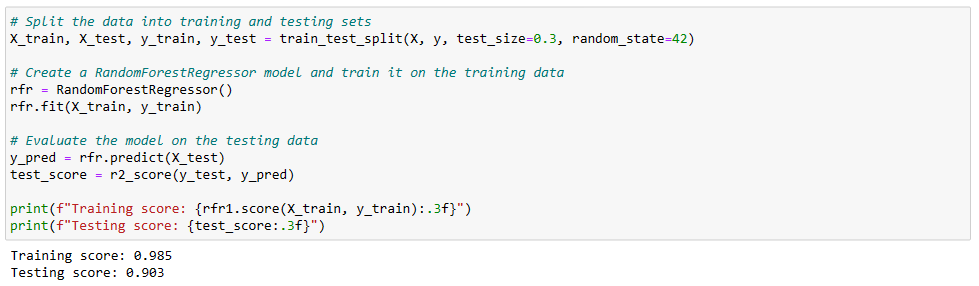
In the image below, the **train\_test\_split** function is imported from the sklearn library. Following this, the dataset is split into the training and testing sets; the **test\_size** indicates what proportion of the data will be used for testing: in this case it is 30%, with the remaining 70% being used for training. The **random\_state** number 42 is used for the reproducibility of the model:

Text

Description automatically generated

*Figure 23. Train Test Split*

A general regressor object is then created without specifying the number of n\_estimators, meaning that the default value of 100 is taken. The newly created regressor is then fitted with the X\_train and y\_train data previously created:



*Figure 24. Simple Random Forest training*

Once the regressor is fitted, R² and MSE scores are computed for both test and train:

A screenshot of a computer program

Description automatically generated with low confidence

*Figure 25. Simple Random Forest scoring*

And the following code is used to build a graph showcasing the distribution of actual and predicted values for both train and test splits:

A picture containing text, screenshot, font

Description automatically generated

*Figure 26. Simple Random Forest visualization code*

A picture containing text, plot, diagram, line

Description automatically generated

*Figure 27. Simple Random Forest score visualization*

Without any parameter tuning the model seems to return not too bad results if looking just at the R² score.

Those are the model scores:

* R2 Training score 0.986
* R2 Testing score 0.906
* MSE Training score 10.665
* MSE Testing score 64.002

In this case, the model has a high R² training score, which suggests that it is able to accurately predict the target variable on the training data. However, the lower testing score indicates that the model may be overfitting to the training data, and may not perform as well on new, unseen data (as we can see from the plotted graph).

For this reason, we further evaluate the model and consider ways to reduce overfitting, testing normalization and cross-validation techniques.

To showcase improvements from our latest report submission, I recall previous results we got using a Random Forest model with 1000 estimators:

A screenshot of a computer code

Description automatically generated with low confidence

*Figure 28. Previous Random Forest score results*

The difference between results gotten from previous report are mainly because there was a conceptual error in how the model was trained.

Even though a train and test split was performed, the model was still trained on the whole dataset – resulting high accuracy when performing a test accuracy.

**Random Forest – Training with dataset scaling**

Before trying to optimize our results using cross validation techniques, we try to scale our dataset.

Feature scaling is a technique used to standardize the range of features or variables in a dataset. The goal of feature scaling is to ensure that each feature contributes equally to the analysis, as features with larger ranges can dominate the analysis.

To do so, the following code is implemented:

A screenshot of a computer program

Description automatically generated with low confidence

*Figure 29. Random Forest training with scaling*

And a general regressor is created and fitted:

A picture containing text, screenshot, font

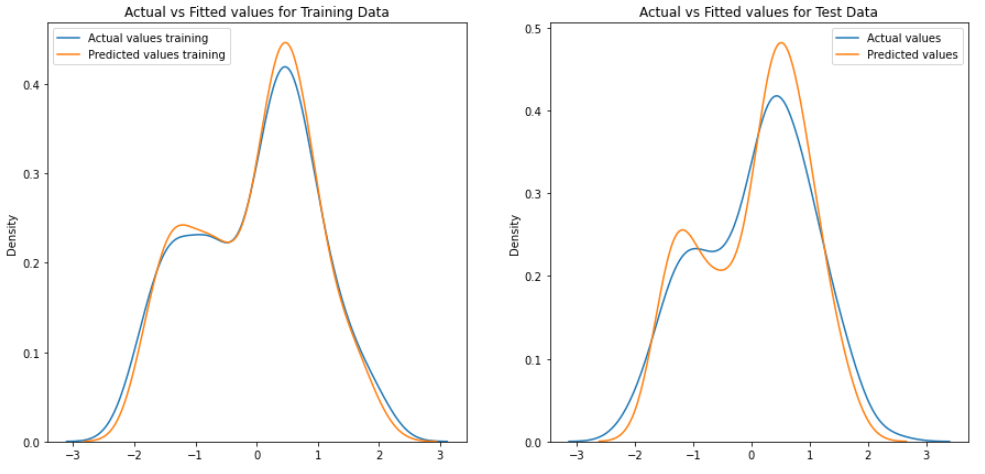
Description automatically generated

*Figure 30. Random Forest scoring with scaling*

As we can see from the screenshot the following scores are obtained:

* R2 Training score 0.984
* R2 Testing score 0.905
* MSE Training score 0.016
* MSE Testing score 0.086

Also, we can visualize how the model is behaving when plotting the actual and predicted values for both test and train:



*Figure 31. Scaled Random Forest score visualization*

If we compare to previous run without scaling, R² results are not improving that much as we can still observe overfitting.

On the other hand, scaling makes the MSE scoring below 0. This is due to the fact that we are now computing the squared root on the average difference of values that are ranging from 0 to 1.

**Random Forest – GridSearchCV**

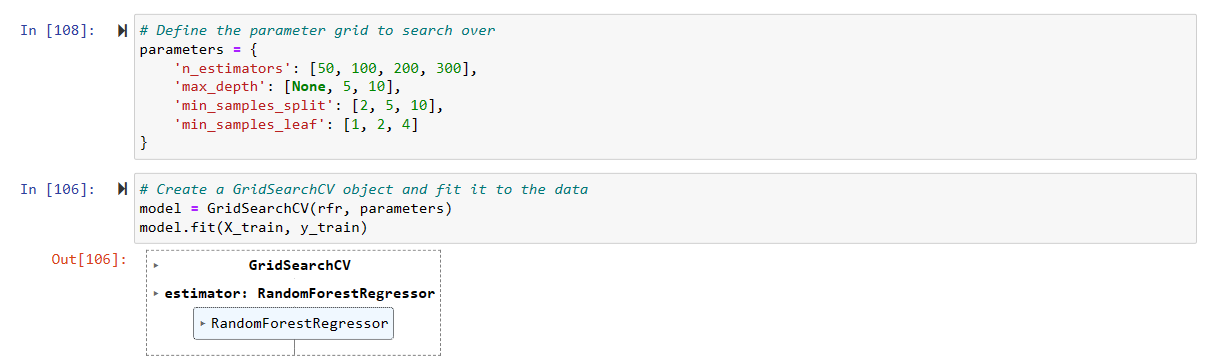
We now try to improve our model, hoping to increase test score while aligning it closer to the training score. To do so, we will use GridSearchCV.

Cross-validation is a technique used in machine learning to assess the performance of a model and to avoid overfitting. GridSearchCV uses cross-validation to evaluate the performance of the model for each combination of hyperparameters and to select the best hyperparameters that result in the highest performance.

Cross-validation works by splitting the data into multiple subsets or folds. We will use 5-fold cross-validation, in this way the data is split into 5 subsets of equal size. Then, the model is trained on 4 of the subsets and evaluated on the remaining subset. This process is repeated 5 times, with each subset being used as the evaluation set once. The performance of the model is then averaged over the 5 folds to get an estimate of the model's performance on new, unseen data.

The hyperparameters we are trying to tune are:

* n\_estimators, the number of trees in the forest
* max\_depth, the maximum depth of each tree
* min\_samples\_split, the minimum number of samples required to split an internal node
* min\_samples\_leaf, the minimum number of samples required to be at a leaf node.



*Figure 32. Random Forest parameters to tune*

Once the model is fitted, the following best parameters are identified:

A screenshot of a computer code

Description automatically generated with low confidence

*Figure 33. Random Forest best parameters*

Produced scores are:

* R2 Training score 0.986
* R2 Testing score 0.907
* MSE Training score 0.015
* MSE Testing score 0.084

And the following plot is produced as before:

A picture containing plot, diagram, line, text

Description automatically generated

*Figure 34. Best Random Forest score visualization*

Overall, in below table we can summarize scores we got for all models tested:

A picture containing text, screenshot, font, line

Description automatically generated

*Figure 35. Random Forest score comparison*

**XGBoost – Training with no optimizations**

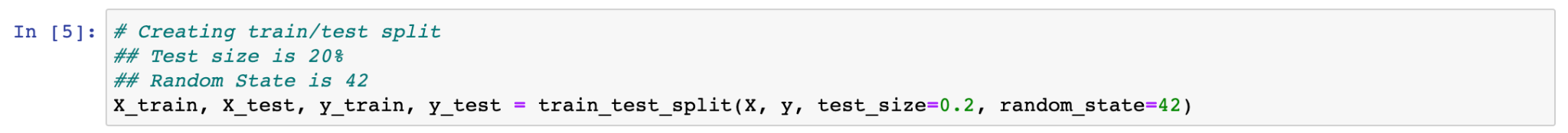
To use XGBoost, firstly, we installed all necessary dependencies used to train and evaluate the model.

A close-up of a message

Description automatically generated with low confidence

*Figure 36. XGBoost libraries*

When initially creating our train, test split, we used a test split size of 20%, this was altered later during our GridSearchCV optimisation.



*Figure 37. XGBoost train test split*

We defined our model and fit the training values. As seen below the model had no hyperparameter tuning. All parameters were set to their default values.

A picture containing text, screenshot, font

Description automatically generated

*Figure 38. Simple XGBoost training*

After training the model, we evaluated the XGBoost’s R2 score, MSE (Mean-Squared Error) and RMSE (Root Mean-Squared Error) and plotted them using Seaborn. These evaluation metrics were chosen as they are the most used for regression models (Agrawal, 2021). Printed below are the evaluations’ values and visualisations.

A screenshot of a computer

Description automatically generated with low confidence

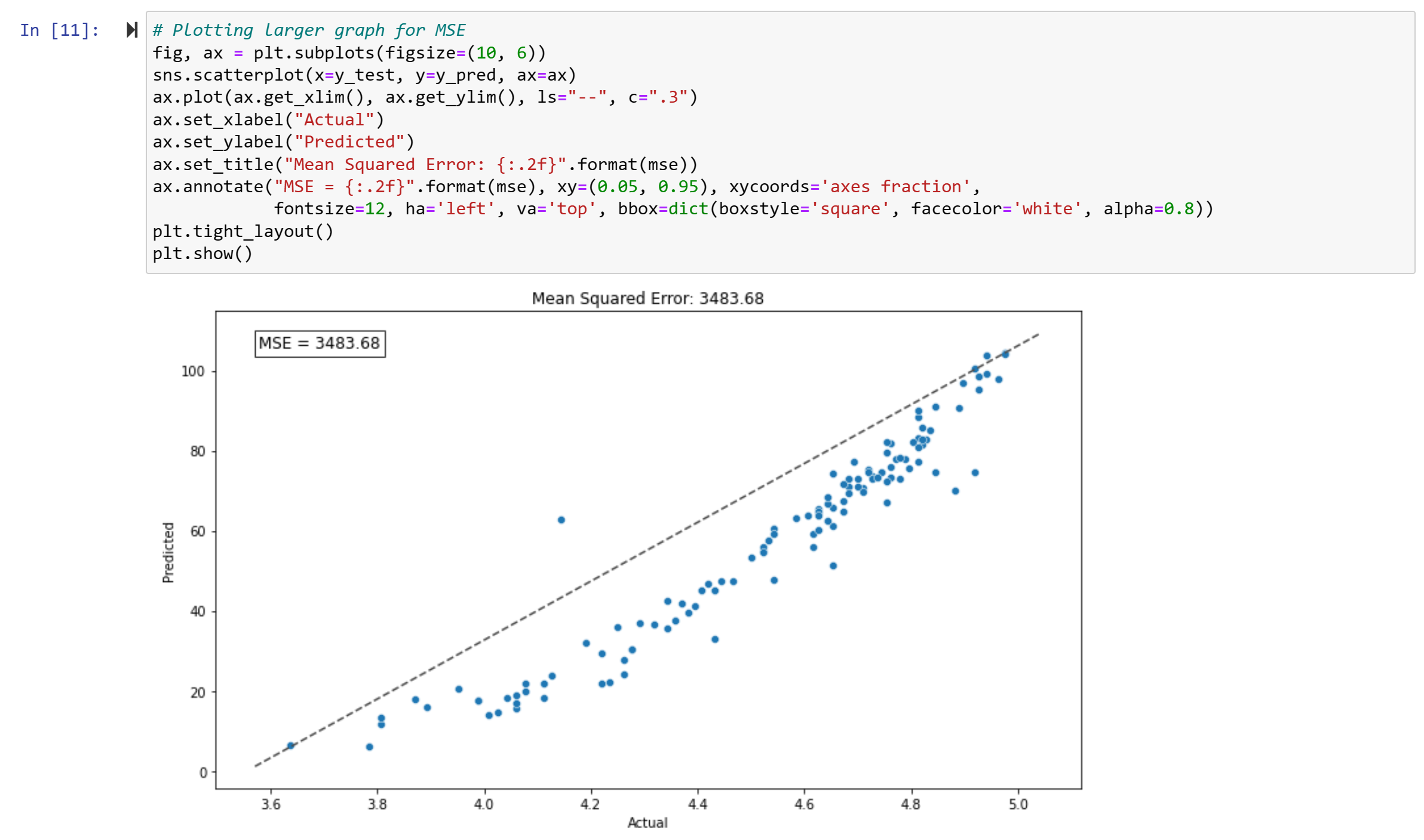
*Figure 39. Simple XGBoost score visualization*

A picture containing text, font, screenshot

Description automatically generated

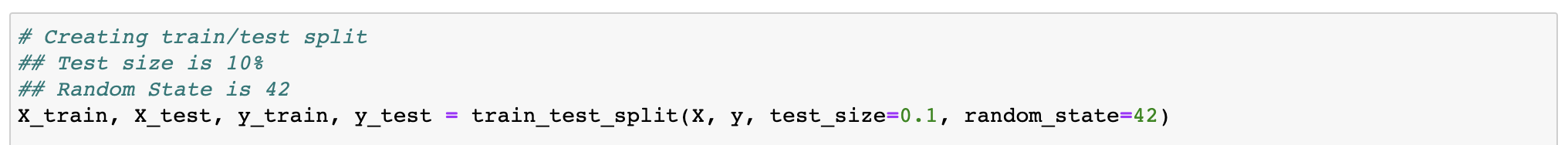
*Figure 40. Simple XGBoost score*

Below we visualised the plotting of our predicted and actual values from the dataset. We had to refit the y\_test values using a label encoder, as the current values would not fit with Seaborn visualisations.



*Figure 41. Simple XGBoost MSE score visualisation*

To determine the best train, test split we attempted to use multiple (0.2, 0.15, 0.1, 0.05) testing values, and move forward with the best-performing result. From the visualisation below comparing the 20% and 10% testing split, we can see that the 10% split performs better with the MSE evaluation and RMSE. The difference in the R2 score is slight but not negligible.



*Figure 42. XGBoost train test split*

A picture containing text, screenshot, diagram, plot

Description automatically generated

*Figure 43. XGBoost comparison with initial model*

**XGBoost – GridSearchCV**

We implemented GridSearchCV to tune hyperparameters to improve the XGBoost Regressor’s performance. The parameters of ‘learning\_rate’, ‘max\_depth’, ‘min\_child\_weight’, ‘subsample’, and ‘colsample\_bytree’, altered the decision trees used in the XGBoost model. ‘n\_estimators’ and ‘gamma’, made no change to the model’s performance, seen in our testing.

A screenshot of a computer

Description automatically generated with low confidence

*Figure 44. XGBoost parameters to tune*

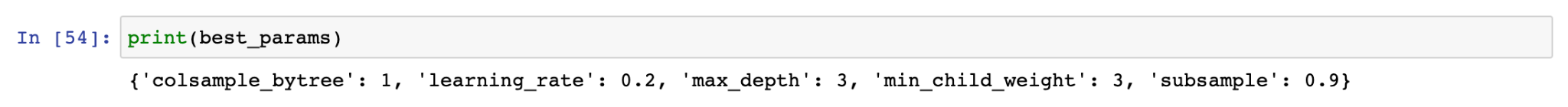
Cross-validation was used to maximise the data being used:

A screenshot of a computer

Description automatically generated with medium confidence

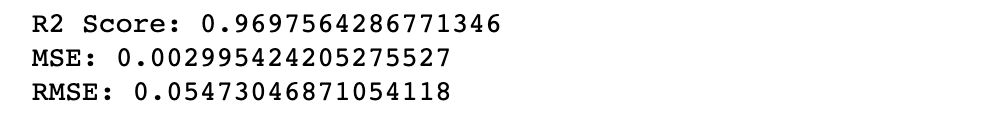
*Figure 45. XGBoost tuning*

Below are the hyperparameters found by GridSearchCV to yield the best results. These would be the parameters we go forward with in the XGBoost Regressor.



*Figure 46. XGBoost GridSearchCV best parameters*

We saw a large increase in the performance of the model after using GridSearchCV.



*Figure 47. Best XGBoost GridSearchCV score*

As seen in the visualisation below, our model was predicting slightly higher values than the actual values from the dataset. Still, the RMSE (as the MSE) score is highly reduced.

A picture containing line, text, plot, screenshot

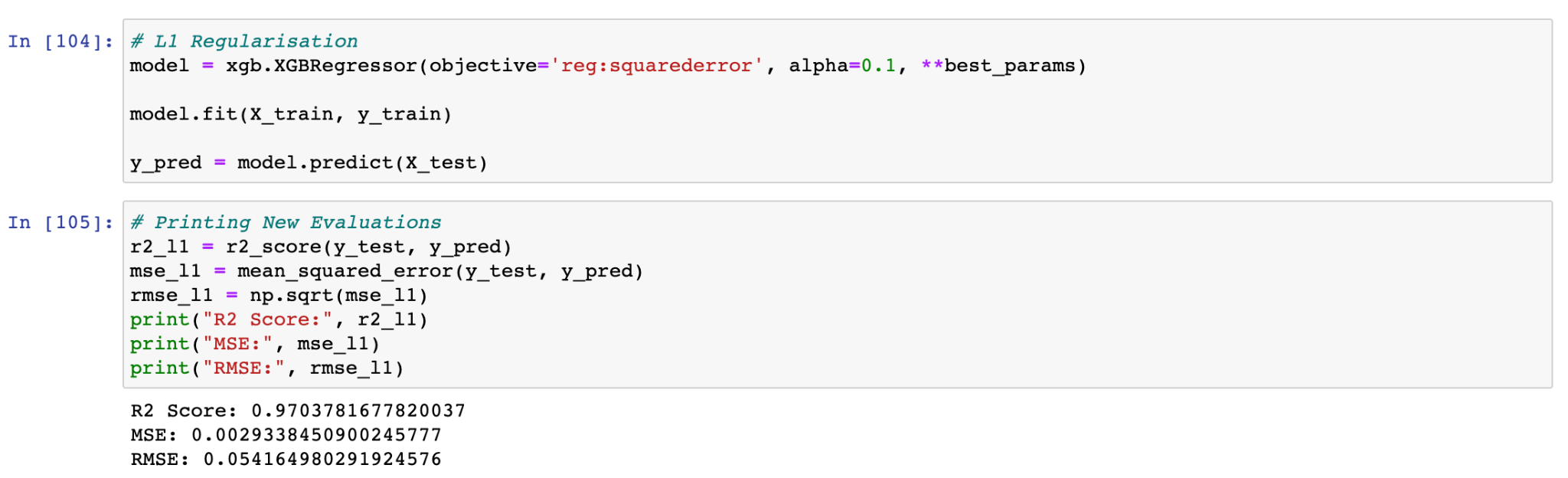
Description automatically generated

*Figure 48. Best XGBoost GridSearchCV RMSE*

**XGBoost – L1 and L2 regularisation**

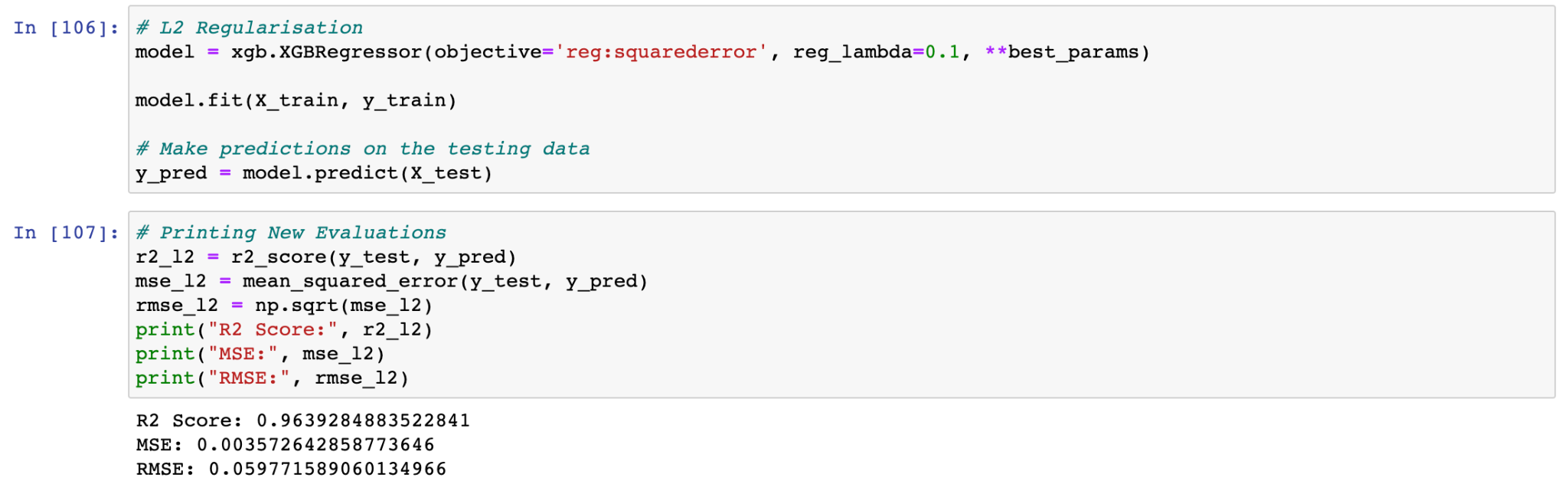
In a final attempt to improve model performance, we used L1 and L2 regularisation.

For L1 Regularisation we added the alpha value to the Regressor. This minorly increased model performance:



*Figure 49. XGBoost l1 optimization*

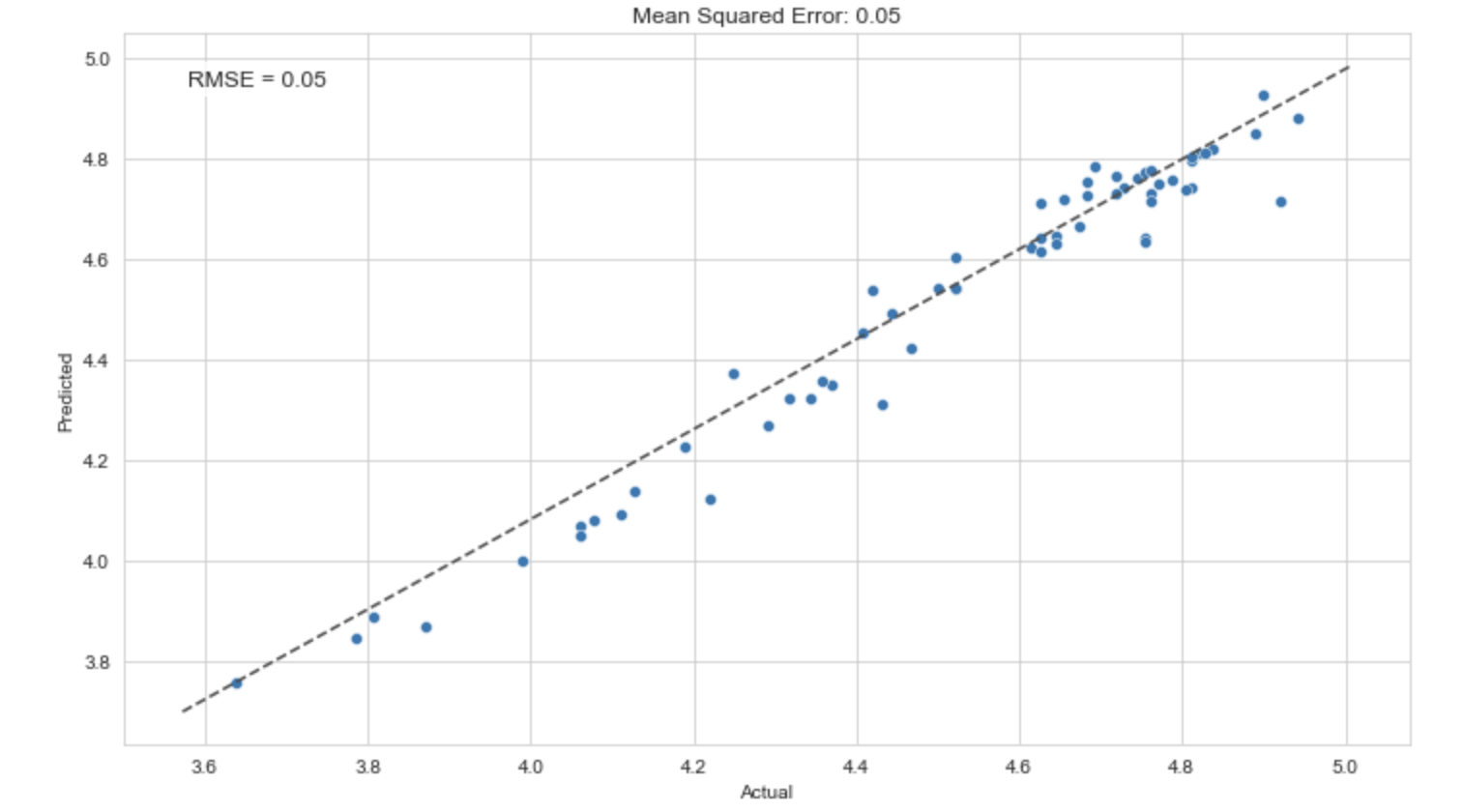
For L2 Regularisation, we added the lambda value. This decreased model performance:



*Figure 50. XGBoost l2 optimization*

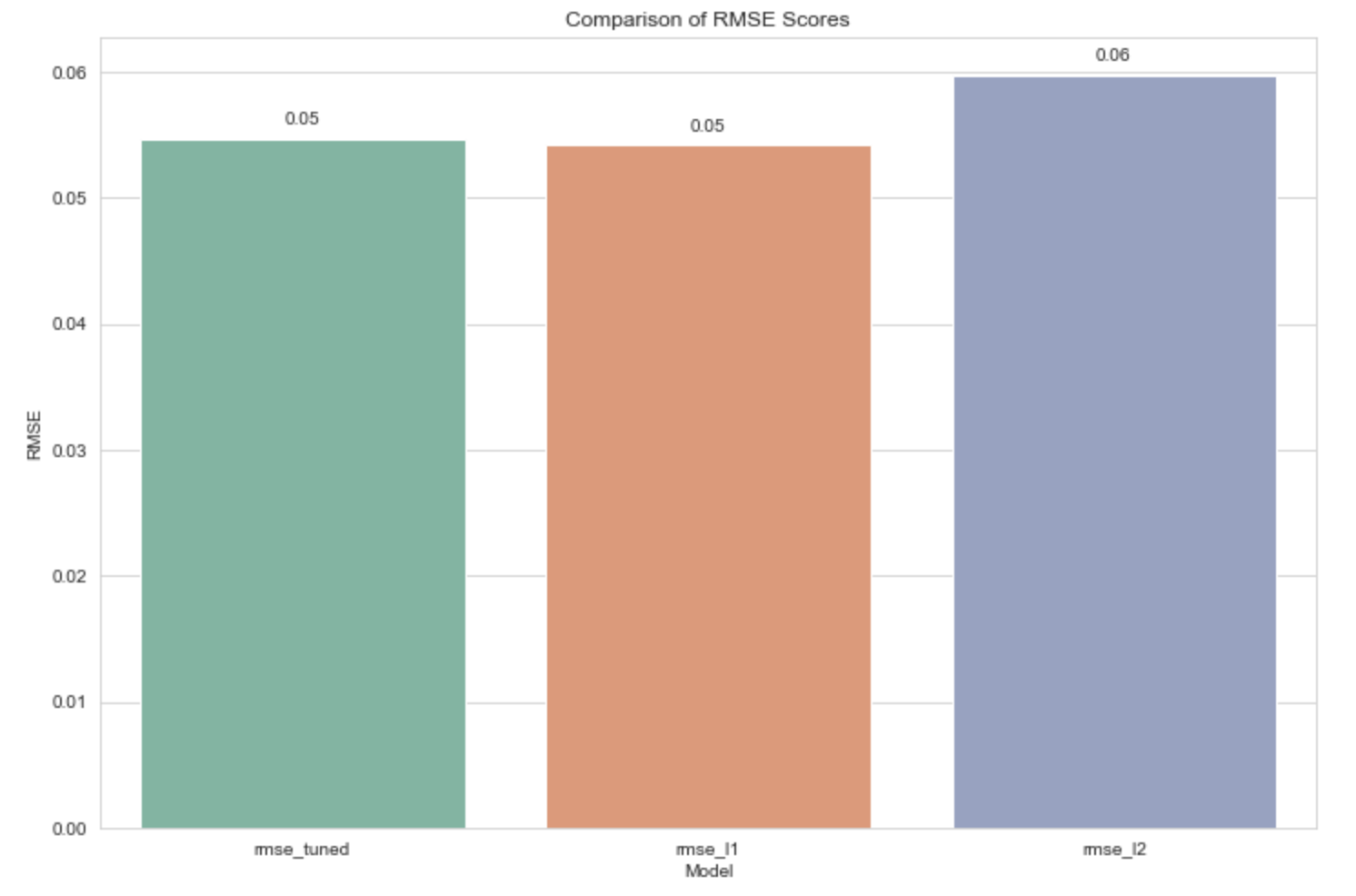
Ending with below scores, we considered this to be a strong performance from the model:

* R2 score 0.964
* MSE score 0.003
* RMSE score 0.059



*Figure 51. XGBoost l1 MSE score*

In the visualisation below you can see the change in our model’s RMSE scores after tuning and regularisation. The untuned model’s RMSE score has been left out as it makes the tuned model’s score unreadable.



*Figure 52. XGBoost models comparison*

**Neural Network – Training with no optimizations**

Neural networks are subset of machine learning algorithm and at the heart of deep learning. They are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. (IBM, 2023)

We initially built a very simple and standard neural network to see how performance would be.

Our starting points are:

* An input layer made of 25 units and an input dimension of 14 (due to the number of features).
* An activation set to “relu” and a kernel initializer set to “he\_uniform” for the input layer.
* An output layer made of 1 unit (due to the regression problem).
* An activation set to “linear” for the output layer
* 100 epoches.
* A “mean\_squared\_error” loss function.
* An SDG optimizer with learning rate 0.01 and momentum 0.9.

The loss function is the only parameter that has been intentionally defined, as:

“MSE loss is the default loss to use for regression problems. Mathematically, it is the preferred loss function under the inference framework of maximum likelihood if the distribution of the target variable is Gaussian. It is the loss function to be evaluated first and only changed if you have a good reason.” (Jason Brownlee, 2019)

We now build and fit our Neural Network after scaling our dataset as done with Random Forest and XGBoost:

A screenshot of a computer code

Description automatically generated with low confidence

*Figure 53. Simple Neural Network training*

Produced scores are:

* R2 Training score 0.755
* R2 Testing score 0.743
* MSE Training score 0.255
* MSE Testing score 0.233

A couple of graphs have also been built to visualize how the loss and MSE both change over time:

A picture containing screenshot, line, plot, diagram

Description automatically generated

*Figure 54. Simple Neural Network loss change over epochs*

A picture containing screenshot, plot, line, text

Description automatically generated

*Figure 55. Simple Neural Network MSE change over epochs*

Also Mean Squared Logarithmic Error Loss and Mean Absolute Error Loss have been tested as loss function on the same exact model, but the results didn’t improve. For this reason, we proceeded with the optimization of the model keeping the MSE as loss function.

**Neural Network – GridSearchCV**

At this point, we want to optimize our model and we will specifically look at the following:

* Activation functions
* Init Mode
* Optimizer
* Learning rate
* Number of neurons
* Number of epochs

For hyperparameters selection Grid search cross-validation (GridSearchCV) is used. *“GridSearchCV process will construct and evaluate one model for each combination of parameters”*. (Jason Brownlee, 2016).

We will refer to this optimized model as “GridSearchCV Model”.

All hyperparameters are first initialized:

A screenshot of a computer

Description automatically generated with low confidence

*Figure 56. Neural Network parameters to tune*

Our strategy involves utilizing the GridSearchCV technique to evaluate a set of hyperparameter combinations multiple times, specifically five times. We will then select the hyperparameter that produces the best results most frequently and keep track of its frequency using a dictionary.

This approach aims to enhance the accuracy of our model and address the issue of overfitting. Our initial focus will be on evaluating the activation function for the input layer.

The following code is used:

A screenshot of a computer code

Description automatically generated with medium confidence

*Figure 57. Neural Network code for hyperparameter tuning*

Most of the GridSearchCV executions returned ‘relu’ as the best activation function:

A screenshot of a computer code

Description automatically generated with low confidence

*Figure 58. Neural Network best activation function*

And the following scores were obtained:

* R2 Training score 0.619
* R2 Testing score 0.618
* MSE Training score 0.395
* MSE Testing score 0.345

Upon reviewing our results, we have observed that we have not made any significant progress in terms of improving scores from our previous model. The reason for this is that we had randomly chosen hyperparameters previously.

However, we are now in the process of systematically identifying the optimal hyperparameters step by step. In the interest of brevity, we will not include the code for each tuning, but we will report on the results we obtain as we tune different parameters. This will allow us to monitor the model's performance as it improves over time.

Step 1 – init\_mode tuning

Best init\_mode: normal

Best init mode scores:

* R2 Training score 0.641
* R2 Testing score 0.654
* MSE Training score 0.373
* MSE Testing score 0.313

Step 2 – optimizer tuning

Best optimizer: Adagrad

Best optimizer scores:

* R2 Training score 0.198
* R2 Testing score 0.179
* MSE Training score 0.832
* MSE Testing score 0.743

NB: score are now worst because no learning rate has been configured

Step 3 – learn\_rate tuning

Best learn\_rate: 0.3

Best learn\_rate scores:

* R2 Training score 0.763
* R2 Testing score 0.720
* MSE Training score 0.246
* MSE Testing score 0.253

Step 4 – Number of neuros tuning

Best neurons: 50

Best neurons scores:

* R2 Training score 0.800
* R2 Testing score 0.802
* MSE Training score 0.207
* MSE Testing score 0.179

Step 5 – Number of epochs tuning

Best epochs: 500

Best epochs scores:

* R2 Training score 0.958
* R2 Testing score 0.923
* MSE Training score 0.043
* MSE Testing score 0.069

Let’s also check how the loss is evolving with the increasing number of epochs:

A picture containing screenshot, plot, line, text

Description automatically generated

*Figure 59. Best Neural Network loss change over epochs*

## Evaluation

The Evaluation phase looks more broadly at which model best meets the business and what to do next. (Nick Hotz, 2022)

Below table summarizes results we got with all models be trained:

|  |  |  |
| --- | --- | --- |
|  | **R2 Score** | **MSE Score** |
| **RF - No optimization** | 0.906 | 64002,00 |
| **RF - Dataset scaling** | 0.905 | 0.086 |
| **RF - GridSearchCV** | 0.907 | 0.084 |
| **XGBoost - No optimization** | -35000.068 | 3483.679 |
| **XGBoost - GridSearchCV** | 0.969 | 0.003 |
| **XGBoost - L1 optimization** | 0.970 | 0.003 |
| **XGBoost - L2 optimization** | 0.964 | 0.003 |
| **NN - No optimization** | 0.743 | 0.233 |
| **NN - GridSearchCV** | 0.923 | 0.069 |

We must specify that we always started training the models performing no optimization.

This is because we wanted to be able to start from the simplest model possible, without scaling the dataset, to prove how results could be improved applying our understanding of the models implemented.

For this reason, it makes no sense to compare the non-optimized models because in some cases (RF and XGBoost) not scaling the dataset resulted in non-acceptable and comparable scoring results.

For the Random Forest model, we observed that cross-validation didn’t improve our results. Our assumption is that the combination of two factors (how the random forest works and the limited dimension of the testing dataset where cross-validation was applied) didn’t allow us to improve the model performance. In fact, running GridSearchCV multiple times resulted in different best hyperparameters selection that, when used to train a new model, didn’t result in a significant score improvement.

For the XGBoost model, the combination of dataset scaling and cross-validation did significantly improve the model outcome. Furthermore, L1 regularization even more improved the model results.

Finally, the neural network we implemented was the most complex when going beyond the non-optimized NN. Given the huge amount of hyperparameters selection, we couldn’t run a single GridSearchCV evaluation across all hyperparameters. This probably resulted in the selection of very good parameters but possibly not the best.

Overall, if looking at the MSE and R2 scores, we can say that the best performing model is the XGBoost with hyperparameter selection using GridSearchCV and L1 regularization. The results obtained are:

* R2 score 0.97
* MSE score 0.003

It is worth nothing that result might be not completely comparable between XGBoost and Neural Network since one was trained using a 20% test split and the other a 30% test split. This might have resulted in an overfitting of the XGBoost model given the small dimension of the dataset we worked on.

## Deployment

A model is not particularly useful unless the customer can access its results. (Nick Hotz, 2022)

This study is providing a step towards the right direction in predicting the efficacy and utility for a particular job offer, considering the country in which the offer is being released.

When deploying the model into production it would return the quality life index of a country given its commodity price features. This index must then be related to the salary offer with a mathematical formula – not described in this paper – to let the employee evaluate if he/she would live with a higher purchasing power that the current situation.

Overall, we believe that the model would be a good starting point when evaluating a new job offer but it is also important to consider how commodity prices might change in short periods of time. If for example, as it is currently happening, a period of high inflation would cause a data drift, meaning that MLOps principles need to be considered if the model must be used into production.

## Reference List

Hotz, N. (2022). *CRISP-DM. Data Science Project Management*. Available at: https://www.datascience-pm.com/crisp-dm-2/ [Accessed: December 27, 2022].

‌OECD data (2022). *Prices - price level indices*. Available at: https://data.oecd.org/price/price-level-indices.htm [Accessed: December 27, 2022].

Eurostat (2022). *Purchasing power parities (PRC\_PPP)*. Available at: https://ec.europa.eu/eurostat/cache/metadata/en/prc\_ppp\_esms.htm [Accessed: December 27, 2022].

Bakshi, C. (2020). *Random Forest Regression. Medium*. Available at: https://levelup.gitconnected.com/random-forest-regression-209c0f354c84 [Accessed: January 2, 2023].

IBM (2023). *What are Neural Networks?. IBM*. Available at: www.ibm.com. Available at: https://www.ibm.com/topics/neural-networks [Accessed: April 27, 2023].‌

Brownlee, J. (2019). How to Choose Loss Functions When Training Deep Learning Neural Networks. Machine Learning Mastery. Available at: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ [Accessed: April 27, 2023].

Agrawal, R. (2021). Know The Best Evaluation Metrics for Your Regression Model ! [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/ [Accessed 29 Apr. 2023].

‌