# Configuration Manual

MSc Research Project
Data Analytics

Conor Moody
Student ID: 21201765

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

National College of Ireland

| | |
|---|---|
| **Student Name:** | Conor Moody |
| **Student ID:** | 21201765 |
| **Programme:** | Data Analytics |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Jorge Basilio |
| **Submission Due Date:** | 14/08/2023 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 1,085 |
| **Page Count:** | 9 |

| | |
|---|---|
| **Signature:** | Conor Moody |
| **Date:** | 18th September 2023 |

## PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Conor Moody
21201765

## 1 Introduction

The following document is to be used as a guide to set up the environment needed to run the three models that were built to predict the Research & Development (R&D) spend for the Annual Business Survey of Economic Impact (ABSEI) which is coordinated by the Department of Enterprise, Trade & Employment. This manual will cover the environment setup as well as the work needed to perform Data Transformation and Data Preparation, as well as the Implementation of the Recurrent Neural Network, Feed Forward Neural Network and Linear Regression models.

## 2 Dataset

The ABSEI dataset used for this analysis was a large CSV file containing restricted information on Irish businesses. It was made available for this analysis but unfortunately it was only accessible on a designated PC in the Department of Enterprise, Trade & Employment. As such, the raw data used is not accessible for running outside of the Department.

An advantage of the dataset being made available for this analysis was that previously unavailable variables on R&D were accessible for the purposes of this project. This allowed for the ability to build a model to predict the R&D spend for Irish companies.

The initial dataset was made up of 10,856 companies with 1,059 variables for each of them. These 1,059 variables covered the data for each year that the survey was offered (2000 to 2022 inclusive) for 185 questions.

## 3 Environment Setup

### 3.1 Hardware Requirements

The following are the recommended Hardware Requirements needed for the development of this project:

- Processor: Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz 2.11 GHz
- Installed RAM: 16.0 GB
- Storage: 256GB SSD/1TB HDD Operating System.
- : System Type: 64-bit operating system, x64-based Operating System.

## 3.2 Software Requirements

This project was run using the following software:

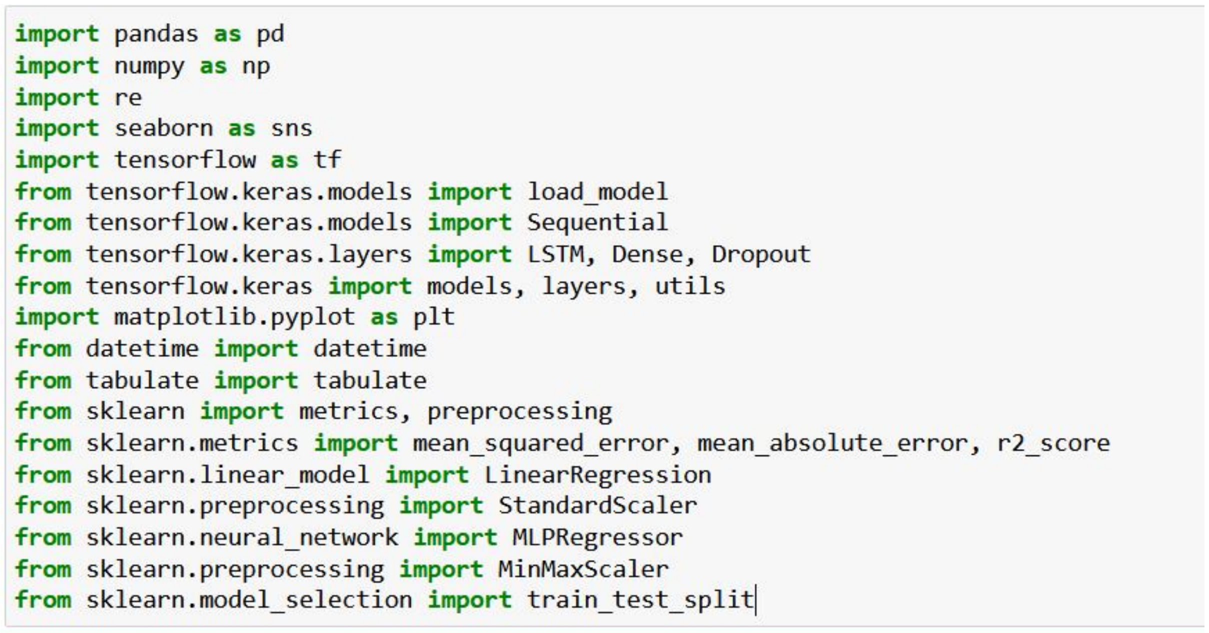
- Python 3.11 - Installed on machine
- Juypter Notebook - Installed on machine
- Overleaf - Online

## 3.3 Libraries

Python has a number of libraries that it can harness in order to improve the processing efficiencies. In particular, the 4 main libraries used in the project were:

- Pandas
- Numpy
- Tensorflow
- Sklearn

Within those 4 libraries, there were a number of packages used. There was also other libraries that were needed at various stage, and all of the packages in Figure 1 below were used during the project:

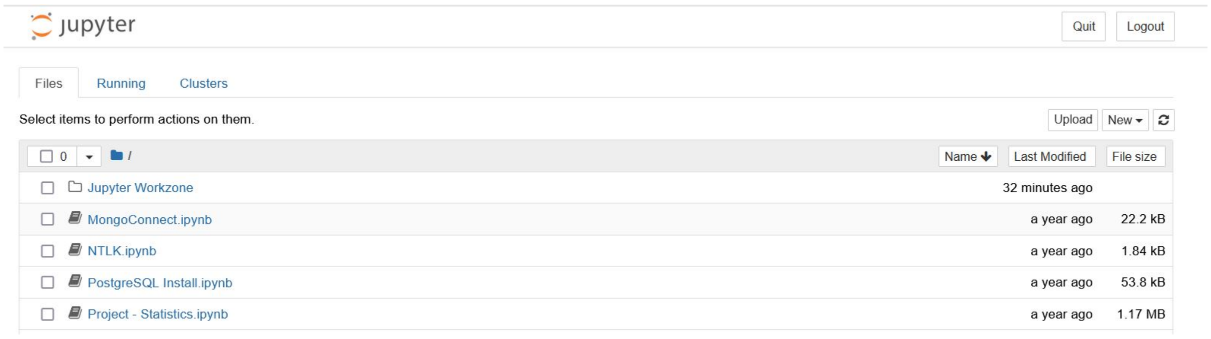


```python
import pandas as pd
import numpy as np
import re
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras import models, layers, utils
import matplotlib.pyplot as plt
from datetime import datetime
from tabulate import tabulate
from sklearn import metrics, preprocessing
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

Figure 1: Imported Packages

## 3.4 Juypter Notebook

Jupyter Notebook was used to edit the Python code. Jupyter Notebook (shown in Figure 2 below) is a web-based Integrated Development Environment (IDE) and it was the most popular IDE when data analysts were surveyed in 2022 (Shafl 2023).



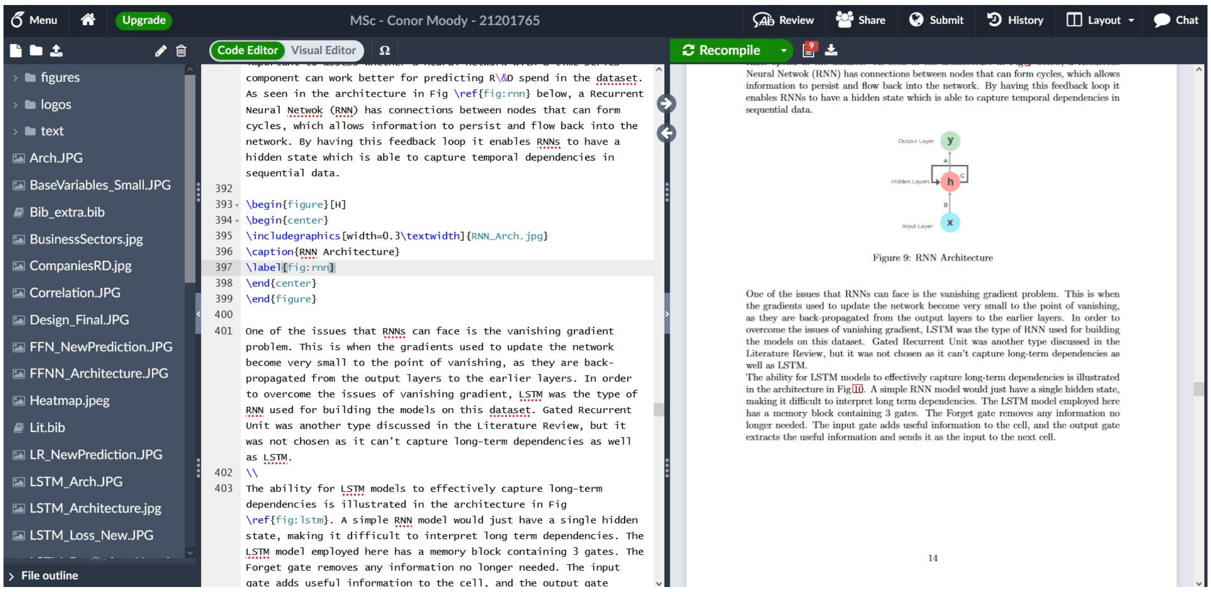
Select items to perform actions on them.

| Name | Last Modified | File size |
|---|---|---|
| Jupyter Workzone | 32 minutes ago | |
| MongoConnect.ipynb | a year ago | 22.2 kB |
| NTLK.ipynb | a year ago | 1.84 kB |
| PostgreSQL Install.ipynb | a year ago | 53.8 kB |
| Project - Statistics.ipynb | a year ago | 1.17 MB |

Figure 2: Jupyter Notebook

## 3.5 Overleaf

The project report was written in Overleaf (shown in Figure 3 below), which is an online LaTeX editor. This was chosen due to the effective ability of LaTeX to handle images and tables for reports, as well as how it is much easier to organise the structure and references of long documents.



```
component can work better for predicting R\&D spend in the dataset.
As seen in the architecture in Fig \ref{fig:rnn} below, a Recurrent
Neural Netwok (RNN) has connections between nodes that can form
cycles, which allows information to persist and flow back into the
network. By having this feedback loop it enables RNNs to have a
hidden state which is able to capture temporal dependencies in
sequential data.

\begin{figure}[H]
\begin{center}
\includegraphics[width=0.3\textwidth]{RNN_Arch.jpg}
\caption{RNN Architecture}
\label{fig:rnn}
\end{center}
\end{figure}

One of the issues that RNNs can face is the vanishing gradient
problem. This is when the gradients used to update the network
become very small to the point of vanishing, as they are back-
propagated from the output layers to the earlier layers. In order
to overcome the issues of vanishing gradient, LSTM was the type of
RNN used for building the models on this dataset. Gated Recurrent
Unit was another type discussed in the Literature Review, but it
was not chosen as it can't capture long-term dependencies as well
as LSTM.
\\
The ability for LSTM models to effectively capture long-term
dependencies is illustrated in the architecture in Fig
\ref{fig:lstm}. A simple RNN model would just have a single hidden
state, making it difficult to interpret long term dependencies. The
LSTM model employed here has a memory block containing 3 gates. The
Forget gate removes any information no longer needed. The input
gate adds useful information to the cell, and the output gate
```

Neural Netwok (RNN) has connections between nodes that can form cycles, which allows information to persist and flow back into the network. By having this feedback loop it enables RNNs to have a hidden state which is able to capture temporal dependencies in sequential data.

Figure 9: RNN Architecture

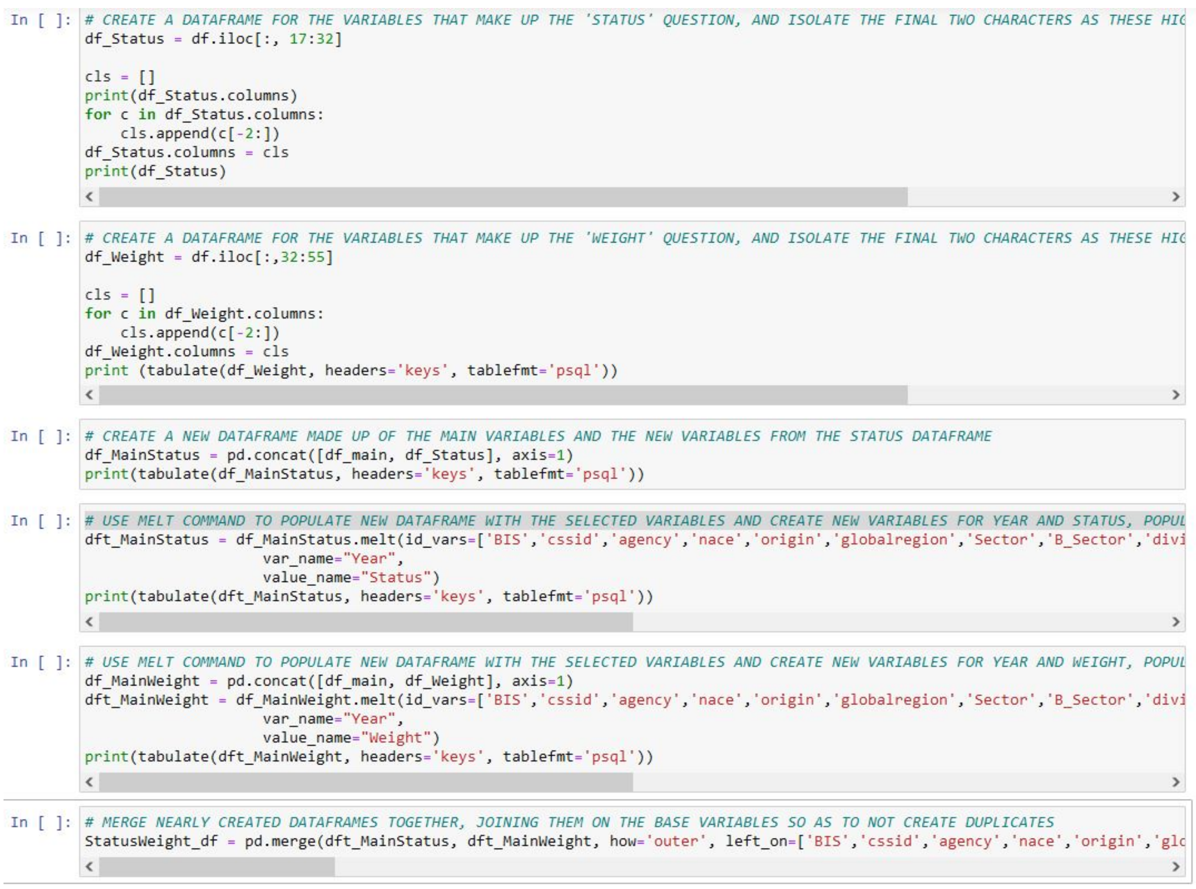
One of the issues that RNNs can face is the vanishing gradient problem. This is when the gradients used to update the network become very small to the point of vanishing, as they are back-propagated from the output layers to the earlier layers. In order to overcome the issues of vanishing gradient, LSTM was the type of RNN used for building the models on this dataset. Gated Recurrent Unit was another type discussed in the Literature Review, but it was not chosen as it can't capture long-term dependencies as well as LSTM.
The ability for LSTM models to effectively capture long-term dependencies is illustrated in the architecture in Fig 10. A simple RNN model would just have a single hidden state, making it difficult to interpret long term dependencies. The LSTM model employed here has a memory block containing 3 gates. The Forget gate removes any information no longer needed. The input gate adds useful information to the cell, and the output gate extracts the useful information and sends it as the input to the next cell.

14

Figure 3: Overleaf

# 4 Data Transformation

The first Juypter Notebook file to be run is 'Data Transformation.ipynb'. This file takes the original raw data, and performs the transformation using the Melt function in Python, to reduce the dimensionality of the dataset and ensure that there is one variable for every question in the survey. By performing this action on every question where there are multiple years' of data, the dataset is reduced to 185 variables, but there are now over 249,000 rows. This allows for the performance of time series analysis on the data, and a code snippet for this section is shown in Figure 4 below.



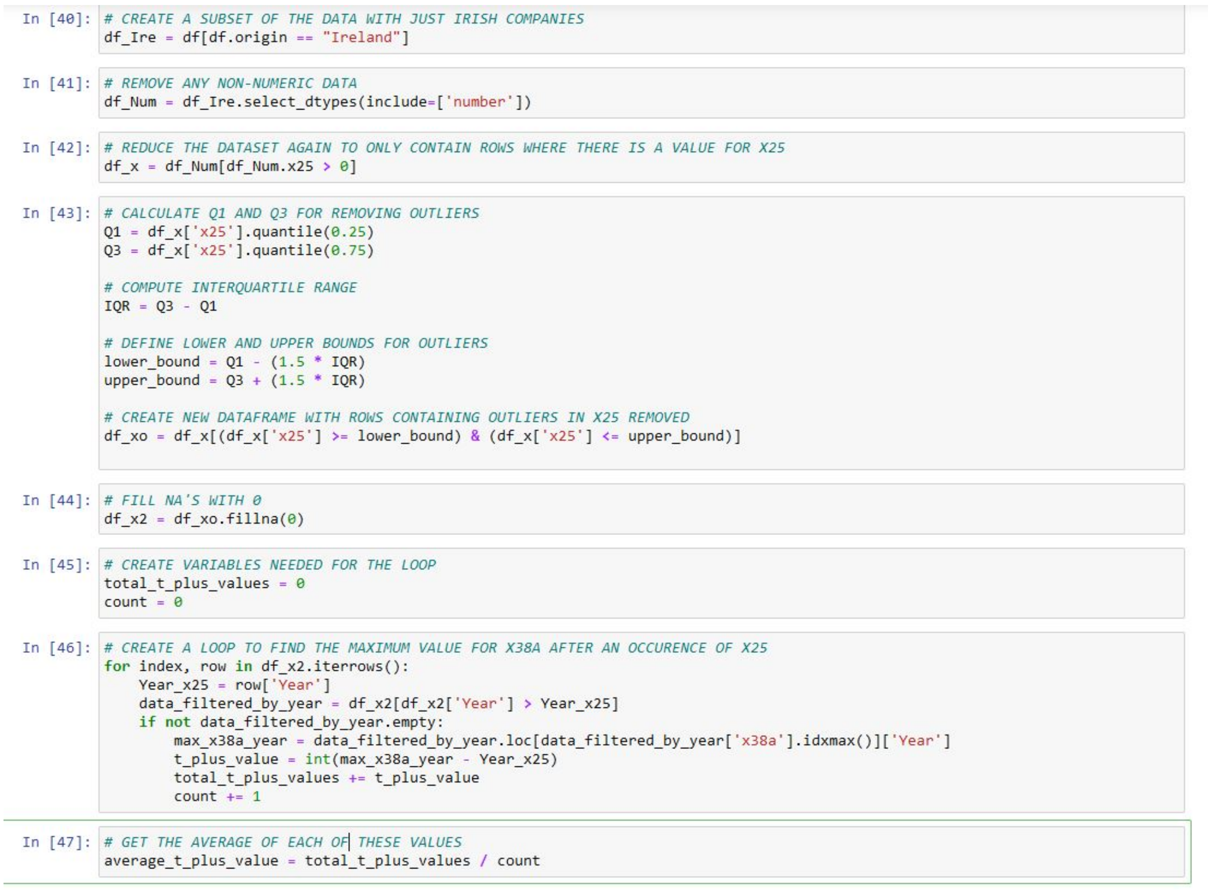
```
In [ ]: # CREATE A DATAFRAME FOR THE VARIABLES THAT MAKE UP THE 'STATUS' QUESTION, AND ISOLATE THE FINAL TWO CHARACTERS AS THESE HIG
        df_Status = df.iloc[:, 17:32]

        cls = []
        print(df_Status.columns)
        for c in df_Status.columns:
            cls.append(c[-2:])
        df_Status.columns = cls
        print(df_Status)

In [ ]: # CREATE A DATAFRAME FOR THE VARIABLES THAT MAKE UP THE 'WEIGHT' QUESTION, AND ISOLATE THE FINAL TWO CHARACTERS AS THESE HIG
        df_Weight = df.iloc[:,32:55]

        cls = []
        for c in df_Weight.columns:
            cls.append(c[-2:])
        df_Weight.columns = cls
        print (tabulate(df_Weight, headers='keys', tablefmt='psql'))

In [ ]: # CREATE A NEW DATAFRAME MADE UP OF THE MAIN VARIABLES AND THE NEW VARIABLES FROM THE STATUS DATAFRAME
        df_MainStatus = pd.concat([df_main, df_Status], axis=1)
        print(tabulate(df_MainStatus, headers='keys', tablefmt='psql'))

In [ ]: # USE MELT COMMAND TO POPULATE NEW DATAFRAME WITH THE SELECTED VARIABLES AND CREATE NEW VARIABLES FOR YEAR AND STATUS, POPUL
        dft_MainStatus = df_MainStatus.melt(id_vars=['BIS','cssid','agency','nace','origin','globalregion','Sector','B_Sector','divi
                             var_name="Year",
                             value_name="Status")
        print(tabulate(dft_MainStatus, headers='keys', tablefmt='psql'))

In [ ]: # USE MELT COMMAND TO POPULATE NEW DATAFRAME WITH THE SELECTED VARIABLES AND CREATE NEW VARIABLES FOR YEAR AND WEIGHT, POPUL
        df_MainWeight = pd.concat([df_main, df_Weight], axis=1)
        dft_MainWeight = df_MainWeight.melt(id_vars=['BIS','cssid','agency','nace','origin','globalregion','Sector','B_Sector','divi
                             var_name="Year",
                             value_name="Weight")
        print(tabulate(dft_MainWeight, headers='keys', tablefmt='psql'))

In [ ]: # MERGE NEARLY CREATED DATAFRAMES TOGETHER, JOINING THEM ON THE BASE VARIABLES SO AS TO NOT CREATE DUPLICATES
        StatusWeight_df = pd.merge(dft_MainStatus, dft_MainWeight, how='outer', left_on=['BIS','cssid','agency','nace','origin','glo
```

Figure 4: Data Transformation

## 5 Data Preparation

Once the data is transformed, the 'Data Preparation.ipynb' notebook needs to be run next, and it is the process that worked on getting the data ready for modelling. The dataset was reduced down to just Irish companies. Any non-numeric variables were also removed. In order to properly predict a value for x25, it was important to remove any records where no value for x25 was recorded. This reduced the number of records to just over 29,500. Outliers were then removed as well, as they could overly impact the modelling.

Another process that took place in the Data Preparation phase was to calculate the appropriate number of years between R&D and the resulting New Product Sales ('x38a), and move the values for x38a to the same year that the R&D took place. Figure 5 below shows some of this work in the notebook.



```
In [40]: # CREATE A SUBSET OF THE DATA WITH JUST IRISH COMPANIES
         df_Ire = df[df.origin == "Ireland"]

In [41]: # REMOVE ANY NON-NUMERIC DATA
         df_Num = df_Ire.select_dtypes(include=['number'])

In [42]: # REDUCE THE DATASET AGAIN TO ONLY CONTAIN ROWS WHERE THERE IS A VALUE FOR X25
         df_x = df_Num[df_Num.x25 > 0]

In [43]: # CALCULATE Q1 AND Q3 FOR REMOVING OUTLIERS
         Q1 = df_x['x25'].quantile(0.25)
         Q3 = df_x['x25'].quantile(0.75)

         # COMPUTE INTERQUARTILE RANGE
         IQR = Q3 - Q1

         # DEFINE LOWER AND UPPER BOUNDS FOR OUTLIERS
         lower_bound = Q1 - (1.5 * IQR)
         upper_bound = Q3 + (1.5 * IQR)

         # CREATE NEW DATAFRAME WITH ROWS CONTAINING OUTLIERS IN X25 REMOVED
         df_xo = df_x[(df_x['x25'] >= lower_bound) & (df_x['x25'] <= upper_bound)]

In [44]: # FILL NA'S WITH 0
         df_x2 = df_xo.fillna(0)

In [45]: # CREATE VARIABLES NEEDED FOR THE LOOP
         total_t_plus_values = 0
         count = 0

In [46]: # CREATE A LOOP TO FIND THE MAXIMUM VALUE FOR X38A AFTER AN OCCURENCE OF X25
         for index, row in df_x2.iterrows():
             Year_x25 = row['Year']
             data_filtered_by_year = df_x2[df_x2['Year'] > Year_x25]
             if not data_filtered_by_year.empty:
                 max_x38a_year = data_filtered_by_year.loc[data_filtered_by_year['x38a'].idxmax()]['Year']
                 t_plus_value = int(max_x38a_year - Year_x25)
                 total_t_plus_values += t_plus_value
                 count += 1

In [47]: # GET THE AVERAGE OF EACH OF THESE VALUES
         average_t_plus_value = total_t_plus_values / count
```

Figure 5: Data Preparation

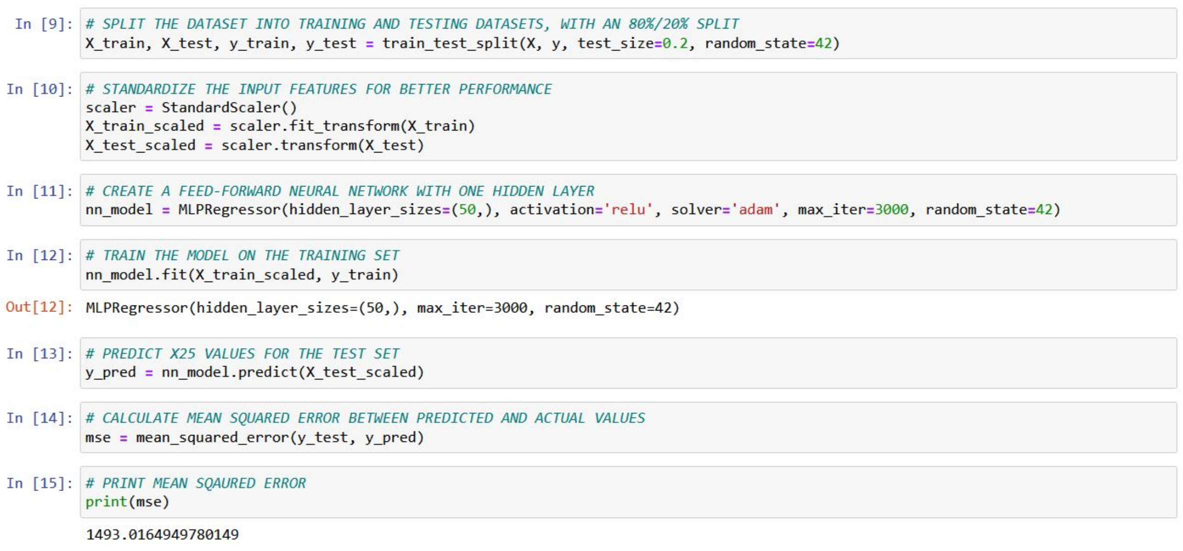
# 6 Implementation

The output of the Data Preparation notebook is a file called 'Model_Data.xlsx', and this file can be run on each of the models independently.

## 6.1 Feed Forward Neural Network

The code in Figure 6 below shows the production of the Feed Forward Neural Network. Once the dataset is split, the input features are standardized. The model is built with one hidden layer with 50 neurons. The ReLU activation function is used as well as the Adam optimization algorithm. The training model is set to 3000 iterations and a random seed for reproducibility is set.

Once the model is trained, x25 values are predicted for the test set, and the results are evaluated.



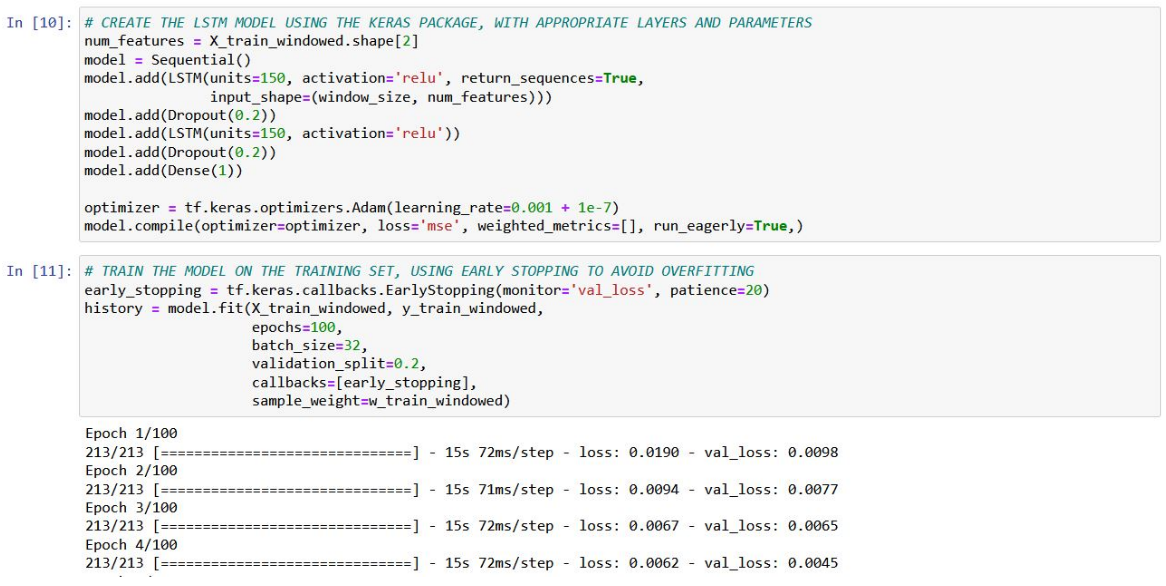
```
In [9]:  # SPLIT THE DATASET INTO TRAINING AND TESTING DATASETS, WITH AN 80%/20% SPLIT
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [10]: # STANDARDIZE THE INPUT FEATURES FOR BETTER PERFORMANCE
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

In [11]: # CREATE A FEED-FORWARD NEURAL NETWORK WITH ONE HIDDEN LAYER
         nn_model = MLPRegressor(hidden_layer_sizes=(50,), activation='relu', solver='adam', max_iter=3000, random_state=42)

In [12]: # TRAIN THE MODEL ON THE TRAINING SET
         nn_model.fit(X_train_scaled, y_train)

Out[12]: MLPRegressor(hidden_layer_sizes=(50,), max_iter=3000, random_state=42)

In [13]: # PREDICT X25 VALUES FOR THE TEST SET
         y_pred = nn_model.predict(X_test_scaled)

In [14]: # CALCULATE MEAN SQUARED ERROR BETWEEN PREDICTED AND ACTUAL VALUES
         mse = mean_squared_error(y_test, y_pred)

In [15]: # PRINT MEAN SQAURED ERROR
         print(mse)

         1493.0164949780149
```

Figure 6: Feed Forward Neural Network Code

## 6.2 Recurrent Neural Network with Long Short-Term Memory

The code below in Figure 7 shows the creation of the Recurrent Neural Network model. It uses the ReLU activation function, and the model is run on a previously set window size. The window size dictates the number of time steps (in this case the number of years) that the model will be trained on. Dropouts are used to prevent overfitting, and a Dense (fully connected) layer with one neuron for regression output is also included.

Early Stopping is included to stop the model when validation loss has stopped improving. The model is then trained, with epochs (full passes through the dataset) set to be 100. Early stopping may mean this is not reached.

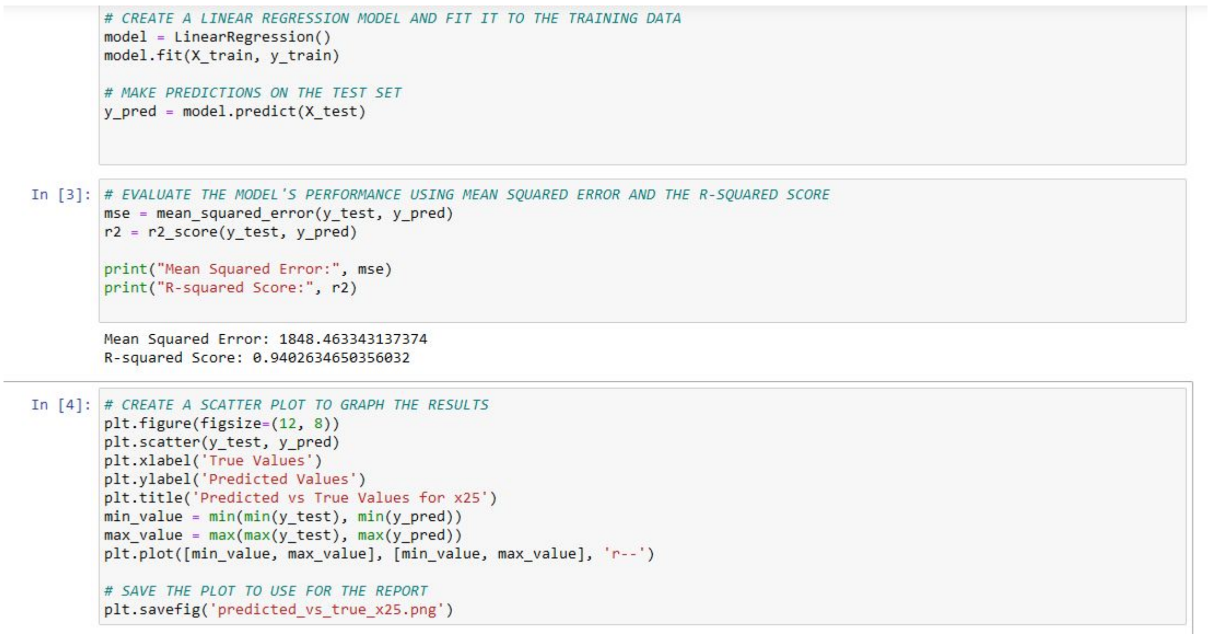


```
In [10]: # CREATE THE LSTM MODEL USING THE KERAS PACKAGE, WITH APPROPRIATE LAYERS AND PARAMETERS
         num_features = X_train_windowed.shape[2]
         model = Sequential()
         model.add(LSTM(units=150, activation='relu', return_sequences=True,
                        input_shape=(window_size, num_features)))
         model.add(Dropout(0.2))
         model.add(LSTM(units=150, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(1))

         optimizer = tf.keras.optimizers.Adam(learning_rate=0.001 + 1e-7)
         model.compile(optimizer=optimizer, loss='mse', weighted_metrics=[], run_eagerly=True,)

In [11]: # TRAIN THE MODEL ON THE TRAINING SET, USING EARLY STOPPING TO AVOID OVERFITTING
         early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)
         history = model.fit(X_train_windowed, y_train_windowed,
                             epochs=100,
                             batch_size=32,
                             validation_split=0.2,
                             callbacks=[early_stopping],
                             sample_weight=w_train_windowed)

         Epoch 1/100
         213/213 [==============================] - 15s 72ms/step - loss: 0.0190 - val_loss: 0.0098
         Epoch 2/100
         213/213 [==============================] - 15s 71ms/step - loss: 0.0094 - val_loss: 0.0077
         Epoch 3/100
         213/213 [==============================] - 15s 72ms/step - loss: 0.0067 - val_loss: 0.0065
         Epoch 4/100
         213/213 [==============================] - 15s 72ms/step - loss: 0.0062 - val_loss: 0.0045
```

Figure 7: Recurrent Neural Network Code

## 6.3 Linear Regression

The Linear Regression model was relatively simplistic by comparison to the other two, and is shown in the code snippet in Figure 8. The Linear Regression function was used to fit the training and test datasets, and the results were then evaluated.



```
# CREATE A LINEAR REGRESSION MODEL AND FIT IT TO THE TRAINING DATA
model = LinearRegression()
model.fit(X_train, y_train)

# MAKE PREDICTIONS ON THE TEST SET
y_pred = model.predict(X_test)
```

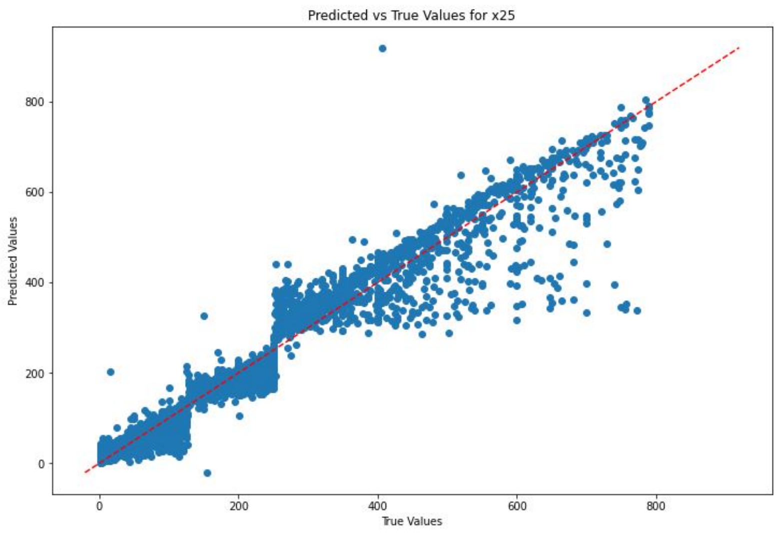
```
In [3]: # EVALUATE THE MODEL'S PERFORMANCE USING MEAN SQUARED ERROR AND THE R-SQUARED SCORE
        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        print("Mean Squared Error:", mse)
        print("R-squared Score:", r2)

        Mean Squared Error: 1848.463343137374
        R-squared Score: 0.9402634650356032
```

```
In [4]: # CREATE A SCATTER PLOT TO GRAPH THE RESULTS
        plt.figure(figsize=(12, 8))
        plt.scatter(y_test, y_pred)
        plt.xlabel('True Values')
        plt.ylabel('Predicted Values')
        plt.title('Predicted vs True Values for x25')
        min_value = min(min(y_test), min(y_pred))
        max_value = max(max(y_test), max(y_pred))
        plt.plot([min_value, max_value], [min_value, max_value], 'r--')

        # SAVE THE PLOT TO USE FOR THE REPORT
        plt.savefig('predicted_vs_true_x25.png')
```

Figure 8: Linear Regression Code

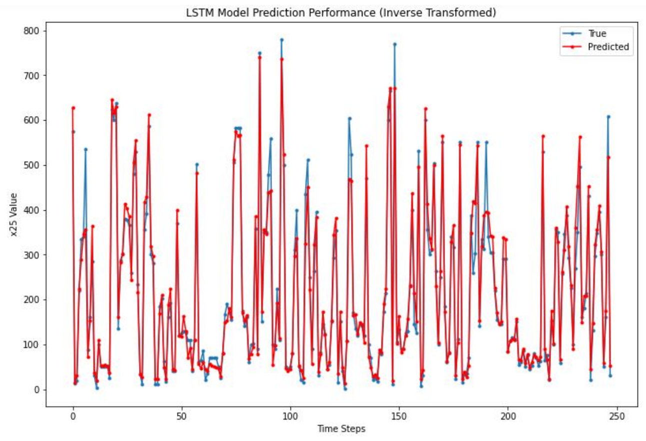
# 7 Evaluation

Figure's 9, 10 and 11 below show the fitting of the predicted values to the actual values in each model. Important metrics such as $R^2$, Mean Squared Error, Mean Absolute Error and Root Mean Squared Error are also calculated in order to evaluate the results of each model.



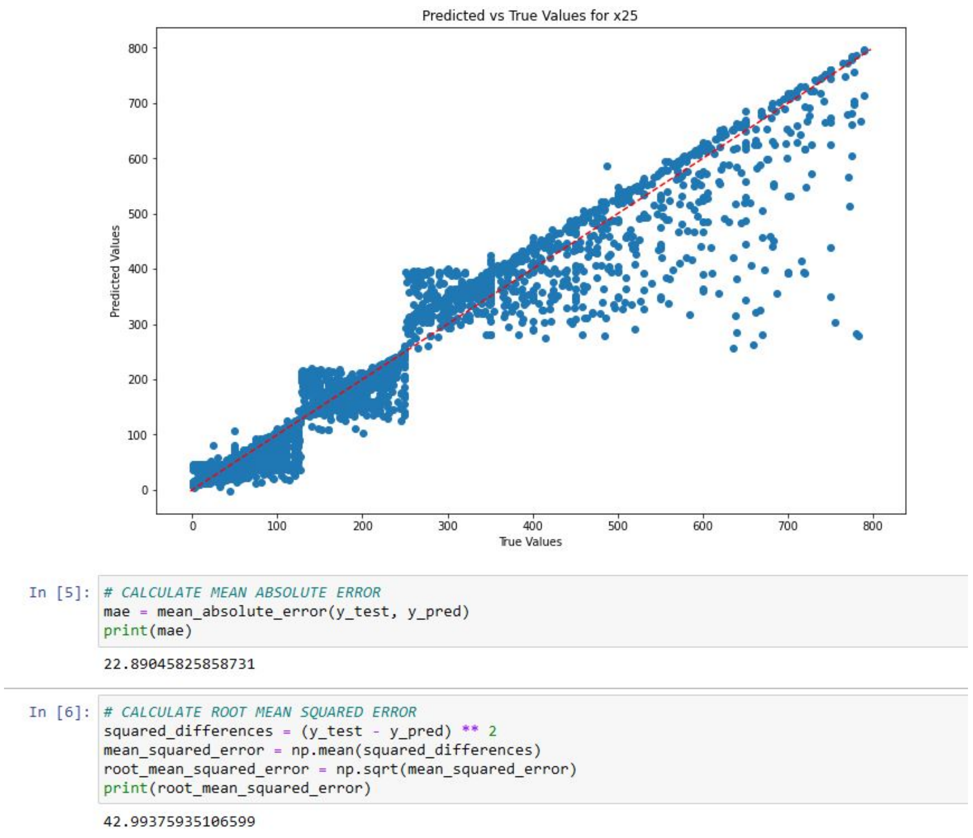
```
In [19]: # CALCULATE AND PRINT THE R-SQUARED VALUE
         r2 = r2_score(y_test, y_pred)
         print(f'R-squared: {r2}')

         R-squared: 0.9516690543420255

In [20]: # CALCULATE ROOT MEAN SQUARED ERROR
         squared_differences = (y_test - y_pred) ** 2
         mean_squared_error = np.mean(squared_differences)
         root_mean_squared_error = np.sqrt(mean_squared_error)
         print(root_mean_squared_error)

         38.639571620011715
```

Figure 9: Neural Network Evaluation



```
In [18]: # CALCULATE THE R2 VALUE
         r2 = r2_score(y_test_windowed, y_pred)
         print(f'R-squared (R²) Score: {r2}')

         R-squared (R²) Score: 0.9634577405094105

In [19]: # CALCULATE THE MEAN SQUARED ERROR
         mse = mean_squared_error(y_test_inv, y_pred_inv)
         print(mse)

         1232.2945302797511

In [20]: # CALCULATE THE MEAN ABSOLUTE ERROR
         mae = mean_absolute_error(y_test_inv, y_pred_inv)
         print(mae)

         21.75538279548768
```

Figure 10: Recurrent Neural Network Evaluation



```
In [5]: # CALCULATE MEAN ABSOLUTE ERROR
        mae = mean_absolute_error(y_test, y_pred)
        print(mae)

        22.89045825858731
```

```
In [6]: # CALCULATE ROOT MEAN SQUARED ERROR
        squared_differences = (y_test - y_pred) ** 2
        mean_squared_error = np.mean(squared_differences)
        root_mean_squared_error = np.sqrt(mean_squared_error)
        print(root_mean_squared_error)

        42.99375935106599
```

Figure 11: Linear Regression Evaluation

## References

Shafl, A. (2023), 'How to use jupyter notebooks: The ultimate guide'.
**URL:** *https://www.datacamp.com/tutorial/tutorial-jupyter-notebook*