City, University London

MSc in Data Science

Project Report

2021

Utilizing machine learning techniques to forecast change in a UK house price index

Yousef Gharib

Supervised by: Dr. Kizito Salako

Submitted: 20th December 2021

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: *Yousef Gharib*

ABSTRACT

MIAC analytics provide a house price index made of true transactions where transactions of a specific date are obtained over the course of a few months. This study aims to predict the change in house prices as more transactions are received using machine learning with previous transactions and macro-economic data as predictors. The machine learning models selected are support vector regressor, multilayer perceptron, recurrent neural network, and long-short term memory network where the results showed that long-short term memory neural networks provided in the lowest RMSE when trained both with and without historic data and predicts more accurately than the stated base predictions.

Keywords: Time series forecasting, macro-economic data, house price prediction, Support Vector Regression, Neural Networks

List of tables

Context

Methods

Results

List of Figures

Introduction

Context

Methods

Results

Discussion

Table of content

1. Introduction and Objectives

1.1. Background

Mortgage Industry Advisory Corporation (MIAC) Analytics is an independent consultancy based in the United States and has been a leading provider of mortgage and financial services such as risk management and data auditing.

MIAC analytics have their own UK house price index based on real property sold, as recorded by the Land Registry in the UK that is updated monthly; transactions are recorded very slowly by Land Registry therefore it may take months to receive all the transactions that occurred in a single month. This issue has worsened since the start of COVID-19 resulting in the need to predict the change in the aggregated house price (known as restatements) as more volumes of transactions are recorded.

The formal explanation of the problem is if $HPI(m, d_i)$ is the house price index value for the date of the transaction d_i for transaction i calculated in index date m, the then generally for $n > 0$, $HPI(m, d_i) \neq HPI(m + n, d_i)$ due to the way data is obtained over time.

If assumed that for $n = N$ large enough, we have $HPI(m + N, d_i) = N = HPI(m + n + k, d_i)$ for all $k > 0$ meaning we have all the transactions for month d_i in month $m + N$, and therefore know the index value at that point and have been approximating $HPI(m + N, d_i)$ using a subset of the full data.

Therefore, the research question this project aims to answer is:

How well can we predict future restatements using macroeconomic data?

Through this we may be able to predict the restated values for lagged months as well as newer months with missing data, thus allowing us to calculate the aggregated price for that index.

This project is expected to have the following beneficiaries:

- The results from this project will directly benefit MIAC analytics as it may attract new customers if advertised as a product.
- Consumers of MIAC products may also benefit from this project as they may have exclusive access to predicted results and can make decisions based on the results.

1.2. Research objectives

To answer the research question, key objectives are required and outlined in this subsection.

- *Data acquisition:* The data required consists of a mix of the index data which will be obtained from the MIAC database and macro-economic data which can be obtained from other sources such as Office for national statistics (ONS). This objective aims to retrieve appropriate data for the remaining objectives.

- *Data exploration and pre-processing:* An exploratory analysis will be conducted to visualize properties of the data such as distribution, outliers and relationships found. The data will then be processed through merging datasets and calculations of new features to allow for all tests to be carried out. This step is crucial, and the resulting datasets will be used for model training and evaluation.

- *Variation of restatements:* The time taken for restatements to converge can be calculated given the processed data; this may vary greatly based on factors such as location and property type. By finding the average time it takes for restatements to converge, we can decide on the amount of historic data to consider when predicting a certain month in the time series.
- *Modelling:* Specific machine learning models are selected based on relevant literature and will be trained on the processed dataset. Different data structures are also tested alongside the models. These models will be compared and used to predict the restatement for lagged months. This is the focus of the project and investigating how good of a prediction we can get using a machine learning model is the objective.
- *Evaluation:* The models will be assessed during the evaluation period using general machine learning metrics. Comparisons are made between models based on data structure and the best models are evaluated with base predictions. This objective allows us to identify each model's performance and evaluate which are the best for this task.

1.3. Methods

An extensive literature review was conducted to evaluate methods that are worth testing based on similar studies conducted. This task is unique in that the time series data are dependent on two different dates, being index month m and date d. Since literature on forecasting data based on multiple dates could not be found, the sources focused on other topics relevant to the study, being time series forecasting using machine learning, short-term time series forecasting and house price prediction with and without the use of macro-economic data.

Python is the main programming language used to complete the objectives stated as it is widely used and provides all the tools necessary. An outline of the methods used to complete objectives are outlined below:

Data acquisition

To acquire the necessary data to carry out the project, the query language SQL was used to access the MIAC analytics database and retrieve the index data as well as data on the number of house sales each index month. The macro-economic data was obtained from public domains such as ONS and with the help of supervisors for more up-to-date data.

Data exploration and pre-processing

The data is investigated thoroughly to find trends between variables and to understand the structure. The data is then processed as seen necessary using python libraries to create new columns, label encode the categorical variables, and prepare the data for model training. Multiple datasets of different data structures are created to train different models and evaluate which data structure is the most suitable, the data structures created are Data split 1 where early months are used to predict later months, Data split 2 with all index months used in training and a dataset where a smoothing algorithm called EEMD is used.

Variation of restatements

Once data exploration is carried out, the variation of restatements can be calculated by evaluating the average time taken for restatements to converge for a subset of geographies, this determines approximately how much historic data would be necessary for training. New variables are created to verify the calculations are correct.

Modelling

Based on the literature, the machine learning models chosen to test are Support vector regression (SVR) and long-short term memory (LSTM) neural networks. Simpler network structures are tested in addition to LSTM, being multi-layer perceptron (MLP) and recurrent neural networks (RNN) to ensure there is an improvement with model complexity. A transfer learning (TL) model is created and tested to include trends found in historic data to predict the restatements of data with no history. While the number of algorithms being tested is low, different data structures and transformations are also tested for the chosen models.

Figures 1.1 and 1.2 are made to demonstrate the difference in normal time series forecasting problems compared to the approach for this project.

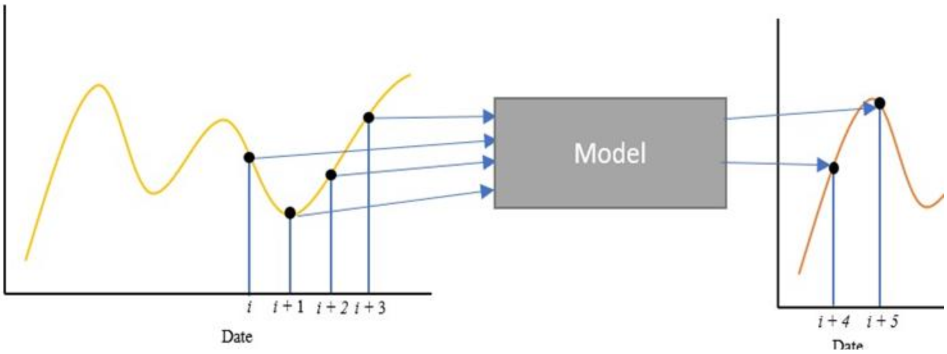


Figure 1. 1: How time series data is fed to a model

Given that restatements are the percentage change in the price each index month of a specific geography that occurred on date d, the data is fed such that the time series represents the change in price each index month instead of the change in price in the same index month.

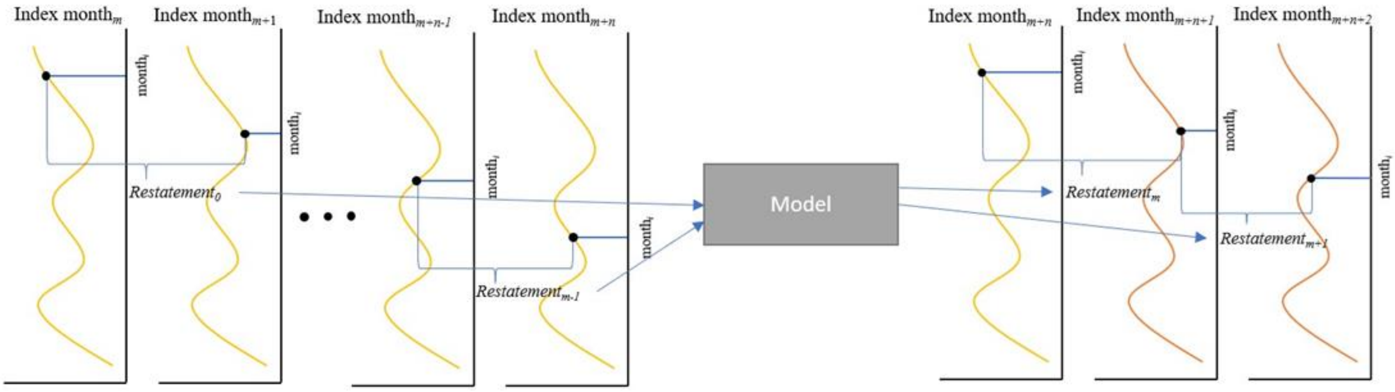


Figure 1. 2: How time series data will be fed

Evaluation

The models are all evaluated using the same metrics and are compared on the same test set. The main evaluation metric used compares the actual and predicted restatement, being root mean square error as done by (Chen, n.d.) where neural networks are used to predict house prices. Different levels of base predictions are created and used to compare if the optimal network can outperform obvious predictions that would be used for restatement, an example of this is the mean restatement for a country or region. In addition to this, the neural networks are also evaluated using model metrics to judge the performance of the models.

1.4. Work plan

A work plan cycle was designed to outline the stages of the projects and relate them to the completion of objectives, shown in figure 1.3. The workflow was designed in a cycle to demonstrate the process of developing new ideas and implementing them throughout the project, particularly when it comes to testing new data structures.

The plan is outlined and grouped based on objectives to identify what tasks help achieve which defined objective. Most of the literature search is carried out before starting the project, however as the project developed more sources are obtained.

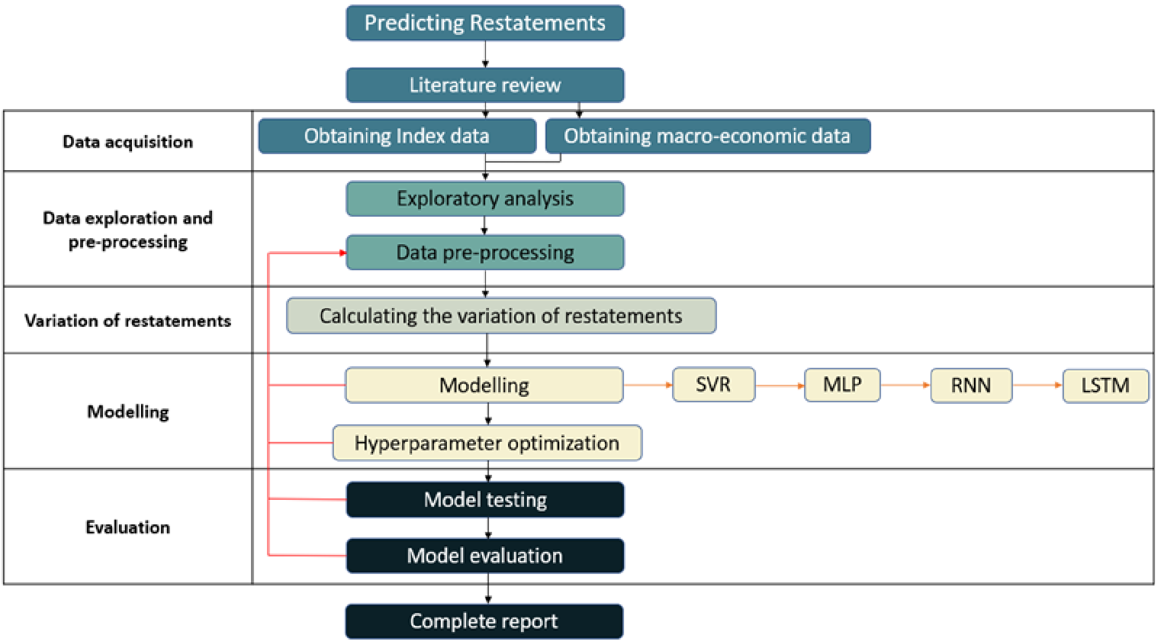


Figure 1. 3: Work plan

1.5. Changes in goals

The initial goal of the project was to investigate how well we could predict future restatements given historic restatements and macro-economic data, the underlying issue with using historic data is that it limits the number of months we can predict, particularly the later months since no historic or not enough historic data is available. The newer months are the ones we are most interested in predicting as they will likely have the greatest change in restatements, this is discussed in later chapters. Without the use of historic data, it is fair to say the task becomes more of a regression problem, comparisons are made with and without the use of historic data and an attempt to indirectly include historic data is made through transfer learning. This resulted in the research question changing slightly to exclude the requirement of using historic data.

1.6. Report structure

The report is split into 6 separate sections, the first section introduces the problem and outlines the objectives such that no previous knowledge on the problem is required.

Section 2 discusses relevant literature and theory around the project to help understand the methods used as well as justify the reasoning for the chosen methods.

Section 3 describes in detail the methods applied. How the data was obtained, the exploration and what was found, the pre-processing steps, modelling and design of experiments and finally how the models are evaluated.

Section 4 provides the results obtained through the application of selected methods, as well as an analysis of the results.

Section 5 discusses the results found in relation to the defined objectives.

Section 6 evaluates the project overall as well as discusses what was learned through the whole process and possible future work.

2. Context

2.1. Overview

This section aims to discuss the context of the project as well as a literature review of papers closely related to the main objectives; the purpose of the literature review is to understand the current state of time series forecasting and to use results of other tests to justify the application of certain methods for this project.

MIAC analytics apply an interpolation methodology to their house price index to add more data points, this is further discussed in the context as it plays a major role in properties of the data which will be examined in later sections.

In addition to this, the models chosen to be used in this project are explained in theory to provide an understanding of how they work as well as their application and performance in general time series forecasting problems found in the relevant literature.

2.2. Time series forecasting

Time series forecasting can be simply defined as making predictions based on historic data. Time series analysis is a very large topic and is popular particularly in econometrics since many variables are dependent on time, such as stock prices.

Forecasting can be generally split into one-step and multi-step forecasting, where one-step forecasting predicts the observation at the next step, while multiple time steps are predicted for multi-step (Brownlee, n.d.) where each prediction is based on the previous. This project focuses on one-step forecasting instead of multi-step.

In more recent years, machine learning algorithms and neural networks are being tested and compared with classical methods for time series forecasting; while they generally don’t outperform classical methods, they have shown to perform well and in some cases outperform classical methods, as shown in forecasting the U.S. real house price index while using macro-economic data (Plakandaras, 2014).

2.2.1. A comparison of classical and machine learning methods

Forecasting algorithms that are more statistically focused are generally what are considered to be classical or simpler methods (Brownlee, n.d.), this includes algorithms such as Auto regressive integrated moving average (ARIMA) and Vector autoregression (VAR) which are some of the most popular methods and are very often used as a baseline model to compare new methods with.

Why use machine learning if classical methods usually outperform them? Classical methods may usually outperform machine learning techniques for house price prediction (Milunovich, 2019), however, they have limitations. A big limitation is that they require historic data to be able to make forecasts whereas machine learning can provide the ability to predict initial values based on other factors. In addition to this, assumptions such as data stationarity which is almost never the case; some transformations can be applied to the data to make it stationary as shown by (Brownlee, n.d.) but carrying these transformations out on data consisting of many time series can be quite challenging, particularly when carrying out hypothesis tests for seasonality in the data, in addition to this the time series data used in this project is likely not long enough to show such trends.

Some machine learning models are used under the assumption the data must be normally distributed, such as linear regression, however algorithms such as neural networks make no assumptions on the data, this not only makes data preparation easier as additional transformations are not required, but machine learning models are generally very adaptable and customizable allowing them to be tailored to a specific problem. Additional data can also be added to the model to increase performance as done by (Plakandaras, 2014) and discussed by (Milunovich, 2019) where macro-economic data is used to predict house prices. The use of machine learning in time series forecasting is still being tested as there is huge potential for them to outperform classical methods, particularly given the flexibility of machine learning models.

2.3. Theory of algorithms

This subsection describes the theory of the methods applied both on the data structure, machine learning algorithms chosen, and their use cases found in the relevant literature.

2.3.1. Principal component analysis (PCA)

PCA is a popular method used in unsupervised learning tasks to reduce the dimensions of a dataset data into components where each component is a linear combination of the variables, each principal component (PC) explains a certain percentage of the variance of the dataset, and so using a certain number of components can significantly reduce the side of the dataset while retaining most of the information (Jolliffe, 2002). PCA is used to reduce the dimensions of some macro-economic variables which provide rates based on different length terms, such as 1 year, 2 years, 5 years etc. PCA aims to minimize the sum of perpendicular distances from an input value to each principal component.

2.3.2. Ensemble empirical mode decomposition (EEMD)

EEMD decomposes signals into finite oscillatory components known as intrinsic mode functions (IMFS) where the decomposition into IMFS is attained through an iterative process until residuals have no local extrema (Plakandaras, 2014). Generally, each decomposition can represent different dynamics of the time series, such as long-term and short-term trends. Decompositions can be summed to result in a smoothed time series as done by (Plakandaras, 2014), and they showed that the use of EEMD smoothing with a support vector regressor outperformed autoregressive models when trying to forecast the U.S. house price index with the use of macroeconomic data. Figure 2.1 shows an example of a time series as well as its decomposed components. Each IMF displays seasonal trends found in the data and can be used for various applications such as detection of abrupt changes (Yun-long Kong, 2015).

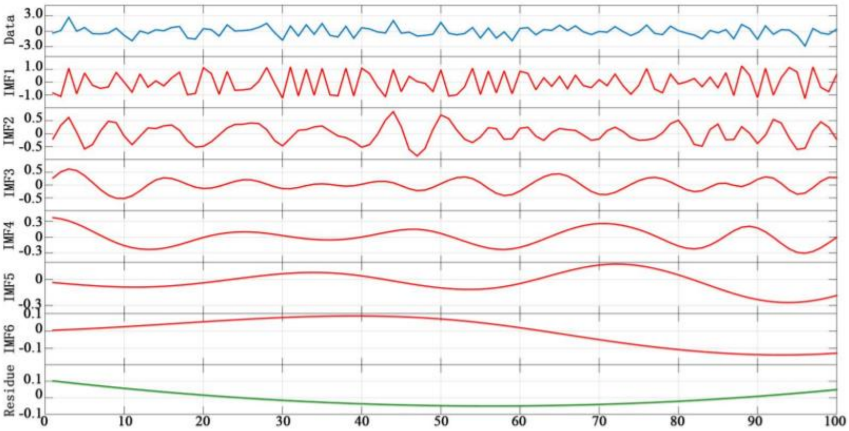


Figure 2. 1: Illustration of decomposed components (Yun-long Kong, 2015)

2.3.3. Support Vector Regressor (SVR)

Support vector machine is a popular algorithm usually used for classification tasks; an adapted version made for regression tasks follows similar principles and maintains its robustness as a machine learning algorithm. In classification tasks, a SVM initializes a hyper-plane to segregate classes in a particular dataset, the hyper-plane is constructed conditionally to maximize the distances nearby data points for each class (Kecman, 2013); for regression tasks, the algorithm aims to create a function where differences between actual data points and the function output are within a threshold accuracy.

SVR can perform excellently for time series forecasting as shown by (Debasish Basak, 2007), where in some cases it has outperformed other popular machine learning models, specifically for short term time series prediction (Plakandaras, 2014). SVRs potential can directly scale with high dimensionality data through kernel functions allowing for data transformations, this creates an easier environment to optimize the loss functions. A limitation of SVR is that it works by solving constrained quadratic equations to minimize the loss function in which time can increase quadratically with the training set (Bottou, n.d.). A combination of hyper-parameter optimization and long SVR training times can result in a significant loss in available time due to model training.

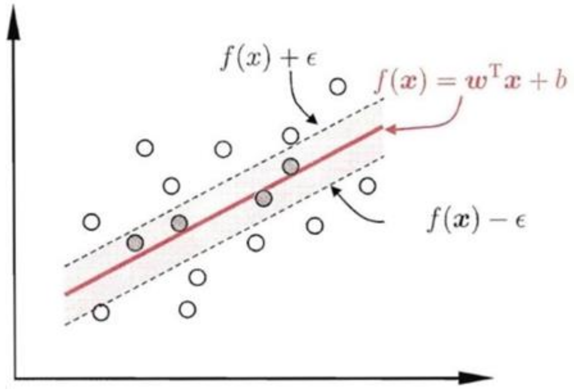


Figure 2. 2: SVR hyperplane with formula (Anon., 2018)

2.3.4. Neural networks (NN)

2.3.4.1. Multi-layer Perceptron (MLP)

MLP is a feed-forward neural network and has been shown to excel in supervised classification and regression tasks. The structure generally consists of an input layer with the same number of neurons as the columns of the dataset, hidden layer(s), and an output layer; for dense layers, all neurons are fully connected to the neurons of the next layer. The model parameters consist of weights and biases that are initialized randomly at the start of model training and are tuned with model training, an activation function is applied to each neuron to deactivate or activate it. The general structure of an MLP is shown in figure 2.3.

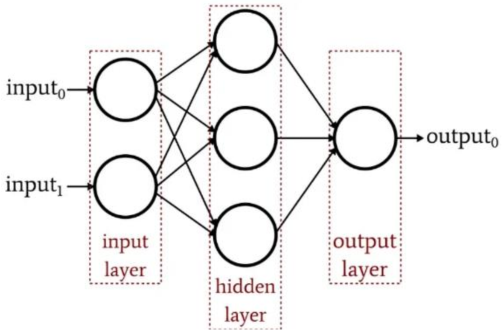


Figure 2. 3: Structure of MLP (Keim, 2019)

MLP has been used for time series forecasting in many cases such as predicting the spread of COVID-19 (Pedro Henrique Borghia, 2020) and was said to describe and predict the behaviour for up to six days. The simplicity of the MLP structure allows for easy implementation and adjustability, making it a popular choice for machine learning tasks.

2.3.4.2. Recurrent Neural Network (RNN)

RNN is a branch of artificial neural networks generally used on sequential or time-based data. Similar to MLP, it is a feed-forward network however RNN nodes have an internal memory state allowing them to process sequences of variables (Dupond, 2019), this can be better understood in application on python where the data input is required in 3-dimensions, the 3rd dimension being sequence length.

Each recurrent node iterates over itself n times, where n is the sequence length this can be seen by the arrow curving and pointing back into the node in figure 2.3.

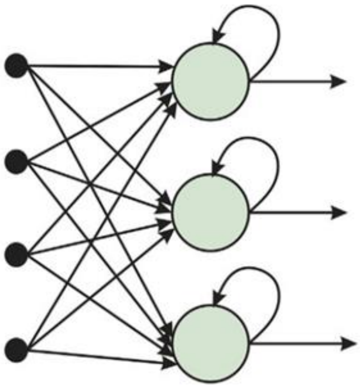


Figure 2. 4: Example of neural network structure with recurrent nodes (Ashkan Eliasy, 2020)

RNNs are a popular choice for time series forecasting because of their internal memory state, however, RNNs are being adapted to account for limitations, examples of variants are Dual-stage attention-based RNN which has shown to adaptively select the most relevant input features as well as capture long-term dependencies for general time series prediction (Yao Qin, 2017), Dual-path CNN-RNN cascade network aimed to improve feature extraction and reduce model training times (C. Yang, 2019).

2.3.4.3. Long-Short Term Memory Neural Network (LSTM)

Long short-term memory neural networks are artificial recurrent neural networks that are widely used in pattern recognition of sequential data such as speech recognition, music composition and more recently time series prediction. LSTMs work similarly to other recurrent neural networks in that the network loops over itself for n iterations where n is the sequence length, also requires 3-dimensional data in the application with Python; the main advantage of LSTMs is that they can learn long term dependencies of sequential data such as seasonal trends through the cell state in the network structure, shown by the line containing nodes 'X' and '+' in figure 2.4. The cell state stores additional information at each iteration.

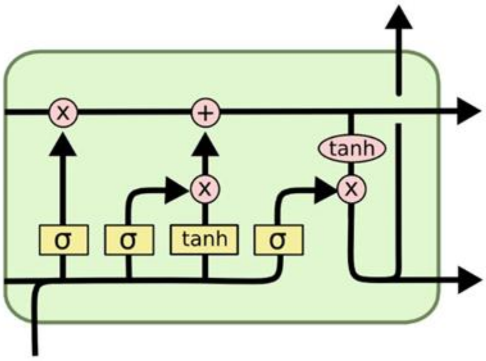


Figure 2. 5: LSTM node structure (Olah, 2015)

The strength of learning long-term dependencies makes LSTM an excellent time series forecasting model. LSTM has become a popular model to test when using machine learning for house price prediction and has been shown to outperform ARIMA (Chen, n.d.). LSTM is versatile in that variants such as bidirectional and stacked LSTMs can be used to adapt to the complexity of the data, as well as the ease of adjusting the output structure to incorporate for multiple predictions that may be based on geographic location as done by (Chen, n.d.).

2.4. MIAC's interpolation methodology

MIAC analytics employ an interpolation methodology to increase the number of data points to reduce the volatility of their house price index, understanding this implementation is necessary to gain more insight on relationships found in the data. To understand their methodology, there are some preliminary requirements to understand, these are:

- Postcode mapping
- Repeat sales
- Matched samples

Postcode mapping

When a property is sold, information such as the date of transaction, the value of a property, postcode, and property type are recorded at the time of registration. As part of MIAC's pre-processing, the postcode is mapped to multiple variables that represent the location in less detail, being Country, Region and Geography. This gives a more general idea of the location of the property allowing them to be grouped easier whereas the postcode is more accurate.

Repeat sales

House prices are not observable until properties are sold or their value is estimated. Therefore, a house price index is most likely made up simply of observed values, an example of this is shown in table 1, where missing points are points with observable house price value for a specific property.

Table 2. 1: Data source summary

Property	**Time**						
	0	1	2	3	4	…	T
Property 1	$P_{1,0}$		$P_{1,2}$			…	
Property 2			$P_{2,2}$			…	$P_{2,T}$
Property 3		$P_{3,1}$				…	$P_{3,T}$
Property 4	$P_{4,0}$				$P_{4,4}$	…	
…	…	…	…	…	…	…	…
Property N	$P_{N,0}$			$P_{N,3}$		…	$P_{N,T}$

Using repeat sales, the change in house price over a given period is approximated by the observed changes in individual properties during the same period; for example, from table 1 we can infer information on the change in house prices for a specific property in a specific period based on the house prices of other properties in the same period.

Matched samples

Homogeneous sub-samples of properties are created with the idea that they are treated as the same property if they share the same characteristics, with this the grouped properties are likely to be of similar value. This can increase the number of repeat sales significantly, in turn providing more information to base the approximated results on.

It is assumed that two properties with the same postcode and property type are sufficiently homogeneous, and therefore their values are approximately equal.

Methodology

Linear interpolation is the algorithm used to estimate the value of a property at time T; this can be simply defined as a straight line between two known points. The formula for linear interpolation is

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)},$$

Where y is the interpolated point between the two known points with coordinates (x_1, y_1) and (x_2, y_2).

This increases the number of data points and smoothens discontinuous time series. This implementation is demonstrated below in table 2 where the interpolated values are coloured in light blue. Not all values can be calculated at a certain point in time as they must be in between known points, for example, $P_{1,3}$ cannot be calculated as $P_{1,n}$ for $n > 3$ is unknown.

Table 2. 2: Data source with interpolation

Property	**Time**						
	0	1	2	3	4	…	T
Property 1	$P_{1,0}$	$P_{1,1}$	$P_{1,2}$			…	
Property 2			$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	…	$P_{2,T}$
Property 3		$P_{3,1}$	$P_{3,2}$	$P_{3,3}$	$P_{3,4}$	…	$P_{3,T}$
Property 4	$P_{4,0}$	$P_{4,1}$	$P_{4,2}$	$P_{4,3}$	$P_{4,4}$	…	
…	…	…	…	…	…	…	…
Property N	$P_{N,0}$	$P_{N,1}$	$P_{N,2}$	$P_{N,3}$	$P_{N,4}$	…	$P_{N,T}$

MIAC adapts this method to the case where prices are given by matched samples that are based on postcode and property type as mentioned previously, resulting in a table like Table 3 below, where **SP** is the value of the matched sample based on postcode, property type **pt** and time **T**.

Table 2. 3: Data source summary with interpolation and matched samples

Postcode	**Time**						
	0	1	2	3	4	…	T
Postcode 1	$SP_{1,0,pt}$	$SP_{1,1,pt}$	$SP_{1,2,pt}$			…	
Postcode 2			$SP_{2,2,pt}$	$SP_{2,3,pt}$	$SP_{2,4,pt}$	…	$SP_{2,T,pt}$
Postcode 3		$SP_{3,1,pt}$	$SP_{3,2,pt}$	$SP_{3,3,pt}$	$SP_{3,4,pt}$	…	$SP_{3,T,pt}$
Postcode 4	$SP_{4,0,pt}$	$SP_{4,1,pt}$	$SP_{4,2,pt}$	$SP_{4,3,pt}$	$SP_{4,4,pt}$	…	
…	…	…	…	…	…	…	…
Postcode N	$SP_{N,0,pt}$	$SP_{N,1,pt}$	$SP_{N,2,pt}$	$SP_{N,3,pt}$	$SP_{N,4,pt}$	…	$SP_{N,T,pt}$

A piece-wise continuous time series can then be constructed for each property within the repeat sales data using the interpolated values to create a smoothened time series.

3. Methods

3.1. Overview

This section aims to describe the methods done in detail used to help complete the objectives. The subsections are split by objective and the methods described in each subsection helped achieve the defined objectives.

3.2. Data

3.2.1. Data acquisition

The necessary data required can be split up into index data and macroeconomic data. The index data provides the necessary information on the location of houses as well as their sale value, while the macro-economic data provides additional information that is assumed to correlate with the house prices found in the index data. In addition to this, volume data is obtained for the exploration phase to understand how the volume of sales is related to the change in price.

3.2.1.1. Index Data

The index data is stored in the MIAC analytics database and is acquired using the query language SQL. Each month, a new index dataset is created in the database with the most up to date index values for new transactions that occurred on recent dates as well as previous dates but are obtained late due to the delay of recorded transactions, for example, if the current month is June 2020, we may only have just received the first batch of transactions that occurred in May 2020 which could account for around 70% of all May 2020 transactions, and in addition to that some other transactions that occurred in months before May 2020; this is the reason there is a lag in the index data and prices change each index month.

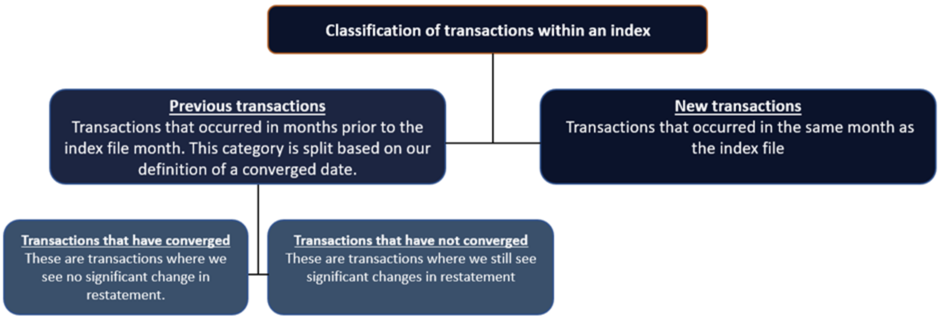

Classification of transactions within an index
Previous transactions
Transactions that occurred in months prior to the index file month. This category is split based on our definition of a converged date.
New transactions
Transactions that occurred in the same month as the index file
Transactions that have converged
These are transactions where we see no significant change in restatement.
Transactions that have not converged
These are transactions where we still see significant changes in restatement

Figure 3. 1: Description of types of transactions found in index data

Restatements cannot be calculated using a single index month since it is calculated as the change in price each index month, therefore multiple index months are required to calculate the restatement time series. A python loop was created to pull all the available index data from the MIAC database individually and append them on top of each other to create one large dataset of all the available index data, this was then saved as a CSV; the first available index date is October 2018 and to avoid anomalies in the index values, months where COVID is present are ignored. Therefore, the range of index months used is October 2018 up till March 2020.

Below is a table of the description of variables in the index data that are used as predictors for the chosen models.

Table 3. 1: Description of variables in Index data

Variable	Description	Variable type
Index_month	Month in which the dataset was created.	Categorical
Date	Date of the transaction.	Categorical
Index_Country	Country of the transaction	Categorical
Index_Region	Region of the transaction	Categorical
Index_Geography	County of the transaction	Categorical
Property_Type	Property type	Categorical
Index	A percentage measure of the change in house prices from a specific date. Starts at 100 on date of transaction.	Float/Real number
Price	An aggregation of all the true house prices of recorded transactions for matched samples.	Float/real number

3.2.1.2. Macroeconomic data

The macroeconomic data to be used were chosen based on literature and what was available. This data was not present in the MIAC database and was obtained with the help of supervisors. The macroeconomic data used are interest rate, unemployment rate, gross domestic product (GDP), zero-coupon retail price index (RPI), SONIA and FTSE 100. An advantage of using index months up until March 2020 is that we do not have any lag with macroeconomic data as we only require old data.

Interest rate, unemployment rate and GDP were downloaded from the official Office for national statistics website (ONS) and is available for public use. The remaining macroeconomic data was obtained through a subscription MIAC analytics have with Bloomberg where they have access to specific up to date macroeconomic data. Below is a table with a description of all the macroeconomic variables used. All the macro-economic data contain a date column that can be merged with the index data.

Table 3. 2: Description of macro-economic data

Variable	Description	Variable type
Interest Rate	The amount of interest due per period.	Float/real number
Unemployment rate	Percentage of UK population unemployed	Float/real number

GDP	A measure of a country’s wealth.	Float/real number
Zero-coupon RPI	A measure of inflation including mortgage rates in the UK.	Float/real number
SONIA	A reflection of the average interest rate based on real transactions.	Float/real number
FTSE 100	A barometer of the UK financial markets.	Float/real number

The Bloomberg data provides rates over multiple years for each variable, to incorporate as much of this data as possible without increasing the number of columns significantly, principal component analysis is applied to the Bloomberg data to reduce the dimensions.

3.2.1.3. Volume data

The number of transactions registered each index month is recorded in the volumes data. This informs us on how many transactions of a specific combination of date of transaction, geography, property type occurred for a certain index month. While this data is not used as a predictor for the models, it plays a major role in understanding the data as well as interpolation methodology and is necessary for the data exploration objective.

This data is stored in the MIAC analytics database and was obtained using a python loop of SQL code to query and append the volume data each month to create the final dataset, similar to the method done to obtain index data.

Table 3. 3: Description of volume data

Variable	Description	Variable type
Index_month	Month in which the data was created.	Categorical
Date	Date of the transaction.	Categorical
Index_Country	Country of the transaction	Categorical
Index_Region	Region of the transaction	Categorical
Index_Geography	County of the transaction	Categorical
Property_Type	Type of property	Categorical
Sales	Total number of sales recorded for a specific geography and property type since the date of the transaction.	Integer

Obtaining the index, volume and macroeconomic data completes the data acquisition objective, with this done, data exploration and pre-processing can take place.

3.2.2. Data exploration and pre-processing

Exploring the data is necessary before pre-processing and model training as it gives insight on the data being used, it also guides us on what steps to take during the pre-processing phase such as scaling the data and removing null values. To complete the second objective, data pre-processing is carried out after sufficient exploration to prepare the data for modelling.

3.2.2.1. Data exploration

The exploration phase is inherently different to others as there is no specific goal, similar to an unsupervised learning task; this makes it difficult to know when to stop and move on to the next phase.

To better understand the data, diagrams and figures were heavily relied on to visualize the relationship between variables. Histograms were used to visualize the distribution of variables, correlation heatmaps are used to display the correlation between variables and boxplots are used to show outliers in restatements. The distribution of restatements based on *Index_month* and *Date* are explored as well as 3D visualizations to show restatements based on *Date* and *Index_month*. Preliminary assumptions can be made on the relationship between the volume of sales and the change in prices, this is further investigated in the exploration to see if the assumptions are true.

3.2.2.2. Data pre-processing

To proceed to the modelling phase, it is necessary to prepare the data in a format that can be fed to the selected models. This plays a major role in the performance of the model as different inputs can significantly affect the accuracy of the models. Figure 3.2 displays the workflow of pre-processing steps taken before modelling.

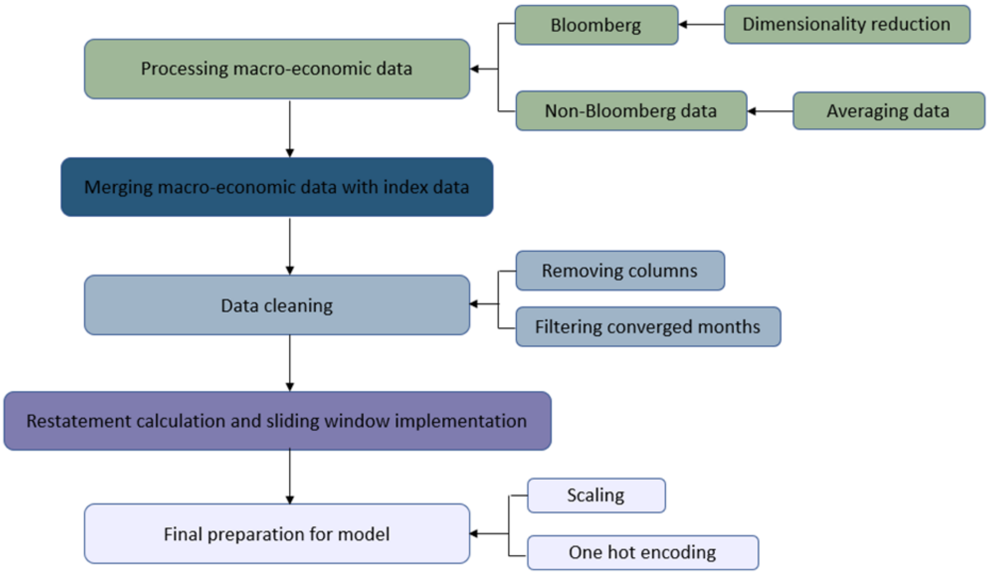

Processing macro-economic data
Bloomberg
Dimensionality reduction
Non-Bloomberg data
Averaging data
Merging macro-economic data with index data
Data cleaning
Removing columns
Filtering converged months
Restatement calculation and sliding window implementation
Final preparation for model
Scaling
One hot encoding

Figure 3. 2: Workflow of pre-processing steps

Processing macro-economic data

The initial step in data pre-processing is preparing the macro-economic data. This data can be split into Bloomberg and non-Bloomberg data, where Bloomberg data are data obtained from the Bloomberg terminal MIAC analytics pay for and non-Bloomberg data are data not obtained from this source. This grouping helps describe the processing carried out easier since the formats of the two groups of data are very different.

Bloomberg data

As mentioned previously, the Bloomberg data provides rates based on terms i.e., the number of months or years into the future. To avoid adding all the columns which may result in a dataset being too large, principal component analysis (PCA) was applied to reduce the dimensions of the Bloomberg data, a function was created to carry out this procedure for each Bloomberg variable. The procedure is as follows:

- *Filter dates.*
- *Inspect for null values.*
- *Split into training and testing.*
- *Scale data.*
- *Apply PCA.*

The dates available are from 1995-01-31 up to 2021-07-30, since the index data we have available is from 2018-10-31 up to 2020-03-31 we can filter out the dates in the Bloomberg data that do not correspond to the available index data months. The Bloomberg data is then inspected for null values as the dimensionality reduction cannot be completed with missing values, once this was done the data was split using the same train/test split that is discussed in the modelling section 3.4, this is done to avoid information from the test data from being leaked into the training data when carrying out model training and testing.

A standard scaler transformation is applied to the data scaling features to have mean $u = 0$ and standard deviation $s = 1$, described by equation (1),

$$Z = \frac{x - u}{s}. \tag{1}$$

Resulting in faster calculations of the principal components and is done at this stage for the Bloomberg data and later for the remaining data, it is important to note that the standard scaler transformation does not standardize the data, this means that the overall distribution remains the same however the range of values is changed.

The transformation applied to the training set is then applied to the test set to ensure they undergo the same transformation.

The final step is to carry out the PCA on the training set and then to fit the transformation on the test set. The training set and test set are then appended, and the number of principal components chosen was based on the cumulative explained variance calculated from the PCA.

Non-Bloomberg data

The non-Bloomberg data is represented as daily rates and not monthly. Therefore, the variables are grouped by month and year, the average of each variable is then calculated and the date was set to the end of each month. To achieve this, the date column was converted from a string to DateTime, making it easier to calculate the monthly average of variables.

Merging macro-economic data with index data

The resulting data is merged into a single dataset. All the macro-economic data are merged on variable *Date* and this merges with the index data on *Index_month*. The resulting dataset has 4,984,092 rows and 27 columns, further cleaning is done to significantly reduce the dataset size as it is too large to use for model training.

Data cleaning

To reduce the dataset size columns which we are not interested in are removed, these are generally columns that provide information relating to the recording of price, such as whether the price value for a specific row is null. In addition to this, the number of rows is reduced by filtering out dates of transactions that occurred before a certain time. This threshold is determined by which dates we consider having a converged price by our definition, by calculating the variation of restatements the dates that have converged can be filtered and removed as we are mainly interested in predicting the restatement of dates that have not converged.

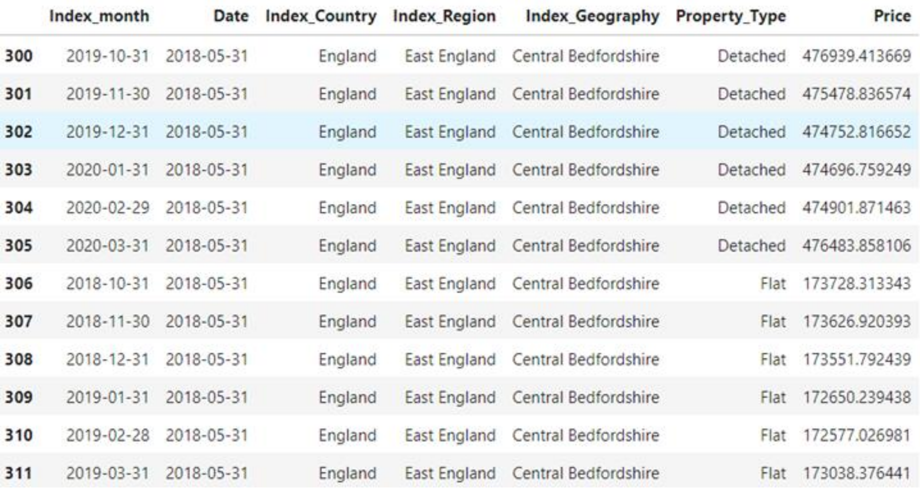
Restatement calculation and sliding window implementation

Restatements are calculated using the formula

$$Restatement_m = 100 \times \left(\frac{Price_{m+1}}{Price_m} - 1\right) \tag{2}$$

Given the structure and size of the dataset, a method of calculating the restatements was made with the aim of quick calculation, this gives freedom in case restatements need to be recalculated and saves time especially since it is a large dataset.

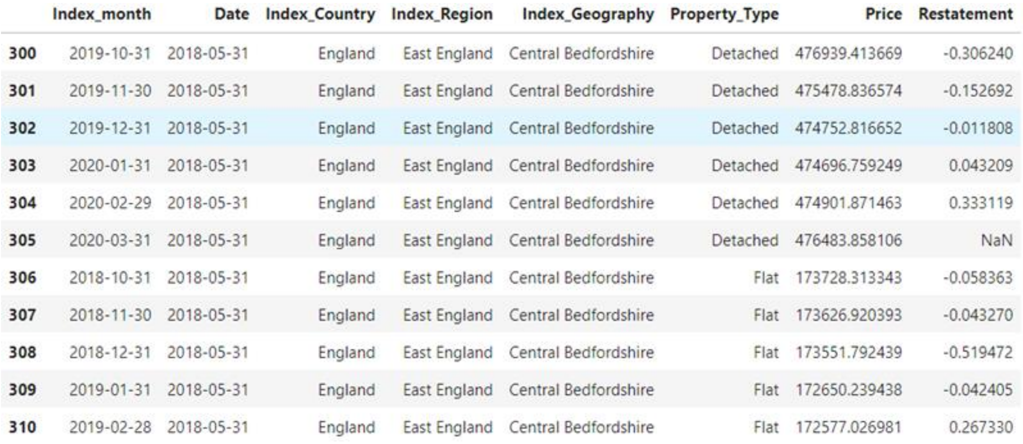
Firstly, the data is sorted in order of columns *Date, Index_Country, Index_Region, Index_Geography*, *Property_Type* and finally *Index_month,* doing this arranges the data such that time series of *Price* for each combination of column variables are appended on top of each other. Figure 9 demonstrates this arrangement on the dataset, the time series of *Price* for transactions that happened on date “2018-05-31” in “Central Bedfordshire” for *Property_Type* “All” can be seen above the time series of *Price* for transactions that happened on date “2018-05-31” in “Central Bedfordshire” for *Property_Type* “Detached”.



	Index_month	Date	Index_Country	Index_Region	Index_Geography	Property_Type	Price
300	2019-10-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476939.413669
301	2019-11-30	2018-05-31	England	East England	Central Bedfordshire	Detached	475478.836574
302	2019-12-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474752.816652
303	2020-01-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474696.759249
304	2020-02-29	2018-05-31	England	East England	Central Bedfordshire	Detached	474901.871463
305	2020-03-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476483.858106
306	2018-10-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173728.313343
307	2018-11-30	2018-05-31	England	East England	Central Bedfordshire	Flat	173626.920393
308	2018-12-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173551.792439
309	2019-01-31	2018-05-31	England	East England	Central Bedfordshire	Flat	172650.239438
310	2019-02-28	2018-05-31	England	East England	Central Bedfordshire	Flat	172577.026981
311	2019-03-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173038.376441

Figure 3. 3: Sample of dataset

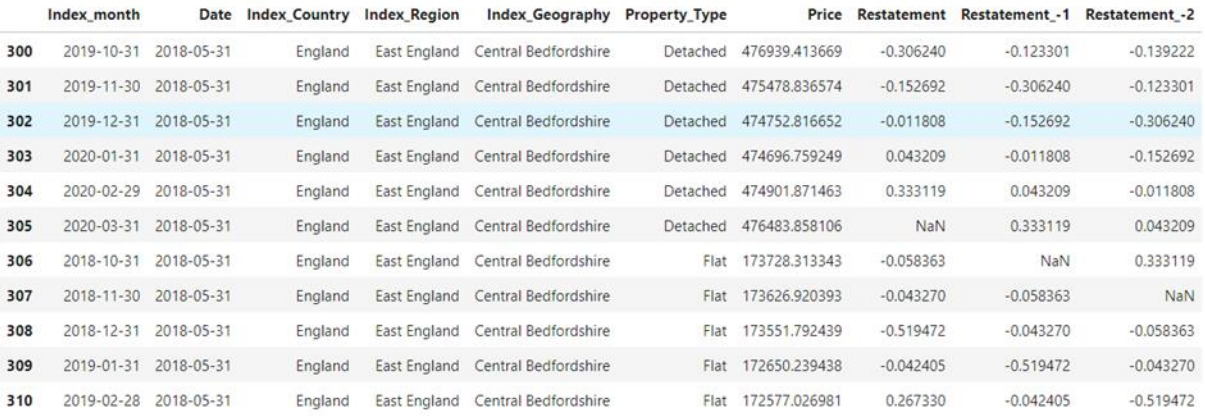
With this arrangement, we can calculate the restatements by applying formula 3 to the *Price* column. However, this results in the calculation of restatements between different combinations of column variables; to fix this, the restatements where $Index\_month\ =$ “$2020-03-31$”, is set as null. This is true anyway since to calculate the restatement for 2020-03-31, we require *Price* of 2020-04-31 which is unavailable as it is in the next index month data that is not used. Doing this to the dataset results in the correct calculation of restatements that does not take a significant time to run. Figure 10 demonstrates this using the same rows as Figure 9, restatements can be verified by applying formula 3 to the respective price values.



	Index_month	Date	Index_Country	Index_Region	Index_Geography	Property_Type	Price	Restatement
300	2019-10-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476939.413669	-0.306240
301	2019-11-30	2018-05-31	England	East England	Central Bedfordshire	Detached	475478.836574	-0.152692
302	2019-12-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474752.816652	-0.011808
303	2020-01-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474696.759249	0.043209
304	2020-02-29	2018-05-31	England	East England	Central Bedfordshire	Detached	474901.871463	0.333119
305	2020-03-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476483.858106	NaN
306	2018-10-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173728.313343	-0.058363
307	2018-11-30	2018-05-31	England	East England	Central Bedfordshire	Flat	173626.920393	-0.043270
308	2018-12-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173551.792439	-0.519472
309	2019-01-31	2018-05-31	England	East England	Central Bedfordshire	Flat	172650.239438	-0.042405
310	2019-02-28	2018-05-31	England	East England	Central Bedfordshire	Flat	172577.026981	0.267330

Figure 3. 4: Sample of dataset with restatement calculation

Once the restatements have been calculated, the next step is creating the sliding window that is used to incorporate historic data. As mentioned previously, the sliding window size is determined by the average time taken for months to converge, the method created allows for easily adjustable window sizes. To add the sliding window, a python loop is created that runs for n iterations where n is the sliding window size. The loop simply shifts the Restatement column down by n and adds a new column to the dataset called *Restatement_-n*.



	Index_month	Date	Index_Country	Index_Region	Index_Geography	Property_Type	Price	Restatement	Restatement_-1	Restatement_-2
300	2019-10-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476939.413669	-0.306240	-0.123301	-0.139222
301	2019-11-30	2018-05-31	England	East England	Central Bedfordshire	Detached	475478.836574	-0.152692	-0.306240	-0.123301
302	2019-12-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474752.816652	-0.011808	-0.152692	-0.306240
303	2020-01-31	2018-05-31	England	East England	Central Bedfordshire	Detached	474696.759249	0.043209	-0.011808	-0.152692
304	2020-02-29	2018-05-31	England	East England	Central Bedfordshire	Detached	474901.871463	0.333119	0.043209	-0.011808
305	2020-03-31	2018-05-31	England	East England	Central Bedfordshire	Detached	476483.858106	NaN	0.333119	0.043209
306	2018-10-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173728.313343	-0.058363	NaN	0.333119
307	2018-11-30	2018-05-31	England	East England	Central Bedfordshire	Flat	173626.920393	-0.043270	-0.058363	NaN
308	2018-12-31	2018-05-31	England	East England	Central Bedfordshire	Flat	173551.792439	-0.519472	-0.043270	-0.058363
309	2019-01-31	2018-05-31	England	East England	Central Bedfordshire	Flat	172650.239438	-0.042405	-0.519472	-0.043270
310	2019-02-28	2018-05-31	England	East England	Central Bedfordshire	Flat	172577.026981	0.267330	-0.042405	-0.519472

Figure 3. 5: Sampled dataset with restatement calculation and sliding window

Figure 11 shows this method with $n = 2$, it works as intended however we see that restatements of different combinations intertwine with each other, as seen on row 305-306 of figure 11 where the final few restatements of the previous combination are considered historic data for the next combination. By setting “2020-03-31” to be null, we ensure there is always a null value in these ‘corrupted’ rows, this means that when null values are removed these rows are too, resulting in correct historic data for each combination.

Encoding categorical variables and scaling data

The next step before model training involves scaling float and integer columns, and one hot encoding categorical variables. After the data is split into training and testing, the training set is scaled, and the same transformation is applied to the test set; the split varies based on the experiment and is discussed in section 3.4. Each column is scaled individually based on their values, the same scaling transformation applied to the *Restatement* column, is applied to the sliding window columns to result in the same scaled values.

Since the models selected do not take strings as inputs, the categorical variables are encoded to numeric values. The types of encoding used are mentioned below with a brief description:

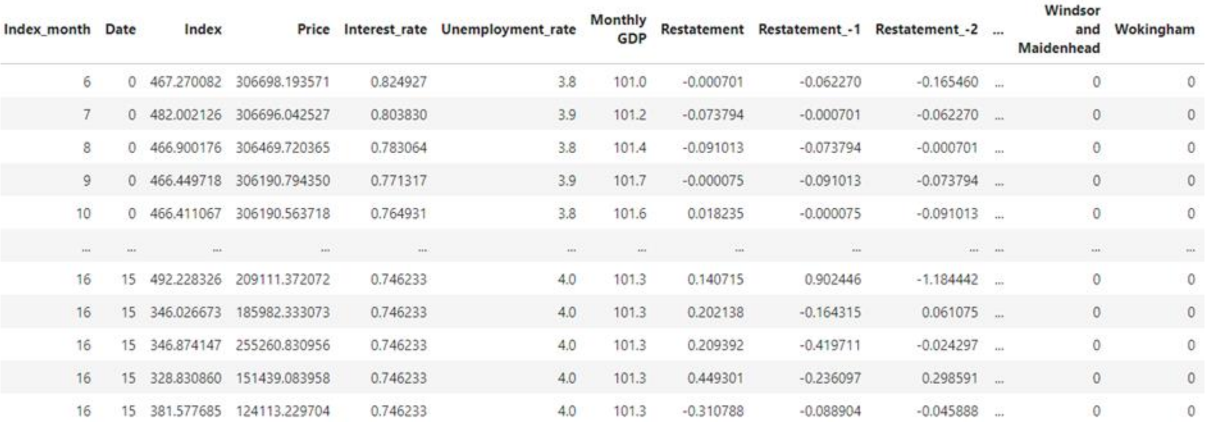
1. ***Label encoding***: Encodes labels of a specific column with an integer value from range 0 to $n - 1$ where n is the number of unique labels.
2. ***One hot encoding***: Creates binary columns for each distinct label and is assigned a value of 1 if the label is True for that row and 0 if False.

These are used depending on the quality of the categorical variable. Label encoding is used for sequential data, being *Index_month* and *Date*. One hot encoding is used for non-sequential columns *Index_Country*, *Index_Region*, *Index_Geography*, *Property_Type*, this significantly increases the size of the dataset as a new column is added based on the total number of unique labels of the one-hot encoded column. This adds a total of 198 columns, table X shows the one-hot encoded variable with the total number of unique labels.

Table 3. 4: Number of unique labels for non-sequential categorical columns

Column	Number of unique labels
Index_Country	5
Index_Region	13
Index_Geography	175
Property_Type	5

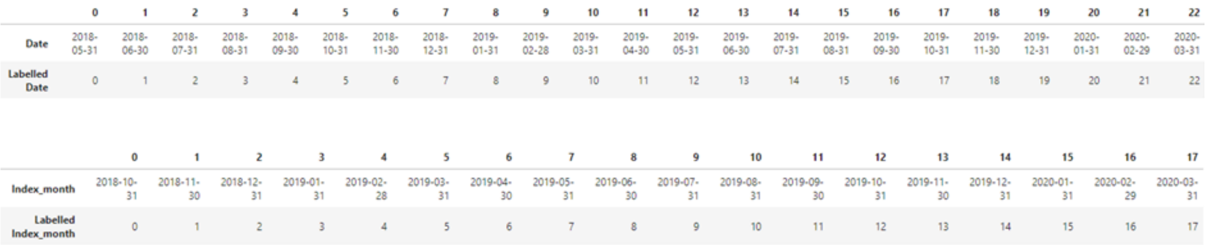
A sample of the resulting dataset is shown in figure 3.6, we see that *Index_month* and *Date* are now numeric values and can be used as a variable to predict, and the new columns *Windsor and Maidenhead* and *Wokingham* are an example of some of the one-hot encoded categorical labels from the *Index_Geography* column which is removed after one-hot encoding.



Index_month	Date	Index	Price	Interest_rate	Unemployment_rate	Monthly GDP	Restatement	Restatement_-1	Restatement_-2	...	Windsor and Maidenhead	Wokingham
6	0	467.270082	306698.193571	0.824927	3.8	101.0	-0.000701	-0.062270	-0.165460	...	0	0
7	0	482.002126	306696.042527	0.803830	3.9	101.2	-0.073794	-0.000701	-0.062270	...	0	0
8	0	466.900176	306469.720365	0.783064	3.8	101.4	-0.091013	-0.073794	-0.000701	...	0	0
9	0	466.449718	306190.794350	0.771317	3.9	101.7	-0.000075	-0.091013	-0.073794	...	0	0
10	0	466.411067	306190.563718	0.764931	3.8	101.6	0.018235	-0.000075	-0.091013	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
16	15	492.228326	209111.372072	0.746233	4.0	101.3	0.140715	0.902446	-1.184442	...	0	0
16	15	346.026673	185982.333073	0.746233	4.0	101.3	0.202138	-0.164315	0.061075	...	0	0
16	15	346.874147	255260.830956	0.746233	4.0	101.3	0.209392	-0.419711	-0.024297	...	0	0
16	15	328.830860	151439.083958	0.746233	4.0	101.3	0.449301	-0.236097	0.298591	...	0	0
16	15	381.577685	124113.229704	0.746233	4.0	101.3	-0.310788	-0.088904	-0.045888	...	0	0

Figure 3. 6: Sample of final dataset with sliding window

Datasets are created to show the label encoded transformations, allowing us to refer to the dates based on the labelled number; these are shown in figure 13. From these, we can tell for example that *Labelled Date* '0' corresponds to '2018-05-31'.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Date	2018-05-31	2018-06-30	2018-07-31	2018-08-31	2018-09-30	2018-10-31	2018-11-30	2018-12-31	2019-01-31	2019-02-28	2019-03-31	2019-04-30	2019-05-31	2019-06-30	2019-07-31	2019-08-31	2019-09-30	2019-10-31	2019-11-30	2019-12-31	2020-01-31	2020-02-29	2020-03-31
Labelled Date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Index_month	2018-10-31	2018-11-30	2018-12-31	2019-01-31	2019-02-28	2019-03-31	2019-04-30	2019-05-31	2019-06-30	2019-07-31	2019-08-31	2019-09-30	2019-10-31	2019-11-30	2019-12-31	2020-01-31	2020-02-29	2020-03-31
Labelled Index_month	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 3. 7: Reference of encoded dates and index months

Index month '2020-03-31' is not used in modelling as we require price from index month '2020-04-30' to calculate the restatements for '2020-03-31'. In addition to this, adding the sliding window creates null values for dates that do not have historic restatements, these dates are removed from the dataset and the final dataset with historic data has 105,512 rows and 219 columns.

3.3. Variation of restatements

Calculating the average time taken for restatements to converge is one of the main objectives of the project as the dates of transactions used as well as the sliding window size depend on it. To complete this objective, new variables are defined, and a criterion is placed that determines our definition of convergence, the general approach taken is that if the change in restatement is less than a threshold in a particular month in the restatement time series for a specific combination of *Date*, *Index_Country*, *Index_Region*, *Index_Geography*, *Property_Type*, then this month is considered to have converged. Multiple conditions are tested and the most appropriate one was selected, below is a table of experiments carried out using different criteria and threshold values.

Table 3. 5: Variation of restatements criterion tested

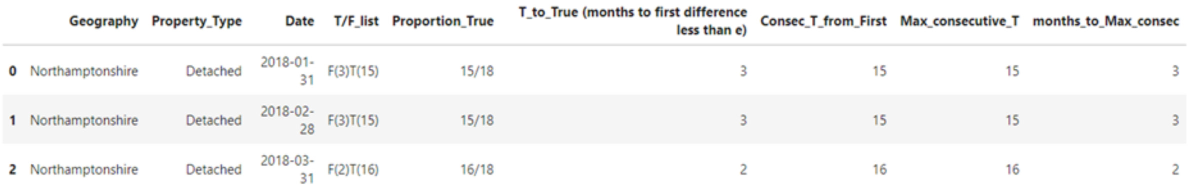
Criterion	**Threshold value e**
$\lvert Restatement_m - Restatement_{m-1} \rvert < e$	$e = 0.5$
$\lvert Restatement_m \rvert < e$	$e = 75th\ percentile\ of\ restatements$ for a specific combination

Using these criteria, the new variables are defined as:

- **T/F_list:** A list of **True** and **false** based on if the criterion is satisfied as well as for how many months the criterion is satisfied.
- **Proportion_True:** A fraction stating the number of months that follow criteria over the total number of months.
- **T_to_true:** The number of months taken until the first month that satisfies the condition appears.
- **Consec_T_from_first:** The number of consecutive months that follow the criteria from the first **True** month.
- **Max_consecutive_T:** The maximum number of consecutive months that satisfy the condition, it is not always the case that it is the first set of consecutive **True** months is the maximum.
- **Months_to_Max_consec:** The number of months taken to get to the maximum number of consecutive **True** months.

The code used to calculate these new variables is found in appendix B.

A subset of unique combinations of *Date*, *Index_Country*, *Index_Region*, *Index_Geography*, *Property_Type* are taken at random and a new dataset is created with the calculation of these variables; an example of this dataset is shown in figure 3.8.



	Geography	Property_Type	Date	T/F_list	Proportion_True	T_to_True (months to first difference less than e)	Consec_T_from_First	Max_consecutive_T	months_to_Max_consec
0	Northamptonshire	Detached	2018-01-31	F(3)T(15)	15/18	3	15	15	3
1	Northamptonshire	Detached	2018-02-28	F(3)T(15)	15/18	3	15	15	3
2	Northamptonshire	Detached	2018-03-31	F(2)T(16)	16/18	2	16	16	2

Figure 3. 8: Sample of dataset used to find average time to convergence

The main column used to calculate the average time taken for restatements to converge is *months_to_Max_consec*, while the others provide validation that the code and calculations are working as intended, they also can provide insight on the convergence of restatements. Calculating the average of *months_to_Max_consec* gives us the average number of months it takes for restatements to have reached the point where the restatements consistently vary insignificantly, at this point we can say that it has converged.

Changing the criteria may have significantly different results, therefore they are tested fairly, and the most appropriate criteria is selected; once this is done, the average time of convergence can be calculated, and this value can be used to determine the dates of transactions used as well as the sliding window size of the dataset.

3.4. Modelling

This section discusses the approach taken to create the data structures, model configurations and design of experiments.

Data structure and train/test split

Multiple data structures and train test splits are tested, these are outlined in the list below and each section is mentioned in detail.

- Data split 1: Train/test split on Index month
- Data split 1 excluding February restatements
- Data split 2: Train/test split on Index month and Date
- EEMD smoothing
- No historic data
- 3-dimensional data structure

Data split 1

The final dataset (shown in figure 3.6) is split into 80% training and 20% testing based on *Index_month* such that the previous 80% of index months are used to predict the next 20%, therefore the data is split so that index months ‘2020-01-31’ and ‘2020-02-29’ are in the test set while months before that are in the training set, using this gives 78,480 rows for training and 27,032 for testing.

Data split 1 excluding February restatements

February is the only index month in the dataset which is affected by COVID since it is the change in price from February to March. A test is carried out to see if keeping February restatements affects the accuracy of the model.

Data split 2

A different train test split is tested where the data is split both on *Index_month* and *Date*. It was made sure that approximately the same proportion of the data is split such that the split sizes match the first train test split; this is important as it keeps the evaluation between different splits fair.

Table 3. 6: Idea of how data split 2 is made

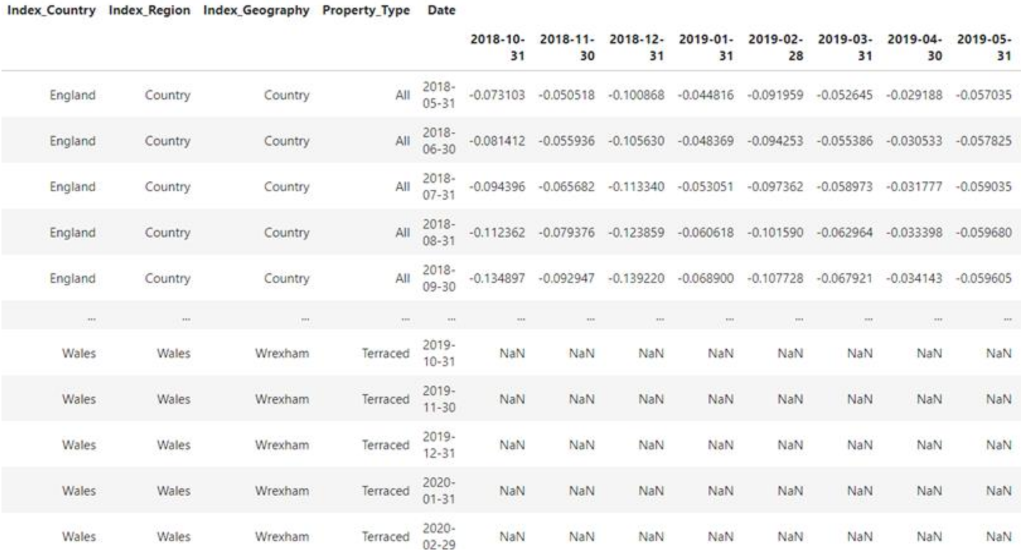
		Date	
		Date < *d*	*d* < *Date*
Index_month	*Index_month* < *m*	Train	Test
	m < *Index_month*	Test	Train

For threshold date *d* and threshold index month *m*, a certain row in the dataset becomes part of the training set if the *Index_month* < *m* and *Date* < *d* or *Index_month* > *m* and *Date* > *d* and is part of the test set if these conditions are not met. This allows for every *Date* and *Index_month* to be seen by the model without training on the whole dataset.

EEMD smoothing

It was shown that a combination of EEMD smoothing and support vector regression yielded the best results when predicting house prices in the U.S (Plakandaras, 2014). In that case, the smoothing algorithm was implemented on a single time series, whereas in our case it is implemented on all unique time series present in the dataset. The implementation of EEMD smoothing on this dataset is discussed in this subsection.

The index dataset can be processed to represent each row as a time series by grouping the data by *Date*, *Index_Country*, *Index_Region*, *Index_Geography*, *Property_Type* and then using the unstack function from pandas on *Index_month* to transform each index month into a column with the respective restatement, an example of this dataset is shown in figure 3.9.



Index_Country	Index_Region	Index_Geography	Property_Type	Date								
					2018-10-31	2018-11-30	2018-12-31	2019-01-31	2019-02-28	2019-03-31	2019-04-30	2019-05-31
England	Country	Country	All	2018-05-31	-0.073103	-0.050518	-0.100868	-0.044816	-0.091959	-0.052645	-0.029188	-0.057035
England	Country	Country	All	2018-06-30	-0.081412	-0.055936	-0.105630	-0.048369	-0.094253	-0.055386	-0.030533	-0.057825
England	Country	Country	All	2018-07-31	-0.094396	-0.065682	-0.113340	-0.053051	-0.097362	-0.058973	-0.031777	-0.059035
England	Country	Country	All	2018-08-31	-0.112362	-0.079376	-0.123859	-0.060618	-0.101590	-0.062964	-0.033398	-0.059680
England	Country	Country	All	2018-09-30	-0.134897	-0.092947	-0.139220	-0.068900	-0.107728	-0.067921	-0.034143	-0.059605
...	...	...	...	...	...	...	...	...	...	...	...	...
Wales	Wales	Wrexham	Terraced	2019-10-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Wales	Wales	Wrexham	Terraced	2019-11-30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Wales	Wales	Wrexham	Terraced	2019-12-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Wales	Wales	Wrexham	Terraced	2020-01-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Wales	Wales	Wrexham	Terraced	2020-02-29	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 3. 9: Sample of dataset of restatement time series

Plotting the first row of restatements results in the time series for all transactions in ‘England’ on a country level, for property type ‘All’ on date ‘2018-05-31’. This is shown in figure 3.10.

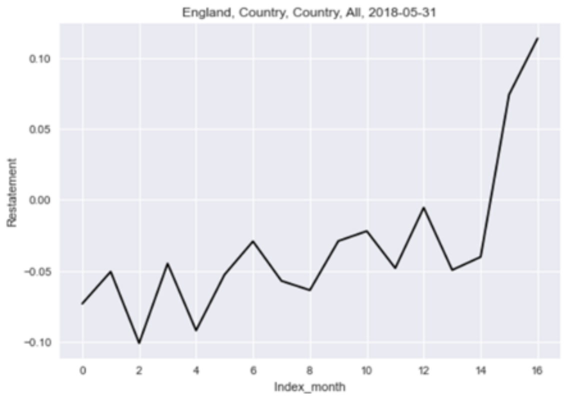


Figure 3. 10: Example of the time series of a restatement found in the dataset

The time series are smoothed by summing the intrinsic mode functions to obtain the general dynamic of the time series as done by (Plakandaras, 2014). Figure 3.11 shows the decomposition of the time series in figure 3.10 using the EEMD algorithm, and figure 3.12 visualizes the result of summing eIMF 2 and eIMF 3 in comparison to the actual time series.

We see that it does smooth the data significantly however it does not accurately represent the data, for example, the restatement at *Index_month* 0 is significantly lower when smoothed, shown by figure 3.12.

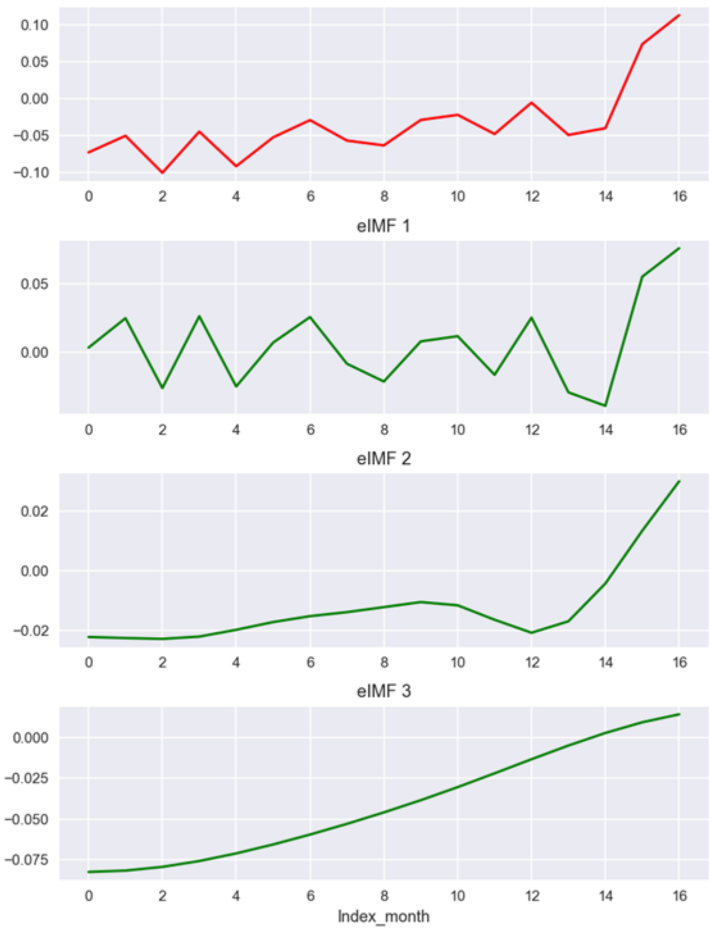


Figure 3. 12: Example of EEMD smoothing on a restatement time series

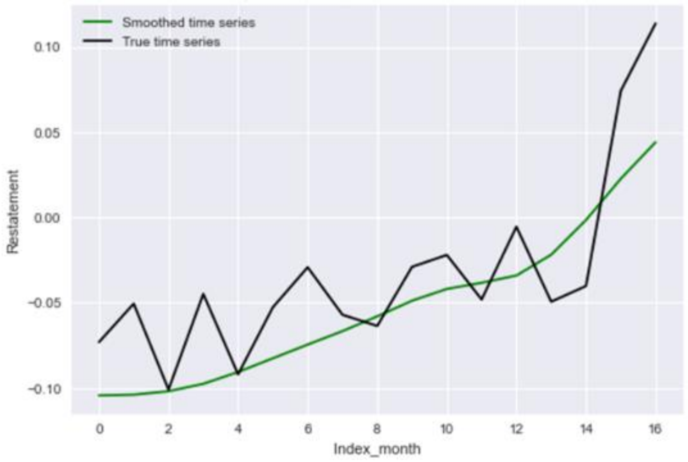


Figure 3. 11: Resulting time series from summing eIMFs

This is repeated with the first 5 time series to see if the same issue occurs. From figure 3.13 we see that all of them suffer from the same issue. The calculation of the smoothed time series as seen in figure 3.13 can be written as,

$$Smoothed\ Restatement_m = eIMF_{m,2} + eIMF_{m,3},$$

Where $eIMF_{m,i}$ is the i^{th} decomposition for the time series m using the EEMD algorithm.

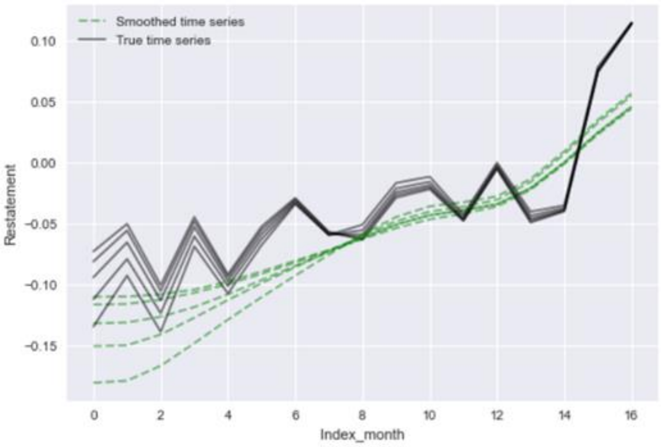


Figure 3. 13: EEMD smoothed time series for random sample of 5

To make the smoothed series more accurate, a linear combination of eIMFs are chosen instead of simply summing them. This can be written as

$$Smoothed\ Restatement_m = x \times eIMF_{m,2} + y \times eIMF_{m,3},$$

Where x and y are random variables. Setting $x = 1, y = 0.5$ results in the smoothed series shown in figure 3.14.

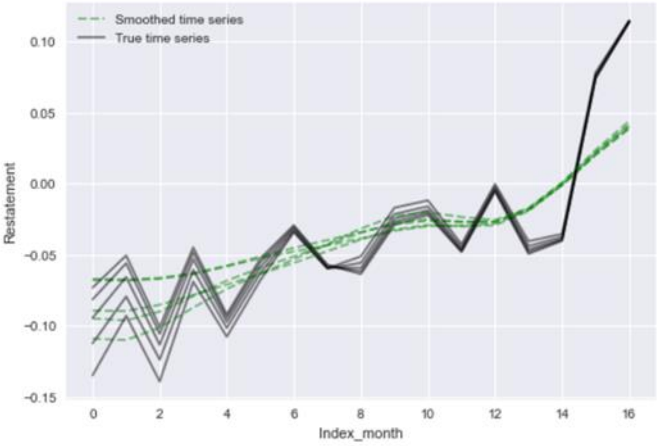


Figure 3. 14: EEMD smoothing with linear combination of eIMFs

It was found that the length of the time series affects the number of eIMFs that can be produced using EEMD; we can't expect to smooth a time series consisting of 2 points as it is simply a straight line connecting them. To take these cases into account, it was decided that EEMD would be applied time series that contained at least 6 restatements, this can be described by the equation below.

$$Smoothed\ Restatement_m = \begin{cases} q \times eIMF_{m,1}, & if\ eIMF_{m,2} = NULL \\ w \times eIMF_{m,1} + v \times eIMF_{m,2}, & if\ eIMF_{m,3} = NULL \\ x \times eIMF_{m,2} + y \times eIMF_{m,3}, & else \end{cases} \quad (3)$$

This approach is taken and adapted to create a dataset exactly like figure 3.6, however with the smoothed restatements instead of the actual ones. The code used to apply EEMD smoothing to the data and adapt the smoothed restatements into the correct structure can be found in appendix C.

No historic data

A data structure is created where there is no sliding window added, this means that the model is trained without historic data. By not including historic data, the dataset is significantly bigger as including historic data removed the dates with no historic or not enough historic data. This structure aims to test the predictability of models without previous restatements and is made simply by not creating the sliding window described in section 3.2.2.2.

3-dimensional data structure

Recurrent neural networks take 3-dimensional data as input, where the dimensions are the number of samples, number of time steps and number of features. Since this structure is not required for SVR and MLP, the data is transformed right before RNN and LSTM training. The structure is created by converting the datasets into NumPy arrays and then using the reshape function to add the 3rd dimension. This dimension represents sequence length and is set to 1. This approaches the structure alternatively as the sequence of historic data are still columns in the dataset and not part of the 3rd dimension.

Model implementation and optimization

Table 3.7 is made to show the design of experiments, this helps simplify the process and guide how to proceed with experiments. The first experiment tests out the different data structures for the SVR model, the second experiment compares the models' performance based on the structures from experiment 1. The 3rd experiment evaluates the models' performance without historic data and the final experiment compares the optimal model from experiment 3 with a transfer learning model. The no historic data and 3D data models are trained with the data split that performs best.

Table 3. 7: Design of experiments

Experiment	Models	Structure
1	SVR	Data split 1
		Data split 1 excluding Feb
		Data split 2
		EEMD smoothing
2	SVR	Best structure from experiment 1
	MLP	Best data split from experiment 1
	RNN	
	LSTM	
3	SVR	Best structure from experiment 1 with no historic data

	MLP	Best data split from experiment 1 with no historic data
	RNN	
	LSTM	
4	Best model from experiment 3	Best data split from experiment 1 with no historic data
	TL	

Support Vector Regressor (SVR)

The support vector regressor model was made using the sci-kit learn library python offers. A grid search is carried out to optimize the following hyperparameters of the SVR shown in table 3.8. Once the search is over, these hyperparameters are used to train all SVR models, they are then evaluated and the data structure with the best performance is used to train the neural networks.

Table 3. 8: SVR hyperparameter description

Hyperparameter	Description
C	Regularization parameter to balance model complexity and approximation error
Kernel	A function applied to transform the dataset and help fit hyper
Gamma	Coefficient that determines the influence of each training sample
Epsilon	Region around hyperplane where no penalty

The SVR model was built using a python library called sci-kit learn, and the grid search was carried out using halving grid search where parameter combinations are tested on a subset of the data and the top 50% of parameter combinations are tested again on a larger subset. This cycle continues until the best hyperparameter combination is found; using halving grid search has been shown to significantly reduce training times (Gilde, n.d.). The hyperparameter search space and time taken are discussed in the results section.

Neural Networks

Long short-term memory neural networks have been shown to perform well for time series forecasting tasks, it is quite a complex model and therefore simpler models are tested and the complexity is built up to LSTM by creating an MLP and a simple RNN before the LSTM.

All neural networks are optimized using a library called Talos, which allows for easy optimization of parameters and provides a wide variety of tools such as analysis of performance based on hyperparameters. The network hyperparameters to be optimized using Talos are:

Table 3. 9: Neural network hyperparameter description

Hyperparameter	**Description**
First neuron	Number of neurons in first hidden layer.
Optimizer	Algorithm used to adjust weights and biases of network.
Learning rate	Determines step size of optimization algorithm.
Activation function	Function applied to neuron.
Batch size	Number of samples processed each epoch.

Multilayer Perceptron

The MLP is initialized using the python libraries Keras and TensorFlow, the hyperparameters mentioned previously are tested as well as additional ones being:

Hyperparameter	**Description**
Hidden layers	Number of additional hidden layers
Shape	Shape of network with hidden layers, i.e. number of neurons in hidden layers.

These are not mentioned in table 3.9 as they are not optimized for the RNN and LSTM. An overview of the MLP structure can be seen in figure 2.3, where the number of hidden layers and neurons are to be determined from the hyperparameter optimization.

Recurrent neural network and Long-short term memory network

Both RNN and LSTM follow a structure such that the input layer is comprised of a certain number of neurons that connect to a dense layer with the same number of neurons, this dense layer is then fully connected to the output layer which contains one neuron.

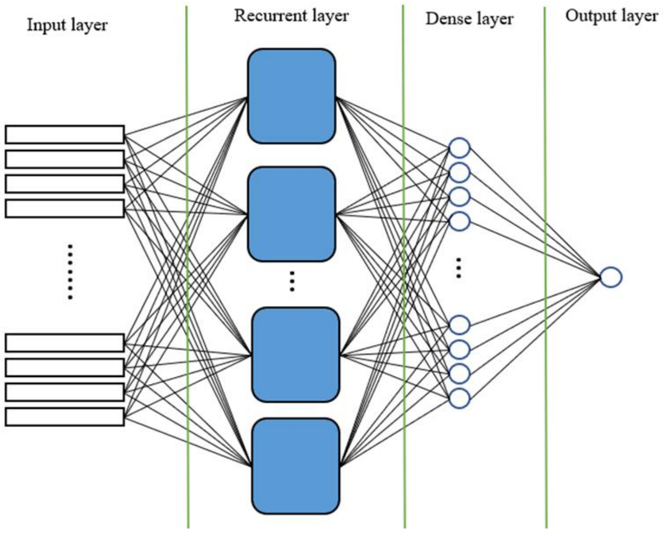


Figure 3. 15: RNN and LSTM network structure

This is visualized in figure 3.15; the number of inputs corresponds to the total number of columns used as a predictor in the dataset, while the number of neurons in the recurrent and dense layer is optimized. The type of neuron in the recurrent layer is a simple recurrent neuron for the RNN and an LSTM neuron for the LSTM model.

Transfer learning structure

Transfer learning aims to incorporate trends found in historic data to predict the restatements for dates with no historic data available. The model that performs best with historic data is used as the base model for transfer learning, the historic data has fewer columns available as predictors compared to the dataset with historic data, where the difference in the number of columns is the size of the sliding window. To preserve as much of the best network found as possible, the changes in the transfer learning model are minimized.

The difference in the number of columns requires a new input layer to be added with the number of neurons being the number of columns in the no historic dataset, this input layer connects to a dense layer with the number of nodes being the total number of columns in the data with the sliding window, this is then connected to the base model. Figure 3.16 visualizes the transfer learning model structure.

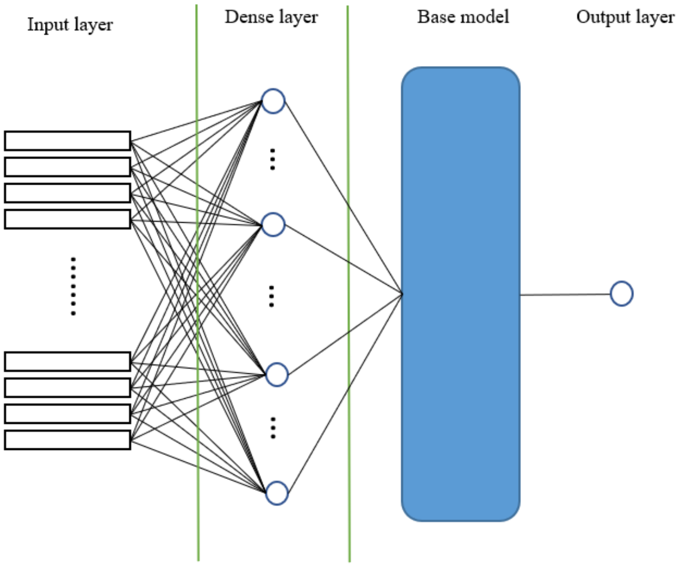


Figure 3. 16: Transfer learning network structure

The transfer learning model is optimized using Talos and the same hyperparameters as LSTM are optimized with the only additional hyperparameter is trainable which determines if the base model can be tuned or not.

3.5. Evaluation

The selected models are compared based on generic model metrics for regressions tasks. Root mean squared error is the main metric used to evaluate models and is described by equations 4 below,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}, \tag{4}$$

Where $i = variable$, $N = number\ of\ datapoints, x_i = true\ observations$ and $\hat{x}_i = predicted\ value.$

The evaluation of how well a model predicts can be subjective based on what our definition of a good prediction is. To judge this, the best model is compared with several baseline estimates of what the restatement at a given point in time is; these estimates are chosen based on what may be obvious, such as the average previous restatements for specific geography and property type. The strength of these estimates may vary greatly, therefore multiple are tested and the most reliable ones are used to determine how well we can predict restatements, in turn answering the main research question.

4. Results

4.1. Data exploration

Consistency in Index data

The exploration started with ensuring the index data is consistent and not missing values from important columns. Heat maps were created to visualize the distribution of restatement counts when grouped based on *Index_month* and *Date*, shown in figure 4.1. There are 872 unique combinations of *Index_Country*, *Index_Region*, *Index_Geography* and *Property_Type*, this is reflected in figure 4.1 where each value is divisible by 872 and doing so results in the number of months available for that specific *Index_month* or *Date*, for example, *Index_month* ‘2018-10-31’ provides the restatements for $\frac{5232}{872} = 6$ unique *Date*’s being ‘2018-05-31’, ‘2018-06-30’, ‘2018-07-31’, ‘2018-08-31’, ‘2018-09-30’ and ‘2018-10-31; there are no later dates as those transactions have technically not occurred yet.

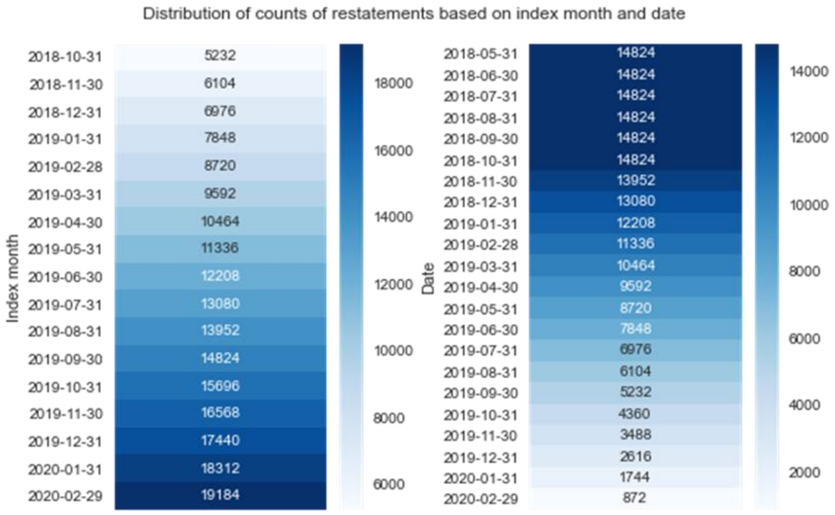


Figure 4. 1: Heat map of count of restatements based on Index_month or Date

Figure 4.1 implies the data is consistent and therefore the distribution of the restatement count based on both *Index_month* and *Date* can be visualized as seen in figure 4.2.

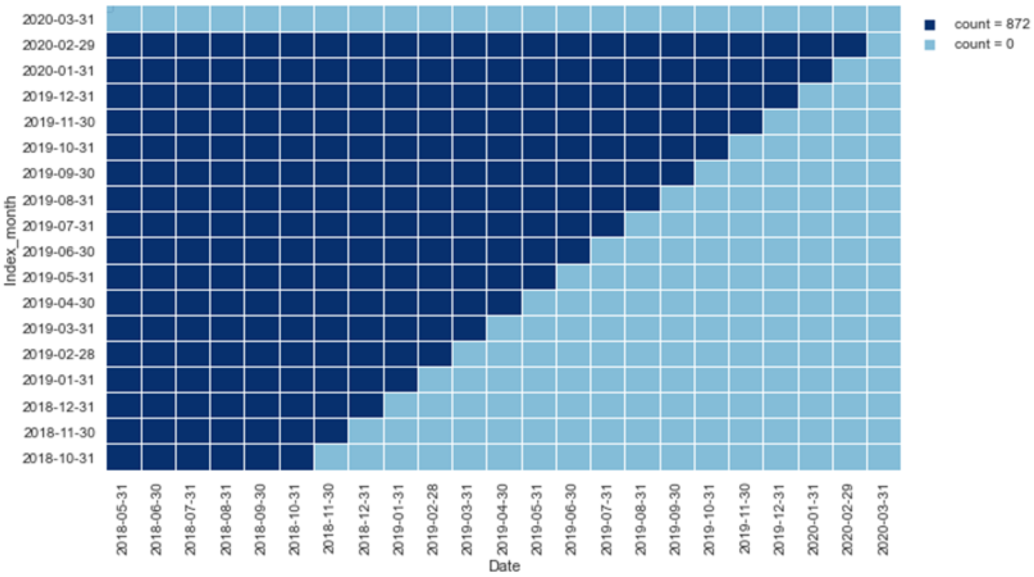


Figure 4. 2: Heat map of count of restatements based on Index_month and Date

This helps us understand the change in restatement counts based on both *Index_month* and *Date*; figure 4.2 implies that as *Index_month* increases, the total count of restatements that index month increases, while as *Date* increases the count decreases. An alternate way of understanding this mathematically is for a transaction that occurred on date d, the restatements of this date will only be available in index months m where $m = d + i$ for all $i \in \mathbb{N}_0$. Whereas for *Index_month* m, the restatements will only be available for date d where $d = m - i$ for all $i \in \mathbb{N}_0$. In other words, a date will only be in an index month from the date the transaction occurred and onwards, whereas an index month will only have transactions from the time of the index month and earlier.

Relationship between change in volume and restatement

The relationship between restatement and volume is investigated using the index and volume data. The volume for a specific combination is defined as the total number of transactions that occurred; it is

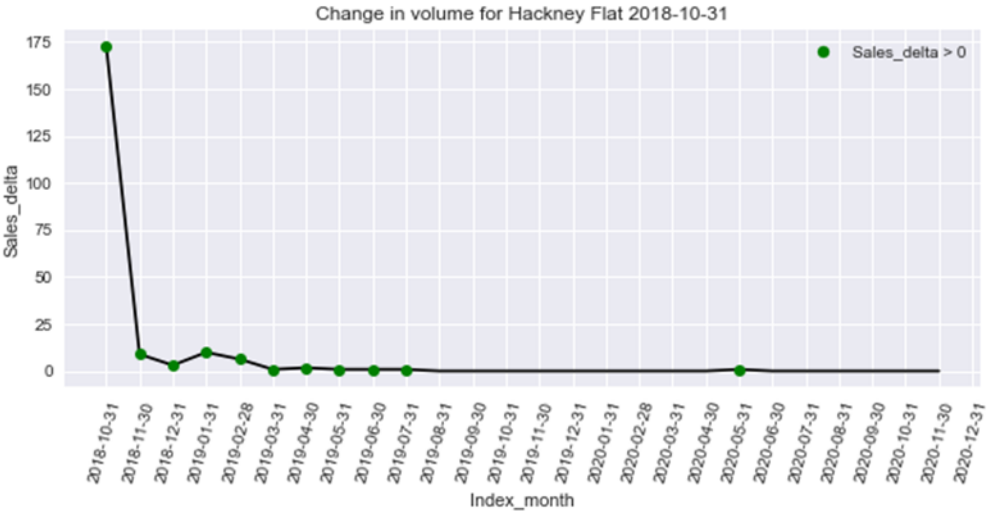


Figure 4. 3: Graph of change in volume for 'Hackney' 'Flat' '2018-10-31'

strictly increasing as the total number of transactions can only increase from the date of the transaction. Therefore, the volumes converge to a specific value after a certain number of months, this indicates that at some point we have all the transactions that occurred on a specific date. Figure 4.3 shows the change in volume for *Index_Geography* ‘Hackney’, *Property_Type* ‘Flat’ and *Date* ‘2018-10-31’ where *Sales_delta* refers to the percentage change in volume.

As expected, the volume change is biggest in the first month and gradually decreases, this is likely the case for all volume series data. Green dots mark where the change is greater than 0, this is done to make it easier to notice where there is a minuscule change compared to when there is no change.

It is expected that for index month m, $Restatement_m \neq 0$ if and only if $Volume_m > Volume_{m-1}$, i.e. there will be a change in price, if and only if there is an increase in the total number of transactions, otherwise we expect $Price_m = Price_{m-1}$ as $Price$ is an aggregation of true sale values that occurred for a specific combination of geography, property type, etc.

Figure 4.4 visualizes the restatement and change in volume for ‘Hackney’, ‘Flat’, ‘2018-10-31’. A double y-axis line graph is plotted to ensure changes in the restatement are noticeable. Since the restatement values are small in comparison to the change in volume, no change in restatement would be visible if they shared a y-axis. Figure 4.4 implies changes in the *Price* can still occur even when there is no change in volume, which is likely due to the smoothing methodology MIAC implements as we would expect $Restatement_m = 0$ if $Sales\_delta_m = 0$. The volume data used contains months up to ‘2020-12-31’ while the index data only goes up to ‘2020-03-31’, hence the cut-off in restatement halfway through the graph.

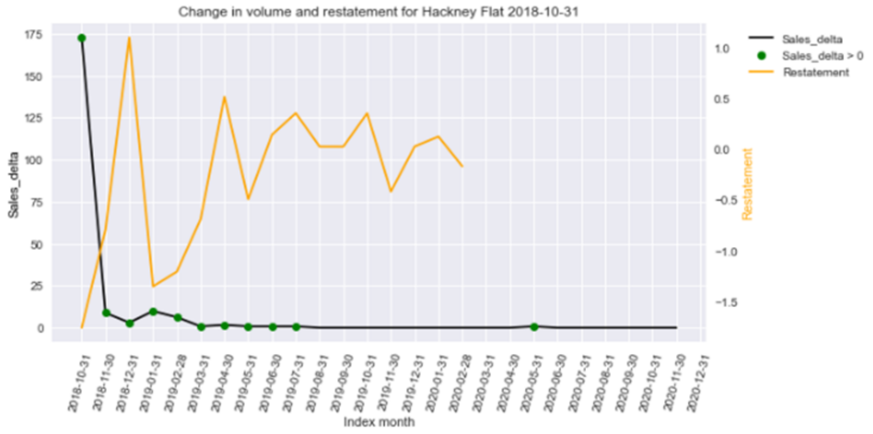


Figure 4. 4: Change in volume and restatement for 'Hackney' 'Flat’ ‘2018-10-31’

It is difficult to say for certain if this is always the case as there are 872 unique combinations to go through, another less reliable way is to investigate other combinations at random to see if they display the same property. This is done for 3 different combinations, shown in figure 4.5, and the same quality is seen especially for the combination ‘Hounslow’ ‘Semi’ ‘2018-12-31’ where there is no change in volume by the 4th month and it is seen continuously that $Restatement_m \neq 0$. While this is the case, Restatements tend to fluctuate around 0 after a certain period.

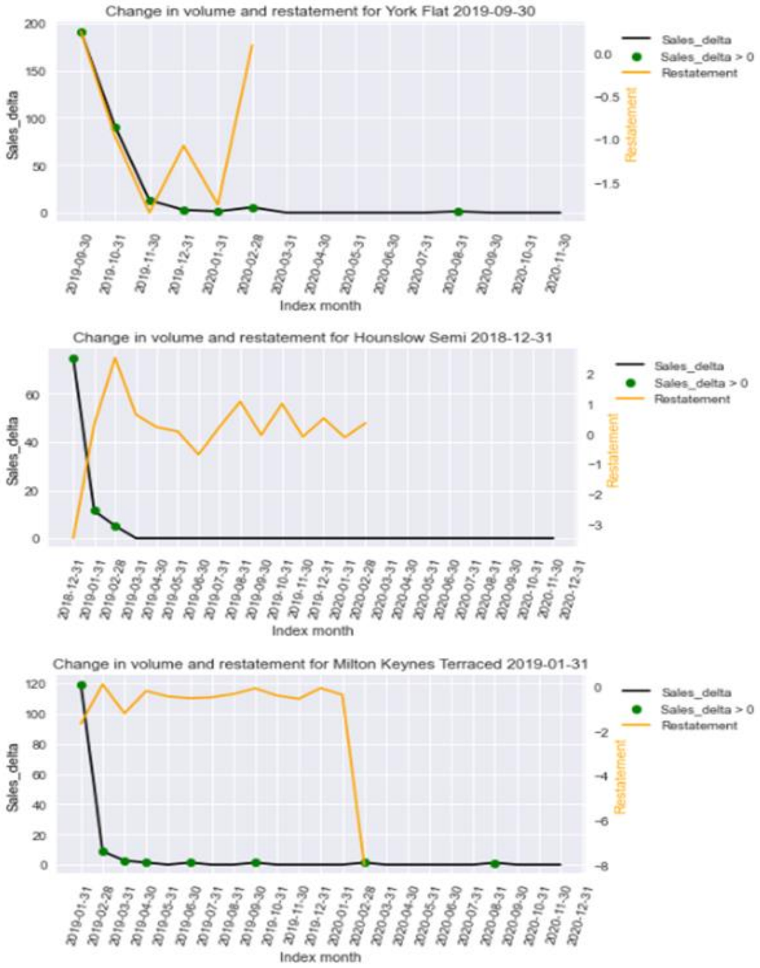


Figure 4. 5: Change in volume and restatement for multiple geographies

Further exploration of Restatements

The next step of the exploration was to try and discover qualities of restatements. A function is created that takes a specific *Index_Geography* and *Property_Type* and returns a graph of the time series of all dates for that combination; this allows for visualizations of different combinations to be plotted easier. Figure X displays the output of this function; a correlation heatmap of the restatements is also plotted.

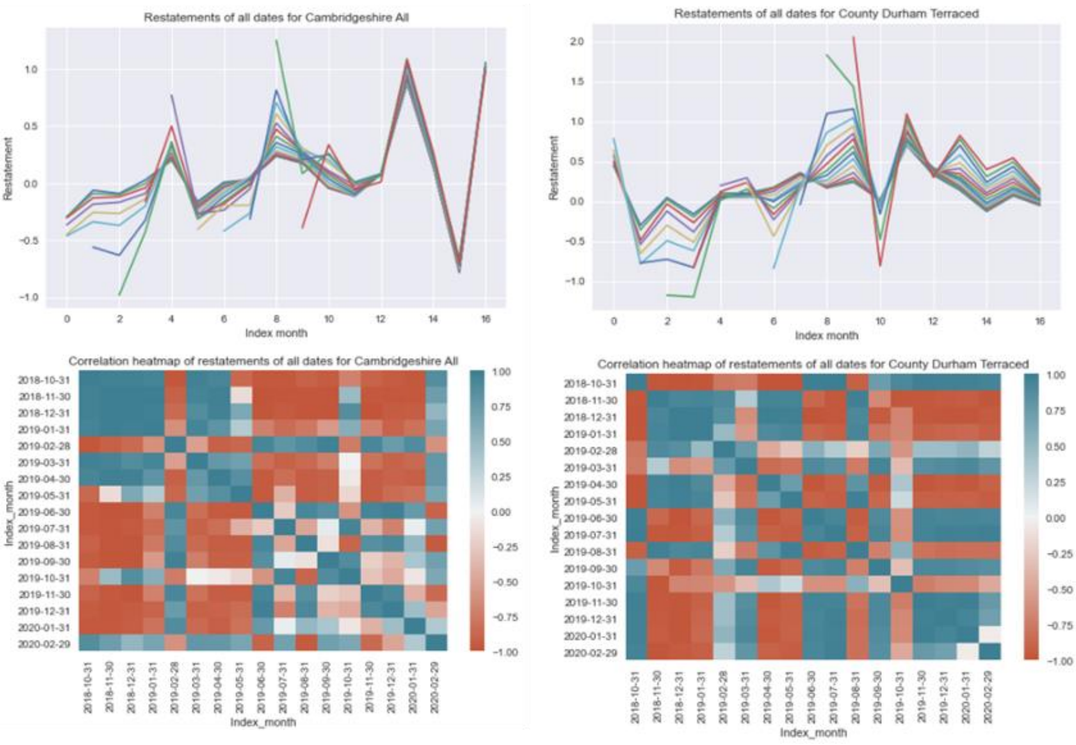


Figure 4. 4: Restatement plot and correlation heat map

The function is used on two random combinations, both of which imply that restatements are generally heavily correlated not only with historic restatements but also restatements of different dates. This is counter-intuitive as we expect restatements of transactions that occurred on different dates to be independent of each other; how can the change in price of a specific geography and property type on *Date* d be directly correlated with the change in price of that combination at time $d - 15$? This implies the price of the transactions that occurred at $d - 15$ are still changing by almost the same amount as the price value at d.

The most probable reason for this relation is the application of matched samples in the interpolation methodology. The assumption made by matched samples is that two properties with the same postcode and property type are sufficiently homogeneous, and therefore their values are approximately equal, given that there is no mention of date being a factor of grouping samples, it is very possible samples of different dates are matched.

The high correlation is unexpected but may not necessarily be a bad thing, it may allow for restatements of different dates to be a highly significant feature for prediction. Evaluating all correlations of restatement time series in the dataset for all combinations would provide a better overview instead of evaluating random samples, such as calculating the 'mean' correlation. This is a difficult task as the range of correlation is $[-1,1]$ and summing the correlations to calculate the mean will likely reflect a misleading relationship.

Restatements are also visualized using histograms and boxplots. Figure 4.7 displays histograms for scaled *Price* and *Restatement*. The prices appear to be positively skewed while the restatements visually resemble a normal distribution. The range of x-axis values is limited so that a clearer view of the histograms is seen.

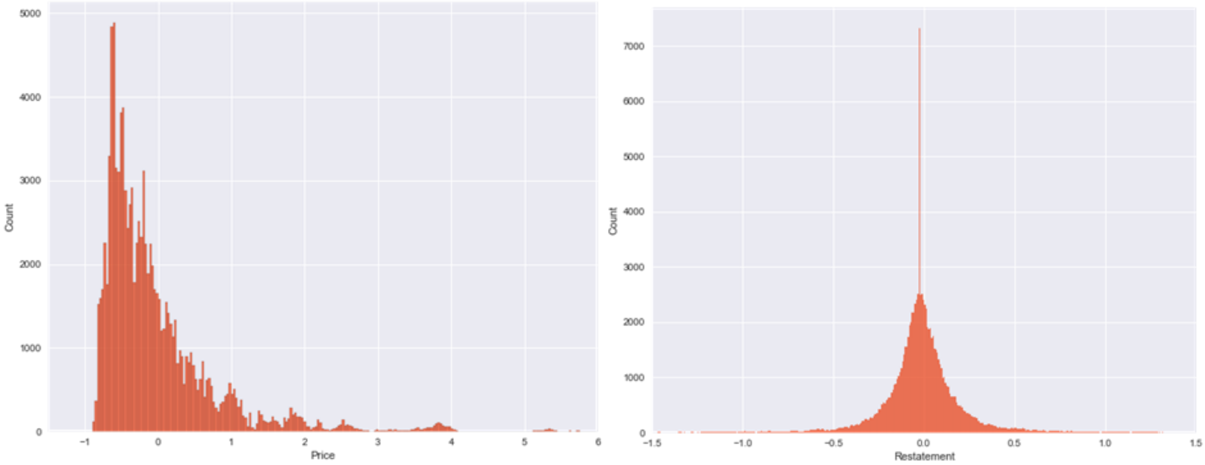


Figure 4. 5: Histogram of price and restatement

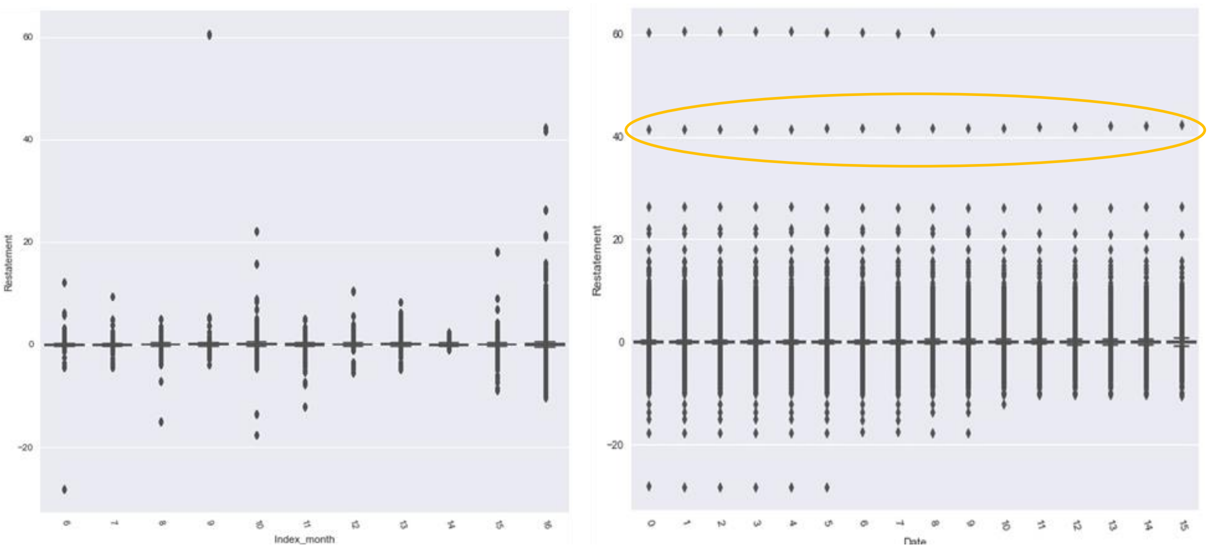
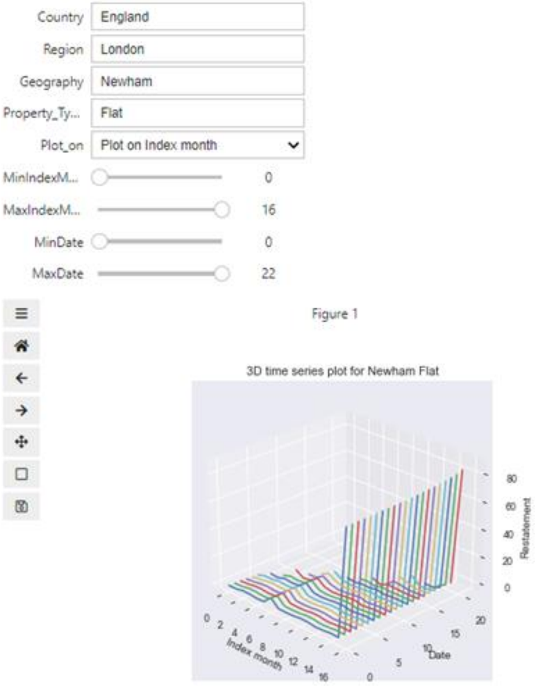


Figure 4. 6: Box plots of restatement based on date and index month

A significant portion of the restatements is considered outliers, as seen by the boxplots in figure 4.8. The boxplots of the restatements by date display an interesting structure where outliers are aligned and may be indirectly grouped by their corresponding time series for a specific index month; an example is highlighted on the graph by the orange circle. This may imply that the whole time series for a combination is considered an outlier for a certain index month.

To see if this is the case, the data is filtered by a range of restatement values based the outliers in the orange circle, being [40,50]. Filtering by this range results in the time series for ‘England’, ‘London’, ‘Newham’, ‘Flat’ for *Index_month* ‘2020-02-29’. This is likely not always the case that the whole time series is considered an outlier, therefore removing outliers will disturb the smoothness and continuity of the time series. Another option would be to remove all time series with at least one outlier, applying this condition results in a dataset of 55,110 rows which is small considering data complexity and the process of splitting into training, testing and validation for modelling; for this reason, outliers are not removed.

To demonstrate the dependency of restatements based on two variables that represent time, being *Date* and *Index_month*, an interactive 3D plot is created that plots the restatements for a specific combination over both *Date* and *Index_month*. A screenshot of this is shown in figure 4.9; the interactive aspect of the plot allows for different categories to be inputted, selecting different ranges for *Date* and *Index_month* as well as rotating the plot on any axis.



Country: England
Region: London
Geography: Newham
Property_Ty...: Flat
Plot_on: Plot on Index month
MinIndexM... 0
MaxIndexM... 16
MinDate 0
MaxDate 22

Figure 4. 7: Interactive 3D plot

The interaction also provides the option to plot the restatements based on *Index_month* or *Date*.

Figure 4.10 shows the difference when plotting on *Index_month* and *Date*.

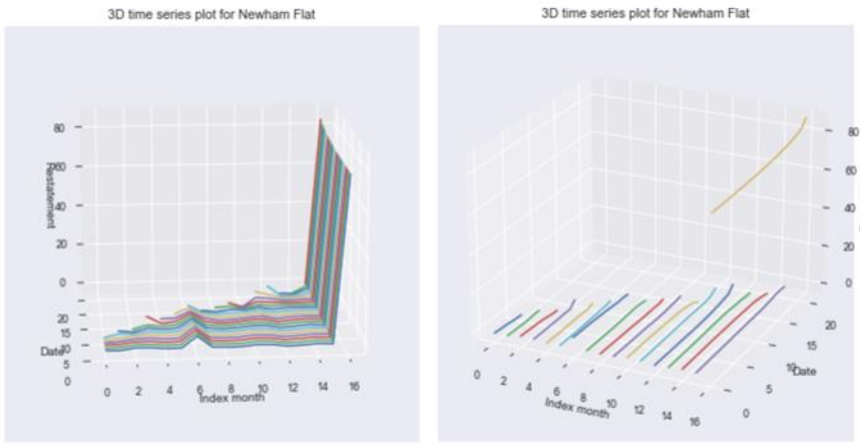


Figure 4. 8: Restatement plotted on index month and date

It shows that for all *Dates*, the restatement of *Index_month* '2020-02-29' is extremely high and is an outlier, however looking at it when plotted by *Date*, we see a different view of the February index. This also helps understand the difference in plotting by *Date* and *Index_month* as well as the boxplots shown in figure 4.8.

Macroeconomic Data

Once an understanding of the restatements was achieved, principal component analysis was carried out on the Bloomberg data to reduce the number of columns. A function was created takes the scaled Bloomberg data and number of principal components and carries out principal component analysis on the dataset, returning the cumulative explained variance of all n components as well as the dimensionally reduced dataset. This function is applied to *SONIA* and *RPI* of the Bloomberg data as they contain many columns. The initial number of components selected is 3, and the cumulative explained variance of the data is shown in table 4.1.

Table 4. 1: Cumulative explained variance of PCA

Data	Principal component 1	Principal component 2	Principal component 3
Zero Coupon RPI	0.60233403	0.97702063	0.99528581
SONIA	0.82813043	0.95586568	0.97814393

Approximately 99% of *RPI* and 97% of *SONIA* can be explained using 3 components, this is satisfactory as adding more components may increase the dataset size significantly without being useful.

Plots of the non-Bloomberg macro-economic data are made before filtering the dates to ensure they are continuous, and the pre-processing is correct. We see from figure 4.11 the large drop in *GDP* in early 2020; this likely corresponds to when countries started having lockdown because of COVID19.

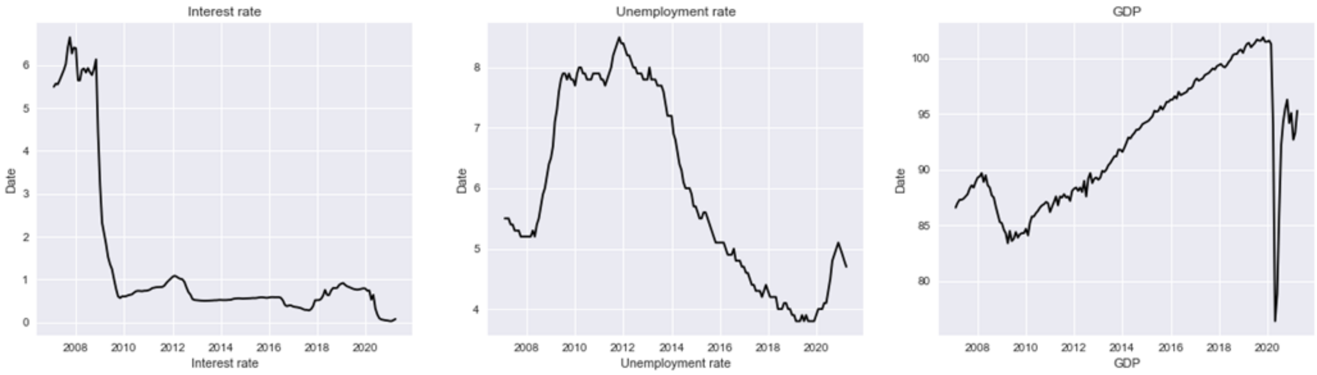


Figure 4. 9: Graphs of non-Bloomberg data

This lowest GDP occurs on date ‘2020-04-30’ and somewhat validates the date range used in the index data to avoid including months affected by COVID. It is important to ensure the macro-economic data is merged on *Index_month* and not *Date* when merging with the index data. Since restatements are the change based on *Index_month* if the data is merged on *Date*, it implies that the macro-economic data are dependent on *Date* and not *Index_month*, therefore when looking at the data of the time series of a specific combination, all the macro-economic variables will have the same value each index month.

Filtering the macro-economic data and replotting the macro-economic data results in graphs shown in 4.12, we see the range is significantly smaller, and there is still a significant drop at ‘2020-03-29’. This hints that restatements of month ‘2020-02-29’ may be affected by COVID19; with this in mind, models are tested with and

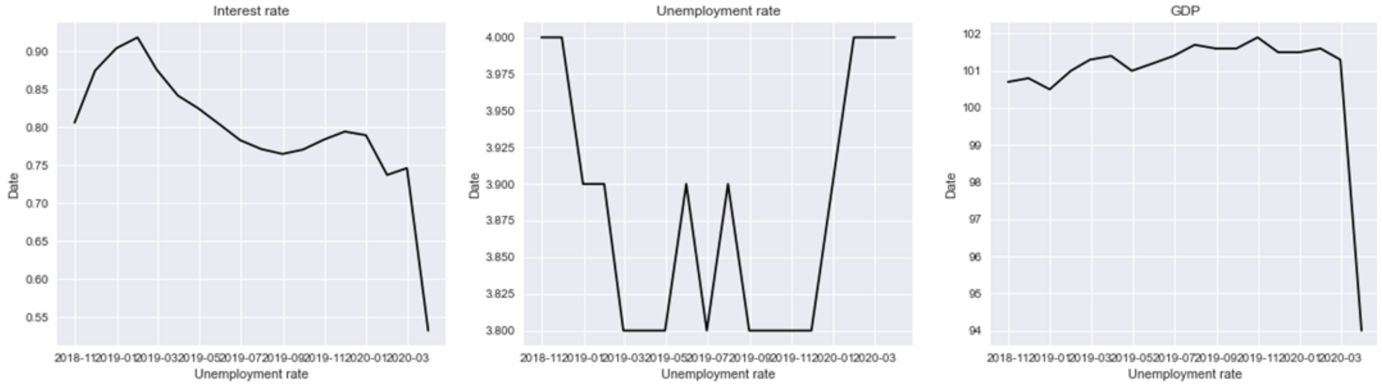
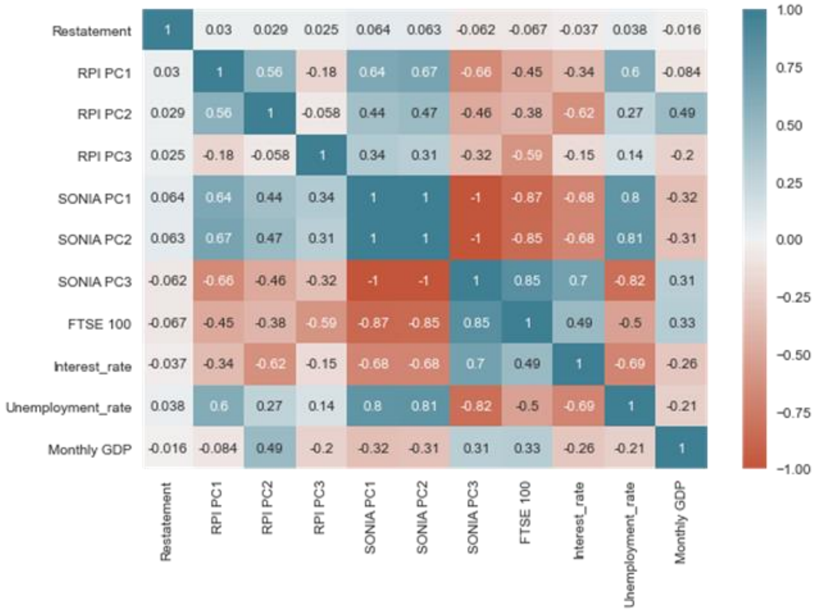


Figure 4. 10: Graphs of non-Bloomberg data with filtered dates

A correlation heatmap is created to visualize the correlation of macro-economic data with restatement, figure 4.13. There appears to be weak correlations of macro-economic data with *Restatement* which may suggest that the macro-economic data may not be good predictors of restatements.



	Restatement	RPI PC1	RPI PC2	RPI PC3	SONIA PC1	SONIA PC2	SONIA PC3	FTSE 100	Interest_rate	Unemployment_rate	Monthly GDP
Restatement	1	0.03	0.029	0.025	0.064	0.063	-0.062	-0.067	-0.037	0.038	-0.016
RPI PC1	0.03	1	0.56	-0.18	0.64	0.67	-0.66	-0.45	-0.34	0.6	-0.084
RPI PC2	0.029	0.56	1	-0.058	0.44	0.47	-0.46	-0.38	-0.62	0.27	0.49
RPI PC3	0.025	-0.18	-0.058	1	0.34	0.31	-0.32	-0.59	-0.15	0.14	-0.2
SONIA PC1	0.064	0.64	0.44	0.34	1	1	-1	-0.87	-0.68	0.8	-0.32
SONIA PC2	0.063	0.67	0.47	0.31	1	1	-1	-0.85	-0.68	0.81	-0.31
SONIA PC3	-0.062	-0.66	-0.46	-0.32	-1	-1	1	0.85	0.7	-0.82	0.31
FTSE 100	-0.067	-0.45	-0.38	-0.59	-0.87	-0.85	0.85	1	0.49	-0.5	0.33
Interest_rate	-0.037	-0.34	-0.62	-0.15	-0.68	-0.68	0.7	0.49	1	-0.69	-0.26
Unemployment_rate	0.038	0.6	0.27	0.14	0.8	0.81	-0.82	-0.5	-0.69	1	-0.21
Monthly GDP	-0.016	-0.084	0.49	-0.2	-0.32	-0.31	0.31	0.33	-0.26	-0.21	1

Figure 4. 11: Correlation heatmap of macro data with restatement

4.2. Variation of restatements

Criterion 1

The initial test to calculate the average time taken for restatements to converge follows the criteria,

$$|Restatement_m - Restatement_{m-1}| < e \text{ for } e = 0.5.$$

Filtering the data within the range of [-0.5,0.5] shows that approximately 90% of the restatements are within this range; with this in mind, the range is selected as the initial threshold to be tested. A random subset of 15 geographies and 15 dates are selected and used to calculate the average time to convergence. Figure 4.14 displays the histogram of the time taken for the whole sample,

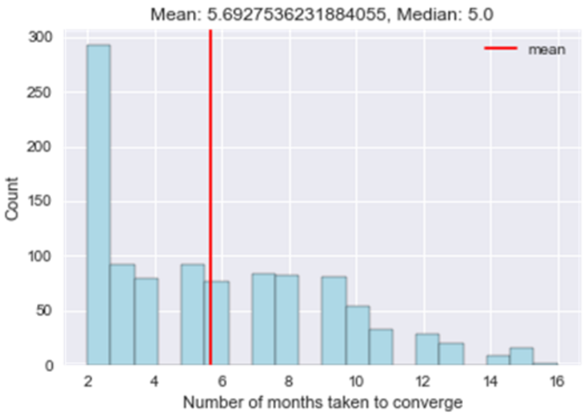
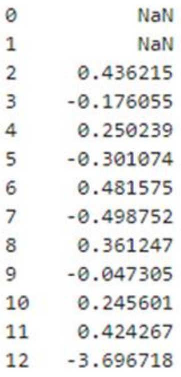


Figure 4. 12: Time taken for months to converge using first criteria

the average time is around 5.69 months, and the median is 5. The histogram appears to be positively skewed with a suspiciously high count of 2 months to convergence. This is suspicious because the second-highest count is around 100 while the count for 2 months is over 250.

This is investigated by applying $|Restatement_m - Restatement_{m-1}|$ for any time series and viewing the output. A screenshot of the output is displayed in figure 4.15, it is seen that the first 2 values are null, and this is because we are finding the change in the change of a value. Since finding the change in something requires an adjacent value, calculating the change in the first or last value results in a null as there is no value to compare it to.

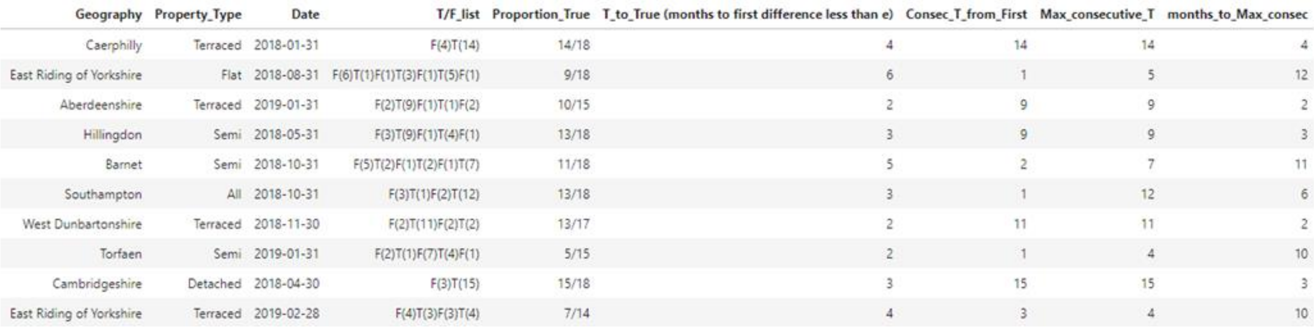


```
0          NaN
1          NaN
2     0.436215
3    -0.176055
4     0.250239
5    -0.301074
6     0.481575
7    -0.498752
8     0.361247
9    -0.047305
10    0.245601
11    0.424267
12   -3.696718
```

Figure 4. 13: Investigating results from first criterion

This causes a huge bias to the average as all the calculations of time to convergence. It would be assumed that because of this, we would find in the histogram the minimum time taken to convergence to be 2, as the actual minimum time possible is 0 and since the first 2 are null values it is $0 + 2 = 2$, and this is true as seen by the histogram.

A screenshot of a random sample from the resulting dataset using these criteria is shown in figure 4.16. The values in each column can be compared to determine if the calculations carried out are correct.



Geography	Property_Type	Date	T/F_list	Proportion_True	T_to_True (months to first difference less than e)	Consec_T_from_First	Max_consecutive_T	months_to_Max_consec
Caerphilly	Terraced	2018-01-31	F(4)T(14)	14/18	4	14	14	4
East Riding of Yorkshire	Flat	2018-08-31	F(6)T(1)F(1)T(3)F(1)T(5)F(1)	9/18	6	1	5	12
Aberdeenshire	Terraced	2019-01-31	F(2)T(9)F(1)T(1)F(2)	10/15	2	9	9	2
Hillingdon	Semi	2018-05-31	F(3)T(9)F(1)T(4)F(1)	13/18	3	9	9	3
Barnet	Semi	2018-10-31	F(5)T(2)F(1)T(2)F(1)T(7)	11/18	5	2	7	11
Southampton	All	2018-10-31	F(3)T(1)F(2)T(12)	13/18	3	1	12	6
West Dunbartonshire	Terraced	2018-11-30	F(2)T(11)F(2)T(2)	13/17	2	11	11	2
Torfaen	Semi	2019-01-31	F(2)T(1)F(7)T(4)F(1)	5/15	2	1	4	10
Cambridgeshire	Detached	2018-04-30	F(3)T(15)	15/18	3	15	15	3
East Riding of Yorkshire	Terraced	2019-02-28	F(4)T(3)F(3)T(4)	7/14	4	3	4	10

Figure 4. 14: Sample of dataset using criterion 1

Criterion 2

The second criterion tested is,

$$|Restatement_m| < e,$$

$$\text{for } e = 75th\ percentile\ of\ restatements \text{ for a specific combination.}$$

A python function is used to calculate the 75th percentile for a specific time series, this is then used as the threshold. It is taken into consideration that calculating restatements results in 1 null value due to the calculation of change. The first month is not included and the calculations are based on months with a restatement. Figure 4.17 displays the histogram of results using these criteria, it was found the average time taken is around 5.1 months and the median is 5.

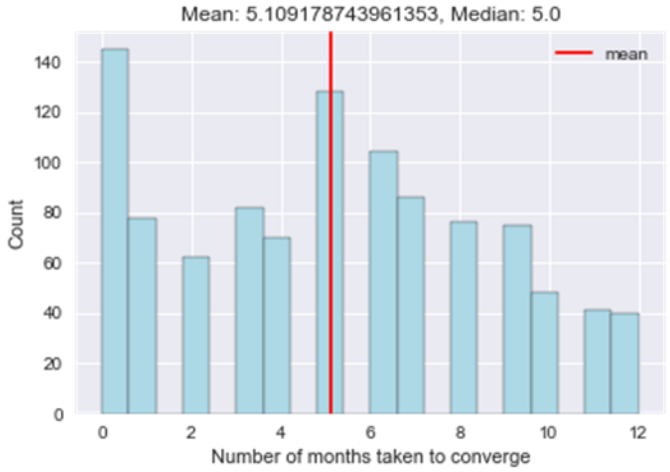
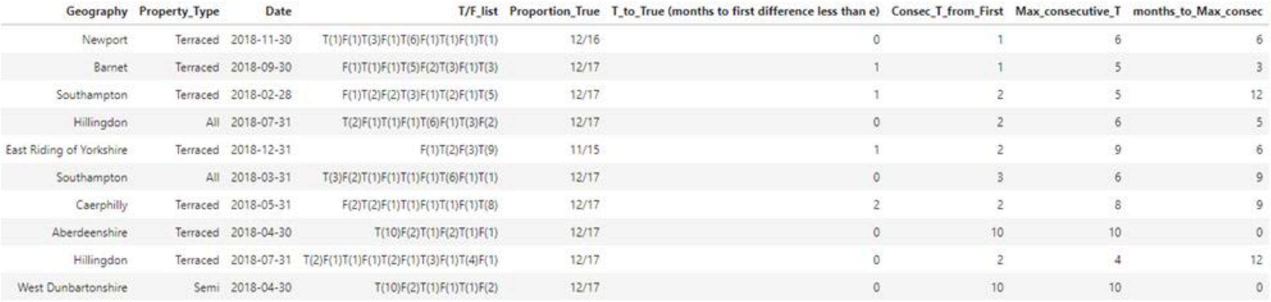


Figure 4. 15: Histogram of time taken for months to converge using criterion 2

A screenshot of the dataset using these criteria is shown in figure 4.18 to validate the results.



Geography	Property_Type	Date	T/F_list	Proportion_True	T_to_True (months to first difference less than e)	Consec_T_from_First	Max_consecutive_T	months_to_Max_consec
Newport	Terraced	2018-11-30	T(1)F(1)T(3)F(1)T(6)F(1)T(1)F(1)T(1)	12/16	0	1	6	6
Barnet	Terraced	2018-09-30	F(1)T(1)F(1)T(5)F(2)T(3)F(1)T(3)	12/17	1	1	5	3
Southampton	Terraced	2018-02-28	F(1)T(2)F(2)T(3)F(1)T(2)F(1)T(5)	12/17	1	2	5	12
Hillingdon	All	2018-07-31	T(2)F(1)T(1)F(1)T(6)F(1)T(3)F(2)	12/17	0	2	6	5
East Riding of Yorkshire	Terraced	2018-12-31	F(1)T(2)F(3)T(9)	11/15	1	2	9	6
Southampton	All	2018-03-31	T(3)F(2)T(1)F(1)T(1)F(1)T(6)F(1)T(1)	12/17	0	3	6	9
Caerphilly	Terraced	2018-05-31	F(2)T(2)F(1)T(1)F(1)T(1)F(1)T(8)	12/17	2	2	8	9
Aberdeenshire	Terraced	2018-04-30	T(10)F(2)T(1)F(2)T(1)F(1)	12/17	0	10	10	0
Hillingdon	Terraced	2018-07-31	T(2)F(1)T(1)F(1)T(2)F(1)T(3)F(1)T(4)F(1)	12/17	0	2	4	12
West Dunbartonshire	Semi	2018-04-30	T(10)F(2)T(1)F(1)T(1)F(2)	12/17	0	10	10	0

From these results, it was decided that the average time taken for restatements to converge is 6 months as the average value is rounded up since the time taken must be an integer.

4.3. Modelling

Tests carried out in the modelling section are based on data structure as well as machine learning algorithm selected. To save time, data structures: data split 1, no historic data, EEMD smoothing, and data split 2 are tested and evaluated using a support vector regressor. The data split that performs best is used to train the neural network models. In addition to this, a SVR is trained with and without the use of the February 2020 index data to see if including restatements affected by COVID19 hinders performance. First, the results from applying the methods to create the data are shown and visualized, then a description and results found in each experiment.

An overview of subsections is shown in the list below:

- *Data structures*
- *Experiment 1*
- *Experiment 2*
- *Experiment 3*
- *Experiment 4*

Data structures

Data split 1

Figure 4.1 displays the counts based on date and index month, we can visualize the train test split used in the initial test. The data is split to have the first 80% of each time series predict the final 20%. Figure 4.18 visualizes the split allowing us to identify which combination of date and index month belongs in the training and testing set.

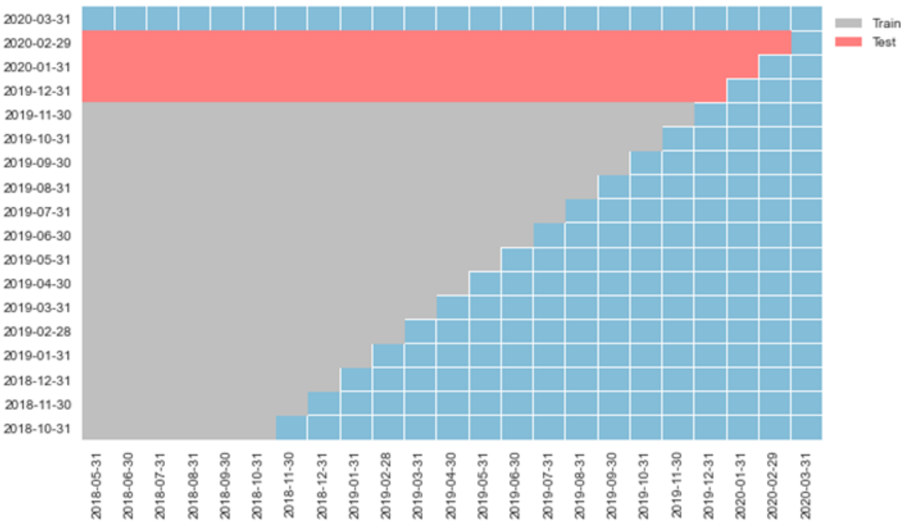


Figure 4. 16: Data split 1 based on index month and date

Data split 2

This data split adopts a different strategy where instead of aiming to have previous months predict the next few, we want to have all index months available in both the training set but not necessarily all combinations of date and index month. Table 3.6 in section 3.4 shows the criteria to determine if a data point belongs in the training or testing set, where m and d are chosen to allow for approximately the same proportion of data split as in data split 1. Figure 4.19 visualizes the approximate distribution of data split for training and testing. Different values for m and d are tested and it was found that $m =' 2019 - 12 - 31'$ and $d =' 2019 - 04 - 30'$ provided a satisfactory data split.

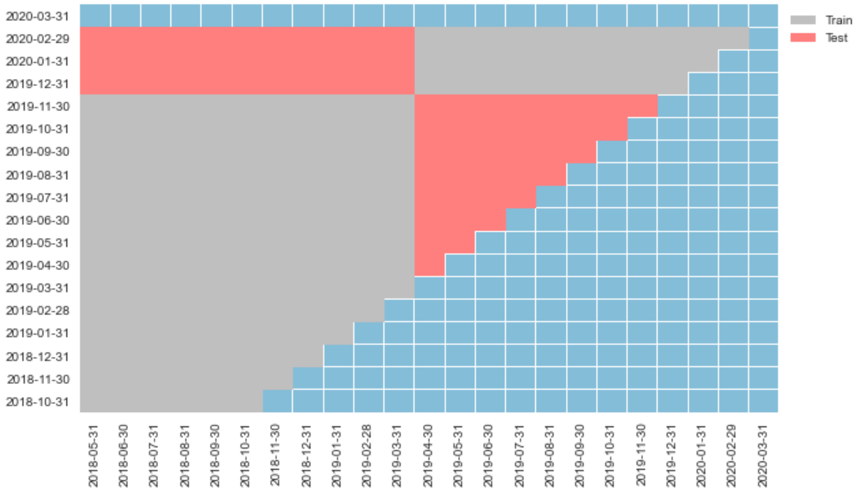


Figure 4. 17: Data split 2 based on date and index month

EEMD smoothing

Different values are tested for q, w, v, x and y for equation 5 using a sample of the time series.

$$Smoothed\ Restatement_m = \begin{cases} q \times eIMF_{m,1}, & if\ eIMF_{m,2} = NULL \\ w \times eIMF_{m,1} + v \times eIMF_{m,2}, & if\ eIMF_{m,3} = NULL \\ x \times eIMF_{m,1} + y \times eIMF_{m,2}, & else \end{cases} \tag{5}$$

Through trial and error, it was decided that the following values are used to smooth the time series,

$$q = 0.3, w = \ 1, v = 0.3, x = 1\ , y = 0.7.$$

Applying this method to the dataset results in the smoothed version of restatements shown in figure 4.20, as well as the true restatements to use as a comparison. It took approximately 0.55 hours to smooth all the time series and it is seen from the smoothed restatement plot that the maximum range of restatements is smaller and appear to be less volatile and correlated.

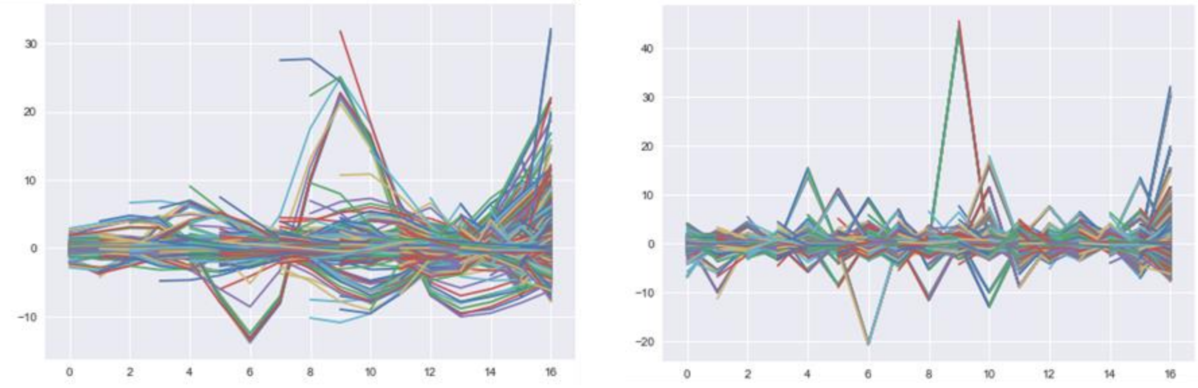


Figure 4. 18: Comparison of restatements after and before EEMD smoothing

This dataset is then saved and used to train a SVR model that is compared with an SVR model using data structures.

Experiment 1

In this experiment, a SVR is optimized using data split 1 and a model is trained using the following data structures:

- Data split 1
- Data split 1 excluding February
- Data split 2
- EEMD smoothing with data split 1

Hyperparameter optimization

Optimization was carried out using halving grid search with hyperparameter options shown in table 4.2.

Table 4. 2: SVR hyperparameter search space

Hyperparameter	**Search space**
C	[1, 100, 500, 700, 1000]
Gamma	[0.1, 0.001, 0.00001, 0.0000001]
Kernel	[Linear, RBF]
Epsilon	[0.1, 0.5, 0.01]

The optimization took approximately 9 hours, and it was found that the hyperparameter combination with the best results was,

$$C = 100, Gamma = 0.001, Kernel = RBF, Epsilon = 0.1.$$

It is important to keep in mind that halving grid search may not result in the same optimal hyperparameters as a regular grid search, however since the time taken for SVR to train is high, it is necessary to use a modified version of grid search that reduces overall time.

Model Training

A model was trained using each data structure, the time taken for the model to train on the respective data is shown in table 4.3.

Table 4. 3: Experiment 1 model training times

Data structure	Model training time taken/ hours
Data structure 1	1.4533
Data structure 1 excluding February	1.2153
Data structure 2	1.114
EEMD smoothing	0.3123

All structures took around the same amount of time while EEMD smoothing approximately a quarter of the time taken to train compared to other structures, this may be because of the decrease in the overall range of restatements decreasing the strictness of the regularization hyperparameter C.

Results

The models were tested using the same test set with unseen data for all models. Table 4.4 displays the results of experiment 1.

Table 4. 4: Results from experiment 1

Models	Evaluation metrics	
	RMSE	
	Train	Test
	Data split 1	
SVR	3.2677	3.1509
	Data split 2	
SVR	3.1872	2.9691
	Data split 2 excluding February in training	
SVR	2.7844	2.600
	EEMD smoothing with data split 1	
SVR	2.9633	2.2947

EEMD smoothing appeared to be the optimal data structure as it has the lowest RMSE on the test set. Data split 2 appears to be slightly better than data split 1 as expected since the model is trained with some data from all index months. Therefore, from the results it was determined that the data structure to use in the following experiments for SVR was a combination of EEMD smoothing using data split 2 and excluding the February index in training. Data split 2 excluding February is adopted for neural network structures.

EEMD is not used for neural networks as a similar approach to (Plakandaras, 2014) is adopted where it is only applied to SVR and given time constraints neural networks were not able to be tested with EEMD smoothing.

Experiment 2

This experiment compares SVR, MLP, RNN and LSTM using the combined data structure for SVR and data split 2 excluding February for neural networks. The neural networks are optimized in this experiment while the same hyperparameters from experiment 1 are used for the SVR.

Hyperparameter optimization

As the number of hyperparameters selected for the neural networks is significantly higher compared to SVR, 3 stages of hyperparameter optimization are carried out where the search space is narrowed down significantly each time; in addition to this, optimal parameters are found using the best data split. All networks are optimized using the grid search provided by the library Talos.

Multilayer perceptron

Table 4.5 shows the hyperparameter space used overall experiments to train the MLP, a fraction limit is set to reduce the total number of combinations to test.

Table 4. 5: MLP hyperparameter search space

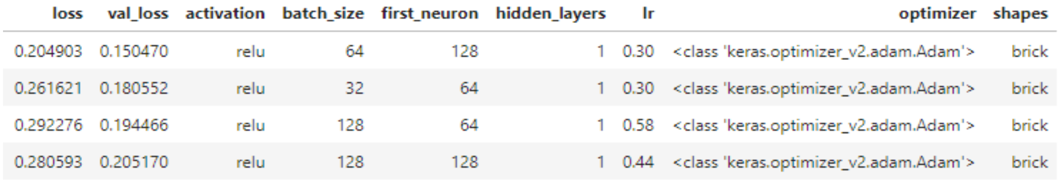
Hyperparameter	**Search space**
Learning rate	Range(0.1,5,15)
First neuron	1, 4, 8, 16, 32, 64, 128
Optimizer	[Adam, Stochastic Gradient descent]
Activation	[‘linear’, ‘relu’, ‘elu’]
Hidden layers	[0, 1, 2, 3]
Shape	['brick', 'funnel', 'triangle']
Batch size	[8, 32, 64, 128]

The top 4 results from the grid search for MLP are shown in a screenshot in figure 4.21. The results are ordered in ascending by loss and the optimal hyperparameters found are

$$activation = relu, batch\ size = 64, first\ neuron = 128, hidden\ layers = 1,$$

$$shape = brick, optimizer\ \ = adam, learning\ rate = 0.3.$$

The brick shape refers to the hidden layers having the same number of neurons as the input.



	loss	val_loss	activation	batch_size	first_neuron	hidden_layers	lr	optimizer	shapes
	0.204903	0.150470	relu	64	128	1	0.30	<class 'keras.optimizer_v2.adam.Adam'>	brick
	0.261621	0.180552	relu	32	64	1	0.30	<class 'keras.optimizer_v2.adam.Adam'>	brick
	0.292276	0.194466	relu	128	64	1	0.58	<class 'keras.optimizer_v2.adam.Adam'>	brick
	0.280593	0.205170	relu	128	128	1	0.44	<class 'keras.optimizer_v2.adam.Adam'>	brick

Figure 4. 19: Top 4 results of MLP hyperparameter optimization

The relationship between hyperparameter and loss is visualized by figure 4.22, where each hyperparameter is plotted against the loss and validation loss. Relationships between hyperparameter and loss can be seen, such as the drop in loss when the learning rate is between 0-1 and an increase in loss afterwards.

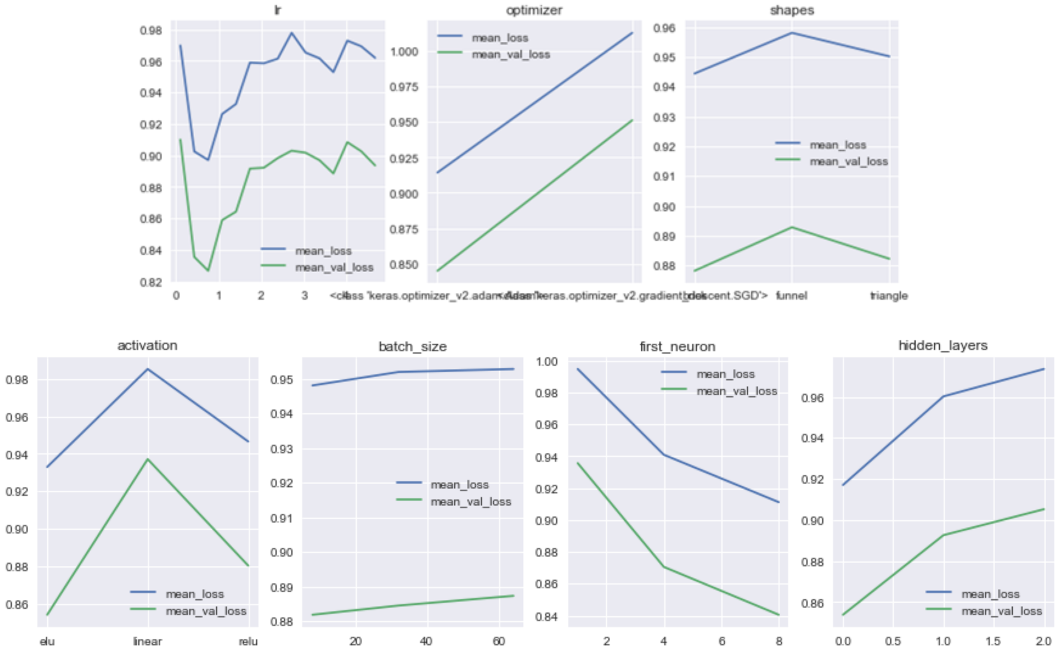


Figure 4. 20: Effect of hyperparameters on MLP

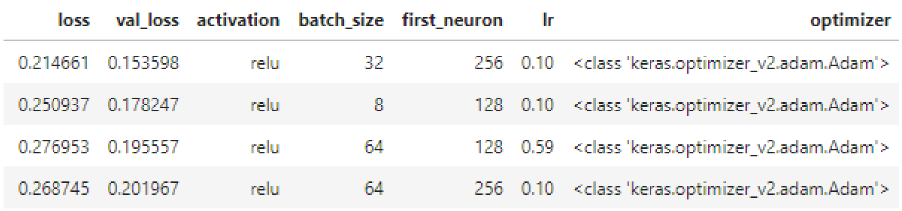
Recurrent neural network

The recurrent neural network was optimized with less hyperparameters tested, where the shape and number of hidden layers are excluded in the search. The search space is shown in table 4.6.

Table 4. 6: RNN hyperparameter space

Hyperparameter	Search space
Learning rate	Range(0.1,5,15)
First neuron	1, 4, 8, 16, 32, 64, 128, 256
Optimizer	[Adam, Stochastic Gradient descent]
Activation	[‘linear’, ‘relu’, ‘elu’]
Batch size	[4, 8, 32, 64, 128]

The hyperparameter combinations tested is less compared to MLP however it took approximately 10 hours since recurrent neural networks generally take longer to train since it is a more complex model.



loss	val_loss	activation	batch_size	first_neuron	lr	optimizer
0.214661	0.153598	relu	32	256	0.10	<class 'keras.optimizer_v2.adam.Adam'>
0.250937	0.178247	relu	8	128	0.10	<class 'keras.optimizer_v2.adam.Adam'>
0.276953	0.195557	relu	64	128	0.59	<class 'keras.optimizer_v2.adam.Adam'>
0.268745	0.201967	relu	64	256	0.10	<class 'keras.optimizer_v2.adam.Adam'>

Figure 4. 21: Top 4 results of RNN hyperparameter optimization

It was determined from figure 4.24 the hyperparameter values for RNN, these are

$$activation = relu, batch\ size = 32, first\ neuron = 256,$$

$$optimizer\ \ = adam, learning\ rate = 0.1.$$

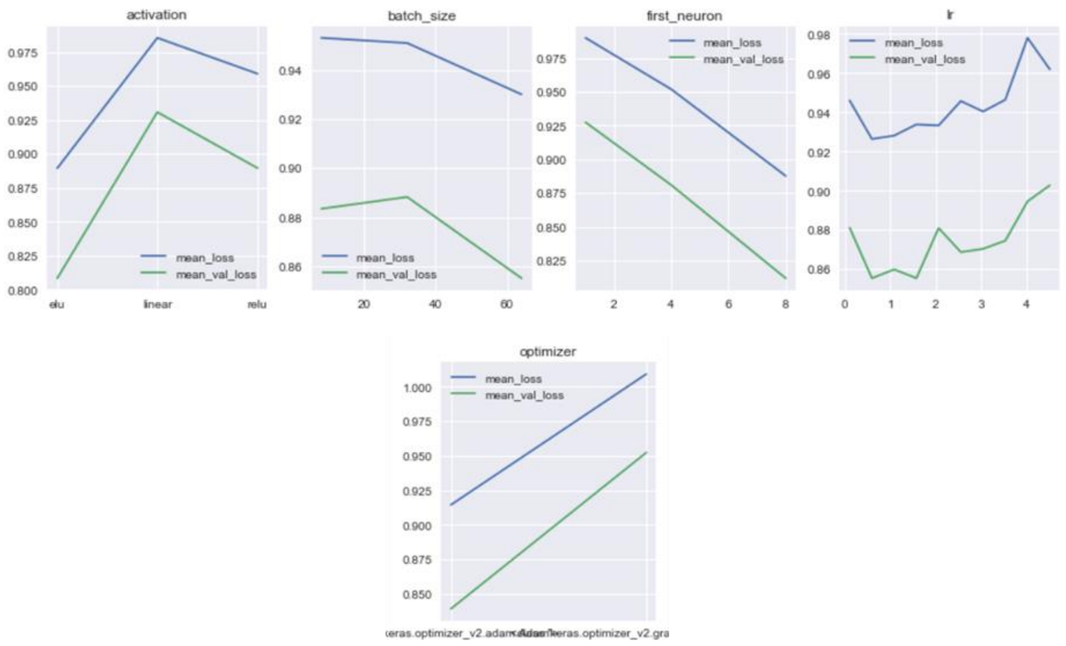


Figure 4. 22: Effects of hyperparameters on RNN

The effects of hyperparameters on the loss and validation loss for RNN can be seen in figure 4.23.

Long-short term memory network

The hyperparameter space used for LSTM is the same as RNN, shown in table 4.7. This optimization took approximately 15 hours which is significantly longer compared to MLP and RNN; this is expected since LSTM is a significantly more complex model compared to MLP and RNN.

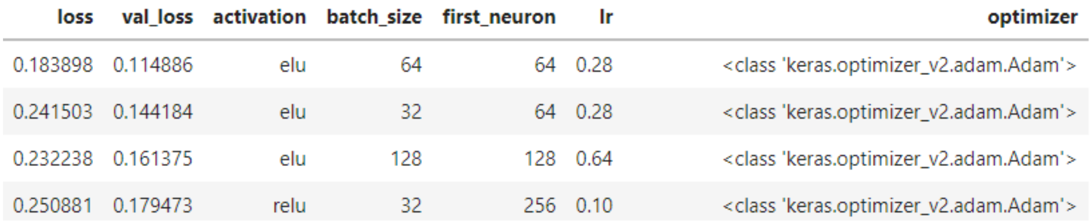
Table 4. 7: LSTM hyperparameter space

Hyperparameter	**Search space**
Learning rate	Range(0.1,5,15)
First neuron	1, 4, 8, 16, 32, 64, 128, 256
Optimizer	[Adam, Stochastic Gradient descent]
Activation	[‘linear’, ‘relu’, ‘elu’]
Batch size	[4, 8, 32, 64, 128]

A screenshot of the top 4 results found from the search is shown in figure 4.25, from this the hyperparameters selected are,

$$activation = elu, batch\ size = 64, first\ neuron = 64,$$

$$optimizer\ \ = adam, learning\ rate = 0.28.$$



loss	val_loss	activation	batch_size	first_neuron	lr	optimizer
0.183898	0.114886	elu	64	64	0.28	<class 'keras.optimizer_v2.adam.Adam'>
0.241503	0.144184	elu	32	64	0.28	<class 'keras.optimizer_v2.adam.Adam'>
0.232238	0.161375	elu	128	128	0.64	<class 'keras.optimizer_v2.adam.Adam'>
0.250881	0.179473	relu	32	256	0.10	<class 'keras.optimizer_v2.adam.Adam'>

Figure 4. 23: Top 4 results of LSTM hyperparameter optimization

Figure 4.26 visualizes the relationship of hyperparameters with mean loss and validation loss.

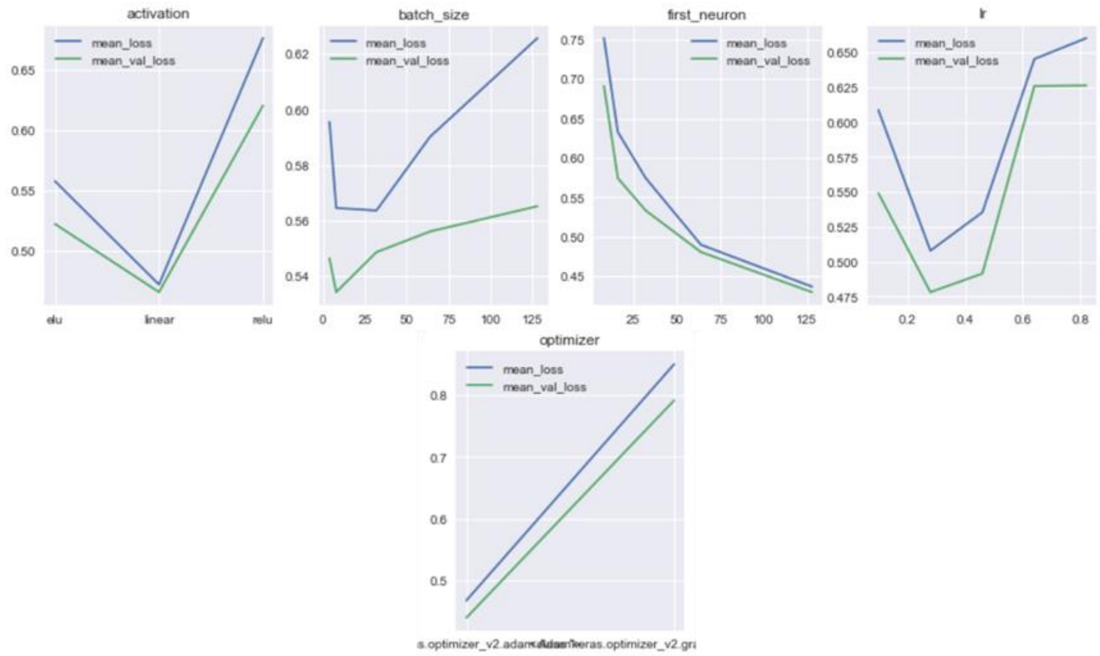


Figure 4. 24: Effects of hyperparameters on LSTM

Model Training

The SVR is trained on the combined data structure and the neural networks are trained on data split 2. Table 4.8 shows the training times for each model and figure 4.27 visualizes the number of epochs taken for the loss of the neural networks to converge.

Table 4. 8: Experiment 2 model training times

Model	Training time/ hours
SVR	0.1494
MLP	0.0775
RNN	0.1750
LSTM	0.23

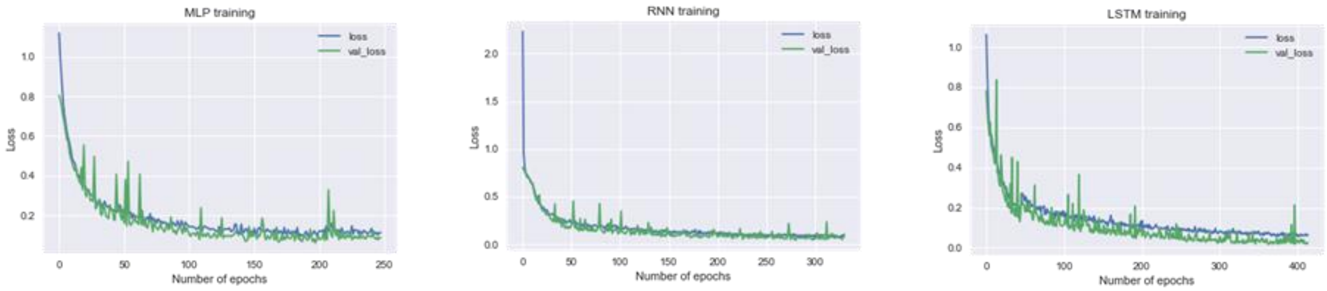


Figure 4. 25: NN loss at each epoch

RNN appears to have the fastest rate of convergence to minimum loss and validation loss as well as the lowest variance in validation loss. While MLP and LSTM have a similar rate of convergence, LSTM has a significantly higher number of spikes in validation loss.

Results

The results of experiment 2 are shown in table 4.10.

Table 4. 9: Results of experiment 2

Models	**Evaluation metrics**	
	RMSE	
	Train	Test
	EEMD smoothing with data split 2 and Feb exclusion	
SVR	0.11292	1.1928
	Data split 2 excluding February	
MLP	0.28030	0.55666
RNN	0.26489	0.44313
LSTM	0.14240	0.30563

There appears to be a relationship with the RMSE on the test set and the model complexity, the RMSE is highest for SVR and lowest for LSTM suggesting LSTM is a better model when using historic data. As expected, the RMSE on the train set is lower compared to the test set for all models, however for SVR the difference in RMSE for the train and test set is very significant, this might suggest that the models' generalizability is weak.

Experiment 3

This experiment trains models without historic data to investigate the ability to predict months with no previous restatements. The 4 models from experiment 2 are used along with their hyperparameters, however they are trained on a new dataset where SVR is trained using no historic data, data split 2 and EEMD smoothing. While the neural networks are trained with no historic data and data split 2.

Model Training

Table 4.11 shows the times taken for each model to train using no historic dataset.

Table 4. 10: Experiment 3 model training times

Model	**Time taken/ hours**
SVR	2.360
MLP	0.153

RNN	0.219
LSTM	0.396

The data with no previous restatements is larger than when historic data is used, hence the training times are higher compared to previous experiments. The time taken scales with model complexity except for SVR which took almost 6 times longer to train.

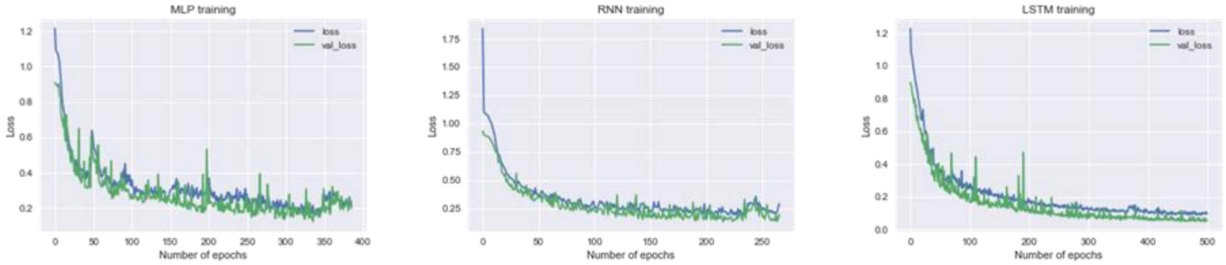


Figure 4. 26: Experiment 3 neural network loss each epoch

The loss and validation loss at each epoch appear to be more volatile compared to training with historic data and take longer to converge as seen by figure 4.28, this may be because there are more dates available in no historic data and therefore more labels have effects on the prediction.

Results

The results of the experiment are shown in table 4.12.

Table 4. 11: Results of experiment 3

Models	**Evaluation metrics**	
	RMSE	
	Train	Test
	No historic data + data split 2 + Feb exclusion +EEMD	
SVR	0.1129	1.1928
	No historic data + data split 2 + Feb exclusion	
MLP	0.4385	0.5918
RNN	0.4164	0.6062
LSTM	0.2010	0.4734

A similar trend is seen as in experiment 2 where the RMSE decreases with model complexity except for RNN, where there is an increase in both train and test RMSE compared to MLP. The same conclusion can be reached with SVR where its generalization capability is lacking compared to other models as seen by the difference in RMSE from the train and test set.

Experiment 4

This experiment implements transfer learning using the best model found that uses historic data and compares it with the best model found that uses no historic data.

Hyperparameter optimization

The hyperparameter space is shown in table 4.12.

Table 4. 12: TL search space

Hyperparameter	Description
Activation function	[‘linear’, ‘relu’, ‘elu’]
Optimizer	[Adam, Stochastic Gradient descent]
Learning rate	Range(0.0001,1,5)
Trainable	[True, False]
Batch size	[64, 128, 256]

The results of the hyperparameter optimization are

$$activation = relu, batch\ size = 256, Trainable = True$$

$$optimizer\ = adam, learning\ rate = 0.28.$$

Transfer learning training

The model took approximately 0.55 hours to train, the loss and validation loss each epoch can be seen in figure 4.29. This model had the overall slowest and most unsteady rate of convergence suggesting the model may not have learned properly.

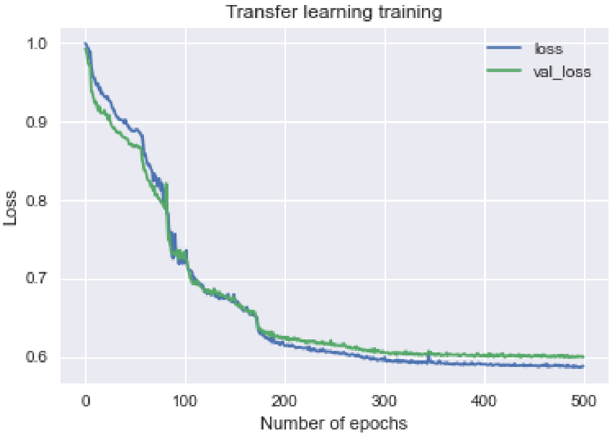


Figure 4. 27: Experiment 4 model training loss each epoch

Results

This transfer learning model is compared with the best model found to predict restatements with no historic data. It is found that LSTM still outperformed the TL model used even though it was trained using that LSTM as a base model. This somewhat implies the application of transfer learning for this case may not be necessary or more testing is required with transfer learning to find a more optimal solution.

Table 4. 13: Results of experiment 4

Models	**Evaluation metrics**	
	RMSE	
	Train	Test
	No historic data	
LSTM	0.2010	0.4734
TL	0.7600	0.9982

Conclusion

The experiments carried out aimed to find an optimal model that predicts restatements with historic data as well as without historic data. LSTM was found to be the most optimal model to use and outperformed other models tested in both cases. As expected, it was found that using historic data improves the performance compared to if historic data is not used.

Table 4. 14: Comparison of base metrics with the optimal model

Models	**Evaluation metrics**	
	RMSE	
	Train	Test
	With historic data	
LSTM	0.1424	0.3056
Country_level_mean	1.003	1.331
Regional_level_mean	1.004	1.330
Geography_level_mean	1.018	1.294
	Without historic data	
LSTM	0.2010	0.4734
Country_level_mean	1.0037	1.2342
Regional_level_mean	1.0039	1.2345
Geography_level_mean	1.0171	1.2318

LSTM is compared with base evaluation metrics discussed in section 3.5. Table 4.13 shows the results of the LSTM models in comparison with the base metrics calculated as per the dataset of the model. LSTM appears to have a significantly lower RMSE compared to all base metrics for both cases of historic and no historic data used. This suggests that using this model is better than using obvious predictions one might make for the restatement of a specific geography.

5. Discussion

5.1. Evaluation of objectives

5.1.1. Data acquisition

Data acquisition is the first objective and arguably the most important. The necessary data required to carry out the project was obtained through different means, being online resources such as ONS as well as querying from the MIAC analytics database through a python loop to obtain the volume and index data each month. This objective was met.

5.1.2. Data exploration and pre-processing

This objective can be split into two subsections, data exploration and pre-processing, both of which are extremely important.

The exploration phase is carried out to a point where relationships and properties in the data are understood; in this case comprehending the relationship of *Sales_delta*, *Index_month* and *Date* with restatements. In addition to this, understanding how the interpolation methodology results in unexpected trends such as highly correlated restatements between the same combination of geography and property type for transactions that occur on different dates, as well as changes in restatement with no increase in volume. A downside to exploring data with hundreds of time series such as this case is that we cannot say anything is for certain for all the data, given that there are 872 time series it is unpractical to look at every single one to see if these trends are found, therefore these are based off random samples. This part of the objective was met very well as it provided very useful insight and resulted in the implementation of an updated methodology for the house price index that is discussed in 5.2.

A large number of outliers in each index seen in the boxplot in figure 4.8 may be a result of the mix of price discrimination and the implementation of matched sampling in the interpolation methodology. The correlation between restatements might imply that the matched sampling indirectly discriminate prices, resulting in a bigger proportion of the data being considered outliers.

The multicollinearity among macro-economic may be the reason the support vector regressor models performed worse than the neural networks as it is shown that SVM is impacted by multicollinearity (Raj, 2019) particularly with a RBF kernel, this may imply that SVR is also impacted as they follow a similar methodology. Neural networks however generally do not suffer from multicollinearity as they are usually overparameterized (Richard De Veaux, n.d.).

There is significant importance in the order of steps taken for pre-processing, particularly when it comes to scaling the data and ensuring no information on the test set is leaked onto the training set, another example is scaling the historic sliding window data using the same transformation as the restatement. Carrying out these steps the other way around can result in different values calculated for the same restatement, this is because the sliding window values may be in a different range of values compared to the restatement, in turn resulting in a different mean and variance when scaling.

The method of calculating restatements as well as the sliding window was optimized extremely well to significantly reduce the time taken, this method uses inbuilt python functions which have been optimized for speed and avoids looping over each time series which would take a long time. The preparation of different data structures was all met except for the 3D data structure, while transforming the data to 3D was successful, the 3rd dimension should account for historic data. Instead of adding the historic restatements to the third dimension, this dimension was set to 1 so the 2D dataset was identical but had a 3rd dimension consisting of 1 row, and the historic data remained in the columns of the dataset. It was intended to transform the dataset to incorporate historic data in the 3rd dimension however due to insufficient time it was unable to be created.

5.1.3. Variation of restatements

The objective of finding the average time taken to convergence was met, this objective not only provides insight on the rates of convergence but is also used practically as the number of historic restatements to use as predictors. More work could have been done to involve using other metrics besides average time taken, such as including the variance of the time taken to influence certain decisions; however, it is difficult to incorporate this into model training.

5.1.4. Modelling

The modelling phase aimed to test multiple data structures and models to find what performs optimally, due to time limitations, not all combinations of data structure and model could be tested such as EEMD smoothing for neural networks. The task of modelling is somewhat tedious and extremely time-consuming, particularly when it comes to hyperparameter optimization. The main models to be tested were SVR and LSTM, simpler neural networks were adopted being MLP and RNN to ensure there is a decrease in RMSE with model complexity instead of jumping straight to a complex model.

While this is a time series forecasting task, the application of general time series methods does not work very well because of the relationship of date and index month to restatement. The main example of this is the use of the sliding window and its limitation on the dates we can predict. Figure 5.1 visualizes the inversely proportional relationship between using a sliding window of size n and the number of months we can predict from the latest month. There appears to be a trade-off between the months we can predict and the amount of historic data we can use. By using the sliding window, we are hard coding the requirement of historic data when trying to predict a certain month, if it does not exist then it cannot be predicted using that model. This limitation makes it significantly different to any other time series task, without the use of historic data this may just be considered a regression task therefore opening much more possibilities in terms of models to use.

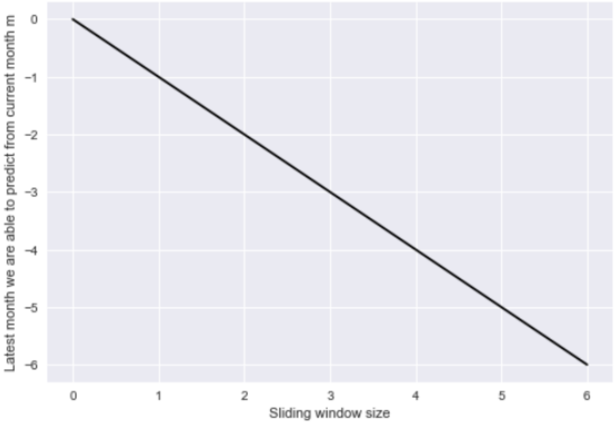


Figure 5. 1: Relationship between sliding window size and months we can predict

Using a sliding window of 6 months may be considered somewhat a contradiction. Since the average time taken for restatements to converge is 6 months, requiring 6 months of historic data to predict the next month may be trivial since the time series is considered to have converged by that time, this may not be completely inconsequential as it is only the average time from a sample of all the time series. With this method, a compromise must be made to use historic data.

5.1.5. Evaluation

The evaluation was carried out to compare models with each other to find the best one, and then evaluate if using the best found model is worth it by making comparisons to base predictions that would be made for the restatement of a specific geography and property type, the base used is average of previous restatements of different levels being country level, regional level and geography level, these are obvious choices to use as a prediction of restatements and so if the model can predict better, then it is worth using. Given more time, further complex base predictions could have been tested with better results to allow for a better evaluation of the models.

It was made sure to keep model comparisons as fair as possible, using the same testing set to calculate the mean square error, root mean square error and mean absolute percentage error; because models with different data structures are compared, it is not possible to train them on the same data.

The objective of the evaluation is to assess all the selected models' performance and determine how well they compare to base predictions, and this objective was met.

5.2. Answering the research question

How well can we predict future restatements?

The main aim of this paper was to investigate how well future restatements could be predicted using machine learning techniques. Given that MIAC analytics have no predictive modelling for restatements, there is nothing to compare the performance of the models to. Therefore, base predictions are selected such as restatement on a country level to evaluate if it is worth using predictive modelling to predict restatements. It is clear the models used to predict restatements are significantly more accurate than using base predictions, with the optimal model found providing over 50% decrease in root mean square error compared to the country, regional and geography level predictions of restatements.

5.3. Interpolation methodology

The discovery of the effects of MIACs interpolation methodology on the data through exploration resulted in the testing and implementation of an updated methodology that also relies on volume data. The method, tests and results of this new methodology are discussed in this subsection.

The main idea of the new methodology incorporates the replacement of prices based on a criterion, where $Price_m$ is replaced with $Price_{m-1}$ if the criteria are not met. The criterion selected is described below.

Criterion 1

The first criterion is described by equation X,

$$Replaced\ Price_m = \begin{cases} Price_m, & if\ Sales\ delta_m > 0 \\ Price_{m-1}, & otherwise \end{cases}.$$

This equation describes the process of the creation of the dataset following the first criterion, where the $Replaced\ Price_m = Price_m$ only if there is a change in volume (also known as sales delta) at month m, otherwise $Replaced\ Price_m = Price_{m-1}$.

Criterion 2

The second criterion is described by equation X,

$$Replaced\ Price_m = \begin{cases} Price_m, & if\ Sales\ delta_m > 0\ and\ |Restatement_m| < e \\ Price_{m-1}, & otherwise, \end{cases}$$

$$\text{For } e = 75th\ percentile\ of\ restatements\ for\ that\ combination.$$

The second criterion uses an idea like the calculation made to find the average time taken for convergence, where $Replaced\ Price_m = Price_m$ if there is a change in volume, and the change is above a certain threshold defined by e.

To demonstrate these in relation to the true prices and change in volume, combinations are randomly sampled, and graphs are made to visualize the differences in criterion. Figure X visualizes how these new methods compare with the current prices in the index as well as the change in volume. Indicators are marked on each graph; a grey dashed line indicates where the criteria is met whereas the black dots indicate where the volume is 0.

From this sample, we see that cases with large changes in price do not follow the criteria and are therefore not part of the replaced prices, an example being the large increase in price for ‘Bracknell Forest’, ‘Flat’, ‘2019-07-31’. The main difference between criteria 1 and 2 is that there is reduced volatility when using the second criteria as small jumps are not included even when there is a volume change.

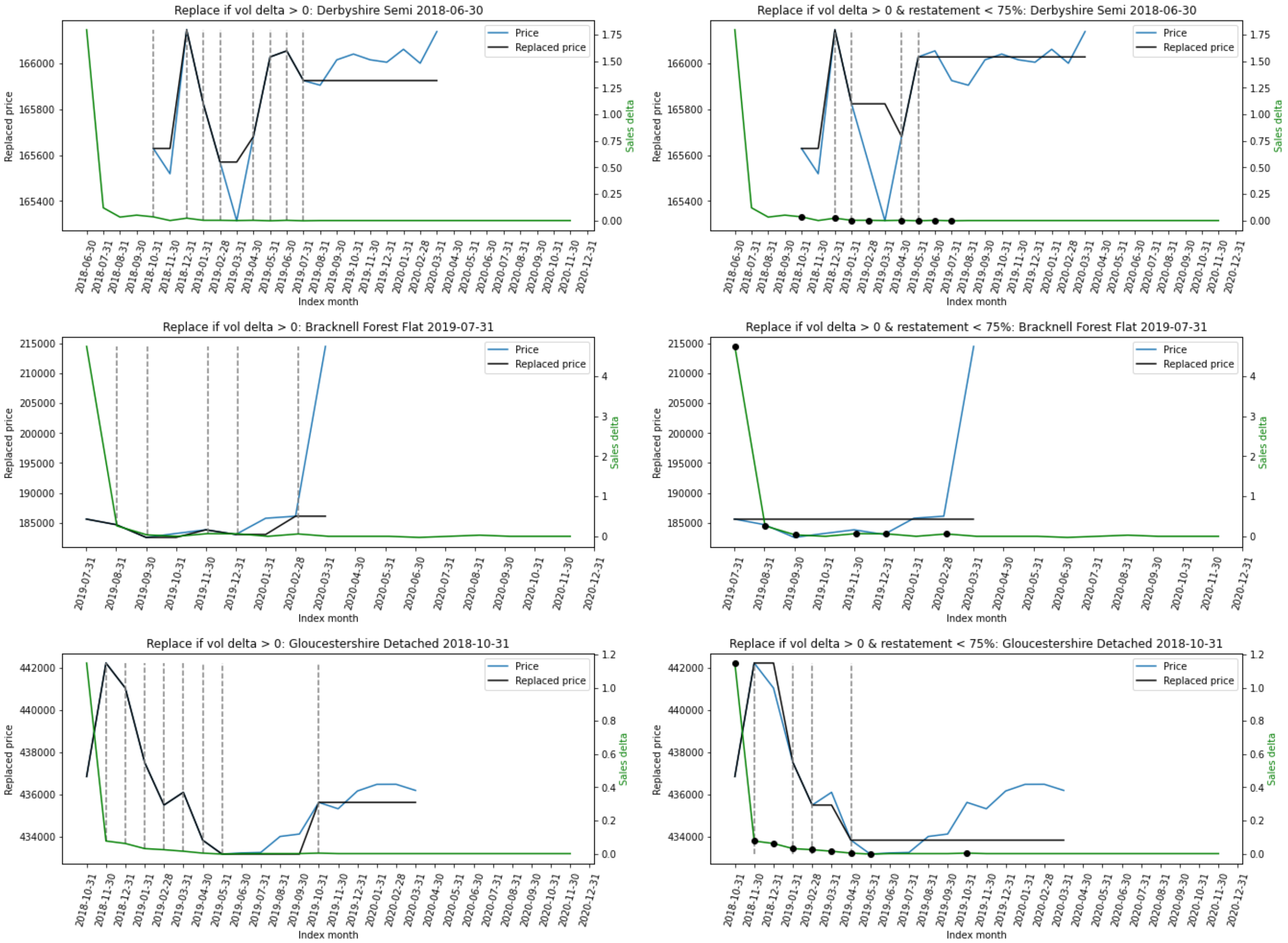


Figure 5. 2: Comparison of new methodologies

It was decided that a new dataset would be created in the MIAC database with the implementation of criterion 2 on the current house price index as it reduces volatility in the index.

6. Evaluation, Reflection and Conclusions

6.1. Choice of objectives

The choice of objectives lined very well with the research question; all parts of each objective contributed to answering the research question. The main objectives that would help answer the research question are modelling and evaluation, which were successful.

The literature sourced for this study guided decisions such as which models to select in the design of experiments, however, research papers on time series forecasting of data that relies on multiple date variables for house price prediction could not be found, the unique property of the relationship between the amount of historic data used with the restatement dates we can predict. Due to this, papers sourced are relevant to certain parts of the project instead of the whole project itself.

The modelling phase was not as smooth as it could have been, a lot of time was wasted with hyperparameter optimization and while the literature search was useful in determining which models to choose, they lacked information on the data structure. The transfer learning model was tested as a concept to incorporate trends found in historic data to predict restatements with no available historic data, this was expected to outperform models trained only with no historic data but turned out to be a bit disappointing.

Overall, the project provided significant insight into MIACs house price index through data exploration, and it was shown that using machine learning, restatements can be predicted more accurately than when using obvious predictions such as the mean restatement.

6.2. Future work

Future work may include testing with the updated interpolation methodology discussed in section 5.3, given that this is a new implementation it was not able to be tested for this study, however it is expected that results may be significantly better since the new methodology appears to reduce volatility in the index, as well as ignore large jumps in the case where there is no change in volume.

Transfer learning was tested with the hope that it would improve the predictions when using no historic data by incorporating trends from previous time series restatements, further research could be done with regards to transfer learning to find a more optimal model structure besides the one used in this study.

Other work may also multi-step forecasting of restatements as well as discovering new base evaluation metrics to use to determine the performance of a model and testing how statistically significant the model's difference is. Comparisons of machine learning models to classical methods such as ARIMA may also be worth testing to see which are better for cases where restatements have historic data.

6.3. Reflection on the project

A significant amount of effort was put into the project, particularly in the data pre-processing and modelling phases. The importance of data exploration was understood as testing was carried out mainly on the February index, resulting in a high error as they contain a significant number of outliers; this could have been avoided if the initial exploration was more thorough. An understanding of time series forecasting for both classical methods as well as machine learning was obtained through literature search and testing.

The insight gained on the house price index as well as the tests carried out can be very useful to MIAC.

7. References

Anon., 2018. *(2018). [Machine Learning] Regression--Support Vector Regression (SVR) - Programmer Sought. ,* s.l.: s.n.

Ashkan Eliasy, J. P., 2020. *The comparison between Recurrent Nueral Network and Feed Forward Neural Network.* [Online]
Available at: https://www.researchgate.net/figure/The-comparison-between-Recurrent-Neural-Network-RNN-and-Feed-Forward-Neural-Network_fig1_338672883

Bottou, L. a. L. C.-J., n.d. *Support Vector Machine Solvers Support Vector Machine Solvers. ,* s.l.: s.n.

Brownlee, J., n.d. 4 Strategies for Multi-Step Time Series Forecasting.

Brownlee, J., n.d. Comparing Classical and Machine Learning Algorithms for Time Series Forecasting.

C. Yang, W. J. a. Z. G., 2019. Time Series Data Classification Based on Dual Path CNN-RNN Cascade Network.

Chen, X. W. L. a. X. J. (., n.d. *House Price Prediction Using LSTM..* [Online].

Debasish Basak, S. P. D. C. P., 2007. *Support Vector Regression,* s.l.: s.n.

Dupond, S., 2019. A thorough review on the current advance of neural network structures.

Gilde, K., n.d. *Faster Hyperparameter Tuning with Scikit-Learn's HalvingGridSearchCV.* [Online]
Available at: https://towardsdatascience.com/faster-hyperparameter-tuning-with-scikit-learn-71aa76d06f12

Jolliffe, 2002. In: *Principal Component Analysis.* s.l.:s.n.

Kecman, 2013. *Support Vector Machines: Theory and Applications,* s.l.: s.n.

Keim, R., 2019. *How to Train a Multilayer Perceptron Neural Network.* [Online]
Available at: https://www.allaboutcircuits.com/technical-articles/how-to-train-a-multilayer-perceptron-neural-network/

L.Karthikeyan, D. K., 2013. Predictability of nonstationary time series using wavelet and EMD based ARMA models.

Limsombunchai, V., 2004. *House Price Prediction: Hedonic Price Model vs. Artificial Neural Network.* New Zealand, s.n.

Milunovich, 2019. Forecasting Australian Real House Price Index: A comparison of Time series and Machine learning methods..

Olah, C., 2015. *Understanding LSTM Networks.* [Online]
Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pedro Henrique Borghia, O. Z. J. P. T., 2020. A COVID-19 time series forecasting model based on MLP ANN.

Plakandaras, V. G. R. G. P. a. P. T., 2014. Forecasting the U.S. Real House Price Index.

Raj, S., 2019. *Effects of Multi-collinearity in Logistic Regression, SVM, Random Forest(RF).* [Online]
Available at: https://medium.com/@raj5287/effects-of-multi-collinearity-in-logistic-regression-svm-rf-af6766d91f1b

Richard De Veaux, L. U., n.d. Multicollinearity: A tale of two nonparametric.

Yao Qin, D. S. C. C. J. W. C., 2017. A Dual-Stage Attention-Based Recurrent Neural Network.

Yun-long Kong, Y. M. W. L. A.-z. Y. Y. Y., 2015. *Satellite Image Time Series Decomposition Based on EEMD,* s.l.: s.n.

Appendix A – Project Proposal

Forecasting the restatement of house prices using machine learning

Yousef Gharib

1. Introduction

Mortgage Industry Advisory Corporation (MIAC) Analytics is an independent consultancy based in the United States and has been a leading provider of mortgage and financial services such as risk management and data auditing.

MIAC analytics have their own UK house price index based on actual property sold, as recorded by the Land Registry in the UK that is updated monthly; transactions are recorded very slowly by Land Registry meaning that it may take months to receive all the transactions that occurred in a single month. This issue has worsened since the start of COVID-19 resulting in the need of predicting the change in the aggregated house price (known as re-statements) as more volumes of transactions are recorded.

The formal explanation of the problem is, if $HPI(m, t_i)$ is the house price index value at time t_i calculated in month m, then generally for $n > 0$, $HPI(m, t_i) \neq HPI(m + n, t_i)$ due to the way data is obtained over time. If assumed that for $n = N$ large enough, we have $HPI(m + N, t_i) = N = HPI(m + n + k, t_i)$ for all $k > 0$ meaning we have all the transaction for month t_i in month $m + N$, and therefore know the index value at that point and have been approximating $HPI(m + N, t_i)$ using a subset of the full data.

Therefore, the research question this project aims to answer is:

How well can we predict future restatements given previous restatements and other macroeconomic data?

Through this we may be able to predict the restated values for lagged months where there is not enough data to calculate the aggregated price for that index.

Research objectives:

- *Data acquisition:* The data required consists of a mix of the index data which will be obtained from the MIAC database and macro-economic data which can be obtained from other sources such as Office for national statistics (ONS). The aim of this objective is to retrieve appropriate data for the next objectives.
- *Variation of restatements:* The time taken for restatements to converge can be calculated given the processed data; this may vary greatly based on factors such as location and property type. By finding the average time it takes for restatements to converge, we can use this information to decide how long of a window to consider when predicting a certain month in the time series.
- *Data pre-processing:* An exploratory analysis will be conducted to visualize properties of the data such as distribution and outliers. The data will also be processed through merging datasets and calculations of new features. This step is crucial, and the resulting dataset will be used for model training and evaluation.
- *Modelling:* Specific machine learning models are selected based on relevant literature and will be trained on the processed dataset. These models will be compared and used to predict the restatement for lagged months. This is the focus of the project and

investigating how good of a prediction we can get using a machine learning model is the objective.

- *Evaluation:* The models will be assessed during the evaluation period using general machine learning metrics. This objective allows us to identify each model's performance and evaluate which are the best for this task.

This project is expected to have the following beneficiaries:

- The results from this project will directly benefit MIAC analytics as it may attract new customers if advertised as a product.
- Consumers of MIAC products may also benefit from this project as they may have exclusive access to predicted results and can make decisions based on the results.

2. Critical context

2.1. Time series forecasting

Time series forecasting is a popular economics problem revolving around predicting future observations by fitting a model to historical data. Classical models such as Auto regressive integrated moving average (ARIMA) and Vector autoregression (VAR) have generally been used for time series forecasting as they can be adapted to any time series problem easily and the only data required is the time series itself. The classical techniques tend to outperform machine learning techniques for time series; however, the multivariate predictability of machine learning techniques leaves a huge potential for machine learning models in predicting time series.

A plethora of machine learning models have been tested for time series forecasting and, in most cases, the optimal model depends on how many months are being forecasted; generally, models with a simpler architecture and input structure perform better for short-term forecasting. Support vector regressors tend to outperform other models for house price prediction [1,2], however LSTMs have shown to work well when forecasting for multiple locations [4].

2.2. Support Vector Regression (SVR)

Support vector machine is a popular machine learning algorithm generally used for classification tasks; this algorithm can also be adapted for regression tasks using similar principles and maintains its robustness as a machine learning algorithm. For classification tasks, a SVM constructs a hyper-

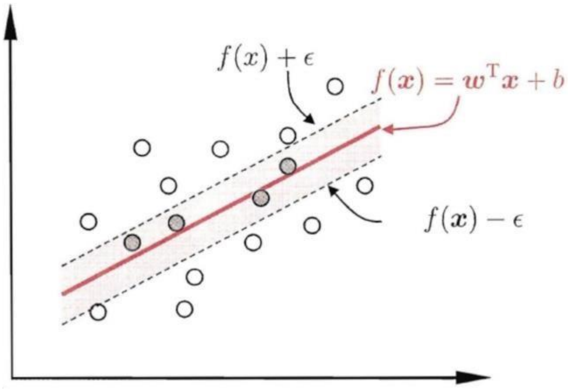

$f(x) + \epsilon$
$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$
$f(\boldsymbol{x}) - \epsilon$

Figure 1: SVR [7]

plane to separate the classes in the dataset, the hyper-plane is constructed with the condition of maximizing the distance of the nearest data points for each class [6]; whereas for regression tasks, the aim is to find a function such that the difference between the actual data points and the function is within a threshold accuracy.

SVR has shown to perform excellently for time series forecasting [8] and has shown to outperform other popular machine learning models, specifically for short term prediction [1,2]. In addition to this, SVR can be highly effective with high dimensionality data through the application of kernel functions to transform the data, in turn allowing for easier optimization of loss functions. However, SVR works by solving constrained quadratic equations to minimize the loss function in which time can increase quadratically with the training set [9]. Combining hyper-parameter optimization with long SVR training times can end up taking a significant amount of time which may not be worth it.

2.3. Long short-term memory (LSTM)

Long short-term memory neural network is an artificial recurrent neural network that is mainly used in learning patterns of sequential data such as music composition, speech recognition and time series prediction. LSTMs work similarly to other recurrent neural networks in that the network loops over itself to allow information to be passed on at each step; the advantage of LSTMs over regular recurrent neural networks is that they are capable of learning long term dependencies through the cell state in the network structure, as shown by the top horizontal line in figure 2. The cell state stores additional information and is updated at each step.

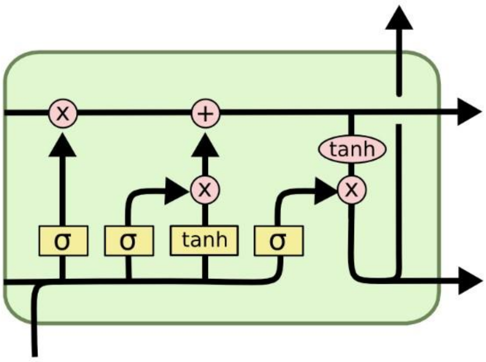


Figure 2: LSTM structure [13]

The ability to learn long-term dependencies is what makes LSTMs excellent at time series forecasting, since they can identify long term trends in the data. LSTM has become a popular model to test when using machine learning for house price prediction and has shown its ability to produce significant results [2,4]. Another advantage of LSTM is its adaptability through variants such as bidirectional and stacked LSTMs, as well as the ease of adjusting the model structure to allow for multiple outputs that may be based on geographic location as done by [4].

3. Approaches

3.1. Literature search and survey

The literature was sourced from Google scholar and other academic sources such as arXiv to validate the approaches chosen as well as give ideas on how the approach can be improved. An ongoing search for relevant literature will be carried out as seen necessary while working on objectives and writing the report.

3.2. Software and machine

The main software used for this project is Python as it provides all the necessary data preprocessing (Pandas), machine learning (Sklearn and Keras) and visualization tools (Seaborn and Matplotlib) to carry out the tasks set. Sequel query language (SQL) will be used for the data acquisition objective to obtain the necessary data.

The objectives will be carried out on a local computer. The use of cloud services will be decided on prior to model training and distributed learning will also be considered if model training times are longer than expected.

3.3. Data acquisition and preprocessing

The index data comprises of an aggregated price based for multiple locations and property types within the United Kingdom as well as dates ranging from January 1995 up to the specified index month; this data will be obtained from MIAC analytics database using SQL. Index data is available for each month from October 2018 up to March 2020 excluding lagged months and dates from when COVID-19 affects the index significantly. Data for each index month will be queried and appended on top of each other resulting in a large dataset consisting of all the necessary index data.

In addition to this, monthly macro-economic data will be obtained from multiple sources to help with the prediction, it is important to investigate if the macroeconomic data is lagged to ensure the dates for all the data match, this avoids issues where we are unable to predict due to missing data points. The initial macro-economic variables chosen are Gross domestic product (GDP), Unemployment rate and Interest rate; these were chosen as they were some of the most common macro-economic variables used to predict house prices, as seen by [1,2]. Extra macro-economic data such as inflation rate may be considered after testing to try and improve the models' performance. Annual macro-economic data may also be considered in addition to monthly data to reduce the variance of macro-economic data and incorporate overall trends instead of monthly change.

Once the data has been obtained and merged, a new column will be created to calculate the restatement as mathematically defined in equation 1.

$m = \text{Index month}$

$i = \text{Date in specific index month}$

$$Restatement_{m,i} = 100 \times \left(\frac{Price_{m+1,i}}{Price_{m,i}} - 1\right) \tag{1}$$

The final pre-processing step is to add a window of previous months of length *n*, where *n* is the average time taken for restatements to converge. This step is done by creating a new column in the dataset and shifting the restatements down by 1 for each column, as shown by table 1.

Table 1: Example of sliding window method for n = 4

Restatement	Restatement_-1	Restatement_-2	Restatement_-3	Restatement_-4
0.5	*NaN*	*NaN*	*NaN*	*NaN*
0.3	0.5	*NaN*	*NaN*	*NaN*
0.11	0.3	0.5	*NaN*	*NaN*
0.43	0.11	0.3	0.5	*NaN*
0.314159	0.43	0.11	0.3	0.5

This method incorporates structure into the model training by using previous *n* months restatements as predictor variables and has been used in many cases for time series forecasting using machine learning [3].

3.4. Variation of restatements

The final dataset of all the appended index months will be used to estimate the average time taken for restatements to converge, this gives us an idea of the number of months it takes for prices to stabilize in the index and informs us of the expected time taken for the total volume of data obtained each month to stabilize since we only expect the price to change if more transactions are obtained each month.

Based on the data, a new table with new variables will be created and defined to calculate the average time taken for restatements to converge along with other features. The time to convergence is found by investigating at what point the restated price (change in price) is not significant, defined as $Restatement_{m,i} < e$ where e is a threshold chosen and $e > 0$. Any number may be chosen as this threshold; restatements may vary greatly based on location and date therefore implementing a threshold that adapts to the location may result in a more reliable average time to convergence, an example of such implementation is using a threshold that is a combination of mean and standard deviation of the restatement values.

The average time taken will be used as the variable *n* to determine the window size to include in model training, doing this incorporates sequential data into the model as required for time series forecasting.

3.5. Modelling and testing

This project uses similar implementation to relevant literature on predicting house prices, however since we are predicting the restatements, the data results in being a time series within a time series; meaning the way the data is fed to the models is different compared to regular time series problems. Figure 3 displays how normal time series data is fed to a machine learning model. For a given time series, months used as training data to predict future months. Figure 4 shows how the time series data will be fed to the model for this project. The data is comprised of multiple datasets representing each index month, within each index month is the date and aggregated price of transactions that occurred at that point in time. Therefore, the time series is comprised of the restated price calculated between index months, for a specific date.

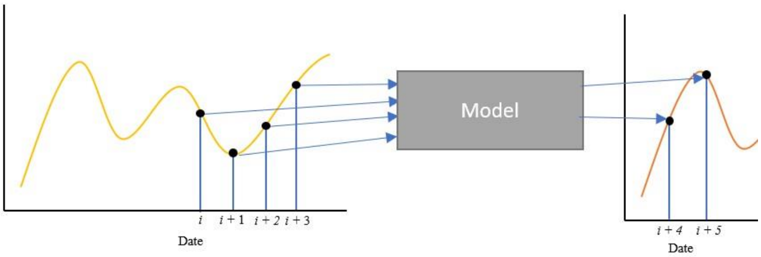


Figure 3: How normal time series data is fed to a model.

While the time series is slightly different, it is expected that all methods found in relevant literature will be applicable and provide similar results in this case.

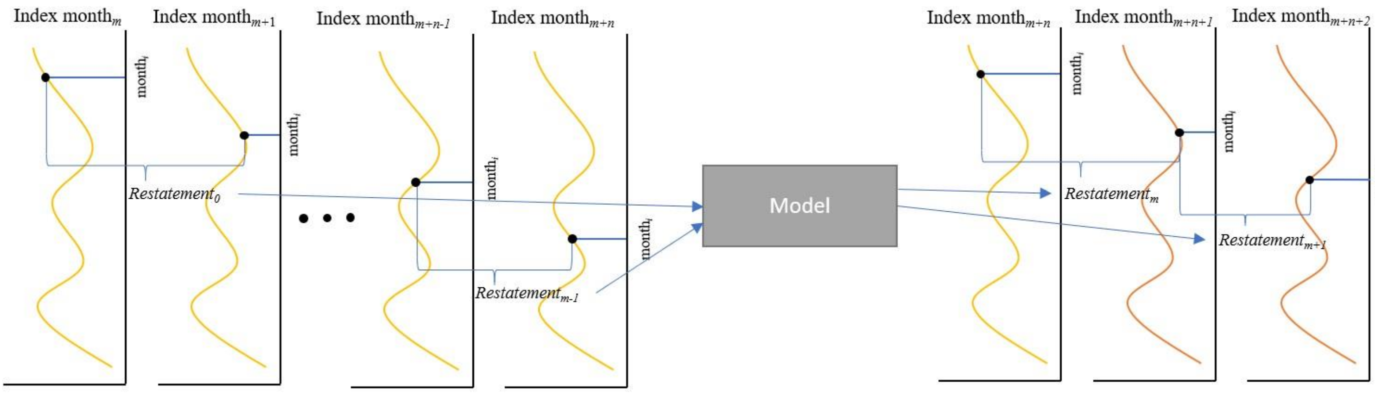


Figure 4: How time series data will be used

The initial machine learning models are chosen based on their performance in relevant literature. Since the focus is short term forecasting, the initial models chosen are support vector regression (SVR) and long short-term memory neural networks (LSTM) as they have shown to perform well for short term house price forecasting [1,2,4]. Other models may be considered such as Random forests as it is an ensemble method (and we are working with multiple locations and property types), in addition to multi-layer perceptron and LSTM variants such as Bidirectional LSTM as it has potential to outperform other mentioned models [5].

Model hyperparameters will be optimized using a grid-search, if it is found that model training times are too long due to this, other hyperparameter optimization methods such as Bayesian optimization may be implemented instead. For a fair comparison, all models will be optimized and tested using the same training and test set. Given the timeframe it is expected that sufficient time will be available to test models in addition to SVR and LSTM, but the initial objective is to test and evaluate those two models. Model implementation and hyperparameter optimization will be carried out using prebuilt python libraries, some models such as bidirectional LSTM may be tougher to implement as code for the model may need to be obtained from online resources, however this will be investigated if the decision to test the model is made.

3.6. Results and evaluation

Standard regression metrics such as mean squared error and mean absolute percentage error will be used to evaluate the models chosen. Model metrics will also be considered where appropriate, such as loss criterion for neural networks.

A random walk with drift will be built to use as a benchmark model to compare the trained models with [1,2]; in addition to using the random walk as a comparison, it helps us comprehend the predictability of our time series forecasts problem.

3.7. Report

The report will be written in parallel with the work carried out to allow for greater detail to be noted. The period will be set before the deadline that will be dedicated to completing the report to allow for feedback and leave enough time to address areas that can be improved. To avoid falling behind schedule, some extra time is allocated to objectives to take unexpected issues into account and avoid falling behind schedule.

3.8. Project meeting

A meeting is scheduled once every 2 weeks with the project supervisor to discuss progress made and possible feedback and advice.

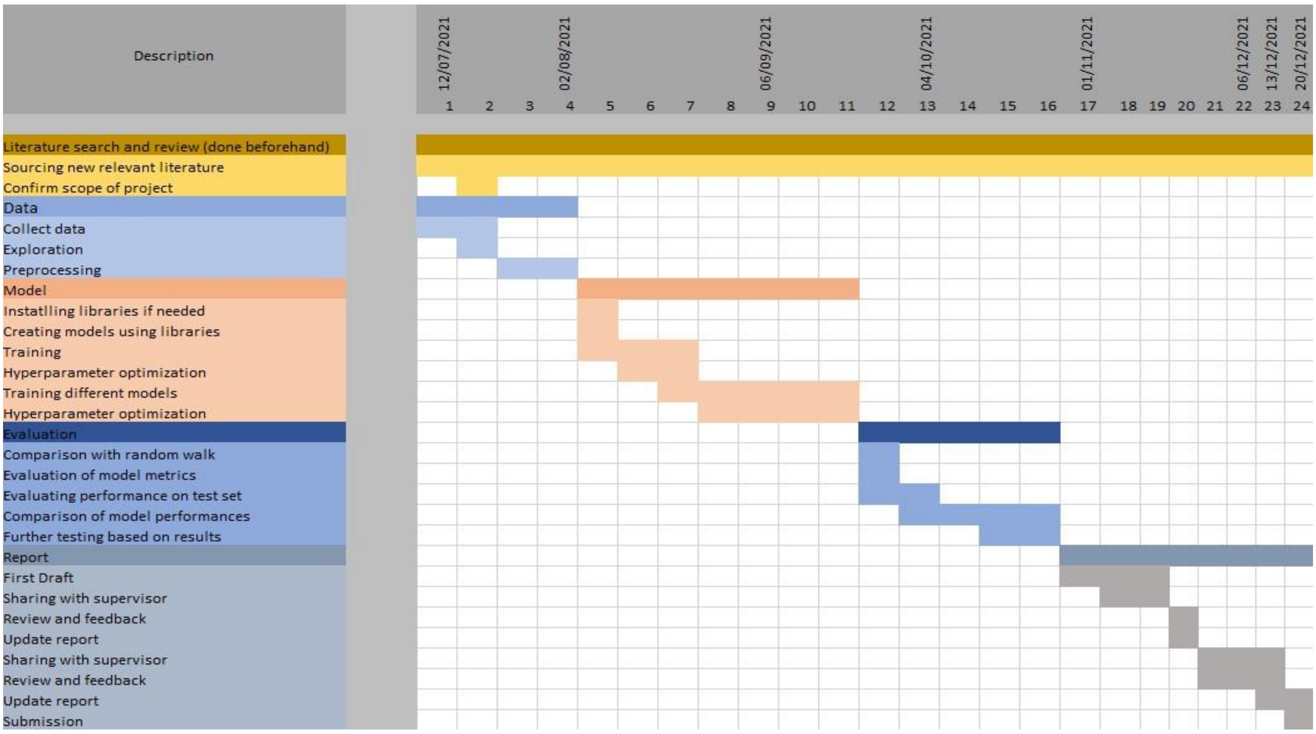
4. Risks

Description	Likelihood (1-3)	Consequence (1-5)	Impact (L x C)	Mitigation
Unable to achieve key milestones	1	5	5	Carry out more research and discuss with supervisor
Model training taking a long time	3	3	9	Adjust training method or use cloud services
Deliverables affected due to illness	2	2	4	Extra time will be allocated between milestones to take external factors into account
Implementation too difficult	1	4	4	Code carrying out similar tasks will be sourced and used as a reference
Loss of motivation	1	4	4	Discuss with supervisor
Code accidentally lost or deleted	1	5	5	Code regularly pushed on version control software e.g., GitHub

5. Workplan

Phase	Description	Outcome
Literature search and review	• Sourcing new relevant literature • Confirm scope of project	Updated Approach
Data acquisition and pre-processing	• Collect data • Exploration to make sure the dataset format is understood • Data processing	Data ready to be used for model training
Variation of restatements	• Create new variables and calculate the average time taken for restatements to converge	Use the average time as the window in model training
Modelling and evaluation	• Installing libraries if needed • Using libraries to create models • Train models on data • Hyperparameter optimization • Training different models	Models will be able to predict lagged months and provide all the necessary results for evaluation
Evaluation	• Comparison with random walk • Evaluation of model metrics and testing on test set • Comparison of model performance • Ideas for further testing	Provides required information to answer research question and written analysis.

Report writing	- Cycle of drafting, getting feedback from supervisor and addressing points until submission	Submit final report



References

[1] Plakandaras, V., Gupta, R., Gogas, P. and Papadimitriou, T. (2014). Forecasting the U.S. Real House Price Index. *SSRN Electronic Journal*.

[2] Milunovich G. (2019). Forecasting Australian Real House Price Index: A comparison of Time series and Machine learning methods.

[3] Hota, H., Handa, R. and Shrivas, A. (2017). Time Series Data Prediction Using Sliding Window Based RBF Neural

Network. *International Journal of Computational Intelligence Research*, [online] 13(5),

pp.1145–1156 [4] Chen, X., Wei, L. and Xu, J. (n.d.). *House Price Prediction Using LSTM*.

[5] Kutlualp, A. (n.d.). *Classical Machine Learning vs. Deep Learning Second Elizabethan Age Financial Portraiture PostEurope: Forecasting the GBP/USD Exchange Rate in the Era of Brexit*.

[6] V.Kecman (2013). Support Vector Machines: Theory and Applications.

[7] Programmersought.com. (2018). *[Machine Learning] Regression--Support Vector Regression (SVR) - Programmer Sought*.

[8] Debasish Basak, Srimanta Pal and Dipak Chandra Patranabis (2007). *Support Vector Regression*. [online] ResearchGate.

Available at: https://www.researchgate.net/publication/228537532_Support_Vector_Regression [Accessed 4 Aug. 2021].

[9] Bottou, L. and Lin, C.-J. (n.d.). *Support Vector Machine Solvers Support Vector Machine Solvers*. [online] . Available at: https://leon.bottou.org/publications/pdf/lin-2006.pdf [Accessed 4 Aug. 2021].

Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC) http://www.city.ac.uk/department-computer-science/research-ethics

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form*.* Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be ***provisional*** *– identifying the planned research as* likely to involve MINIMAL RISK. In such cases you must additionally seek ***full approval*** from the supervisor as the project progresses and details are established. ***Full approval*** must be acquired in writing, before beginning the planned research.

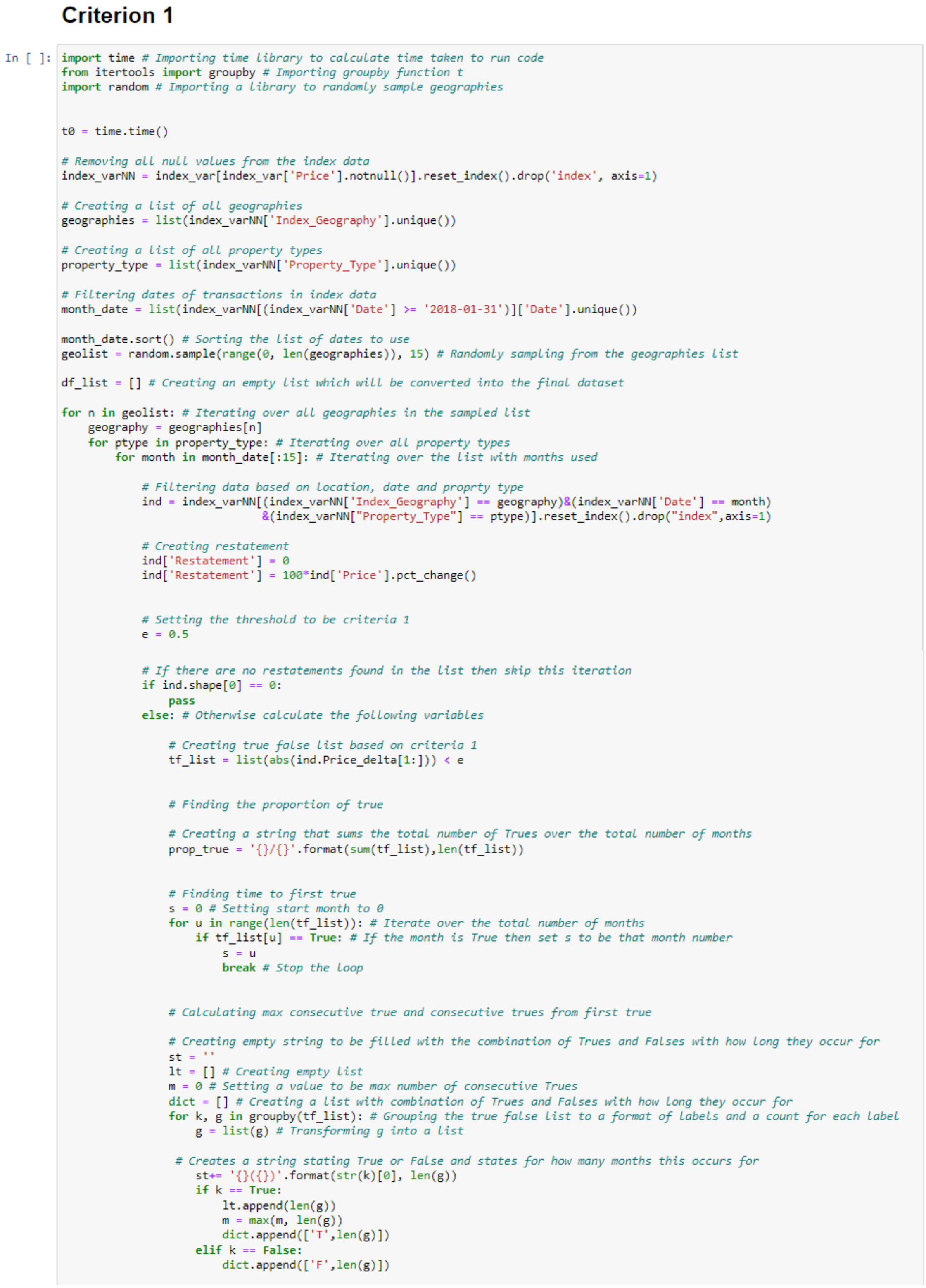
A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)?	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act?	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation?	**NO**

A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent?	**NO**

2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects?	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study?	**NO**
2.6	Does your research involve invasive or intrusive procedures?	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)?	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London?	**NO**

3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK.		*Delete as appropriate*
If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form. **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		
4	Does your project involve human participants or their identifiable personal data?	**NO**

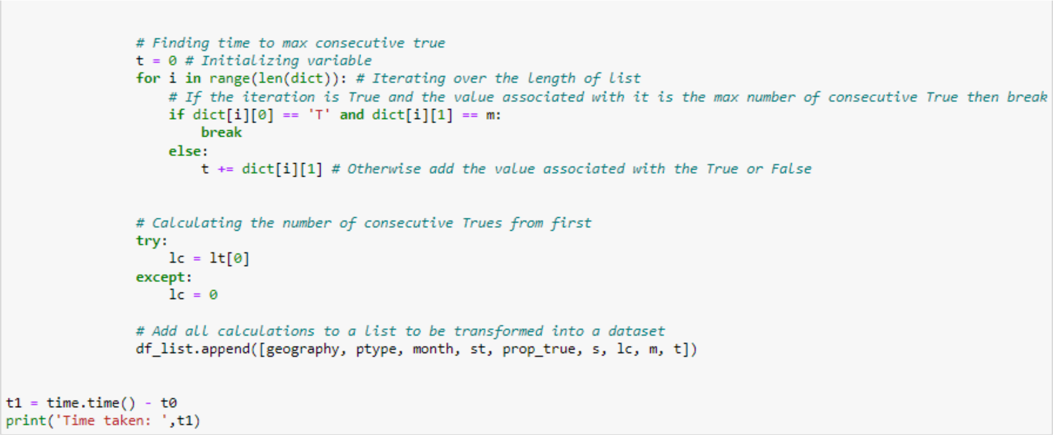
Appendix B – Code used to calculate new variables for variation of restatements



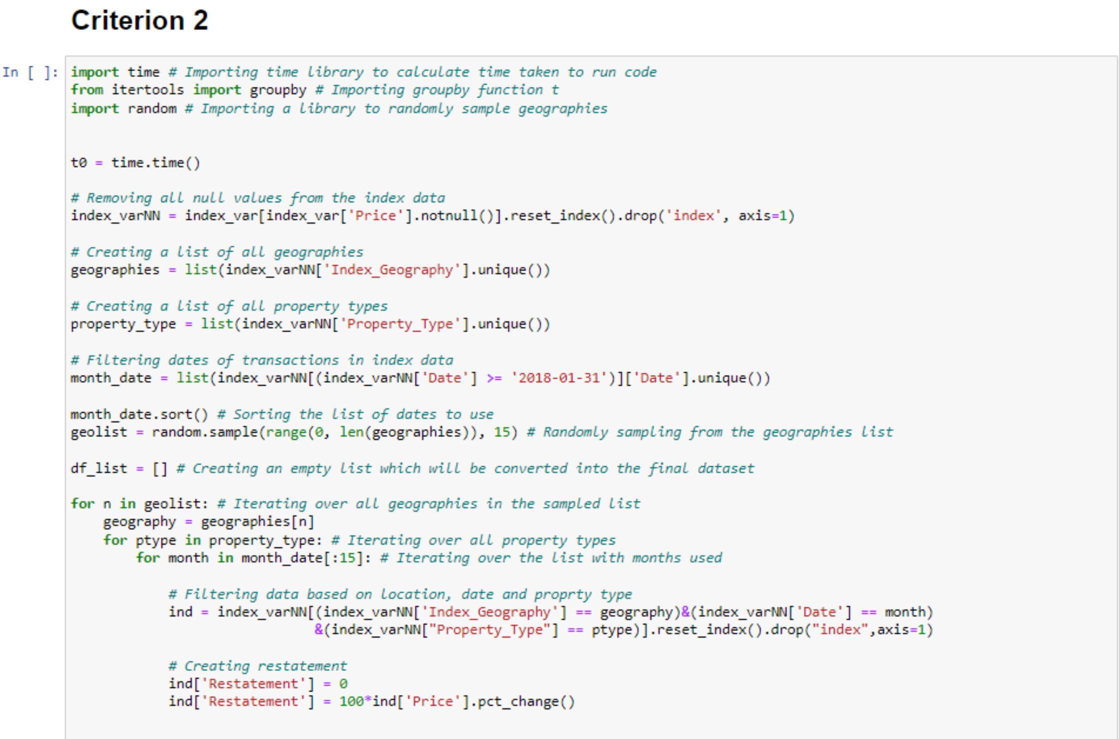
Criterion 1

In [ ]:

```
import time # Importing time library to calculate time taken to run code
from itertools import groupby # Importing groupby function t
import random # Importing a library to randomly sample geographies


t0 = time.time()

# Removing all null values from the index data
index_varNN = index_var[index_var['Price'].notnull()].reset_index().drop('index', axis=1)

# Creating a list of all geographies
geographies = list(index_varNN['Index_Geography'].unique())

# Creating a list of all property types
property_type = list(index_varNN['Property_Type'].unique())

# Filtering dates of transactions in index data
month_date = list(index_varNN[(index_varNN['Date'] >= '2018-01-31')]['Date'].unique())

month_date.sort() # Sorting the list of dates to use
geolist = random.sample(range(0, len(geographies)), 15) # Randomly sampling from the geographies list

df_list = [] # Creating an empty list which will be converted into the final dataset

for n in geolist: # Iterating over all geographies in the sampled list
    geography = geographies[n]
    for ptype in property_type: # Iterating over all property types
        for month in month_date[:15]: # Iterating over the list with months used

            # Filtering data based on location, date and proprty type
            ind = index_varNN[(index_varNN['Index_Geography'] == geography)&(index_varNN['Date'] == month)
                              &(index_varNN["Property_Type"] == ptype)].reset_index().drop("index",axis=1)

            # Creating restatement
            ind['Restatement'] = 0
            ind['Restatement'] = 100*ind['Price'].pct_change()


            # Setting the threshold to be criteria 1
            e = 0.5


            # If there are no restatements found in the list then skip this iteration
            if ind.shape[0] == 0:
                pass
            else: # Otherwise calculate the following variables

                # Creating true false list based on criteria 1
                tf_list = list(abs(ind.Price_delta[1:])) < e


                # Finding the proportion of true

                # Creating a string that sums the total number of Trues over the total number of months
                prop_true = '{}/{}'.format(sum(tf_list),len(tf_list))


                # Finding time to first true
                s = 0 # Setting start month to 0
                for u in range(len(tf_list)): # Iterate over the total number of months
                    if tf_list[u] == True: # If the month is True then set s to be that month number
                        s = u
                        break # Stop the loop


                # Calculating max consecutive true and consecutive trues from first true

                # Creating empty string to be filled with the combination of Trues and Falses with how long they occur for
                st = ''
                lt = [] # Creating empty list
                m = 0 # Setting a value to be max number of consecutive Trues
                dict = [] # Creating a list with combination of Trues and Falses with how long they occur for
                for k, g in groupby(tf_list): # Grouping the true false list to a format of labels and a count for each label
                    g = list(g) # Transforming g into a list

                 # Creates a string stating True or False and states for how many months this occurs for
                    st+= '{}({})'.format(str(k)[0], len(g))
                    if k == True:
                        lt.append(len(g))
                        m = max(m, len(g))
                        dict.append(['T',len(g)])
                    elif k == False:
                        dict.append(['F',len(g)])
```



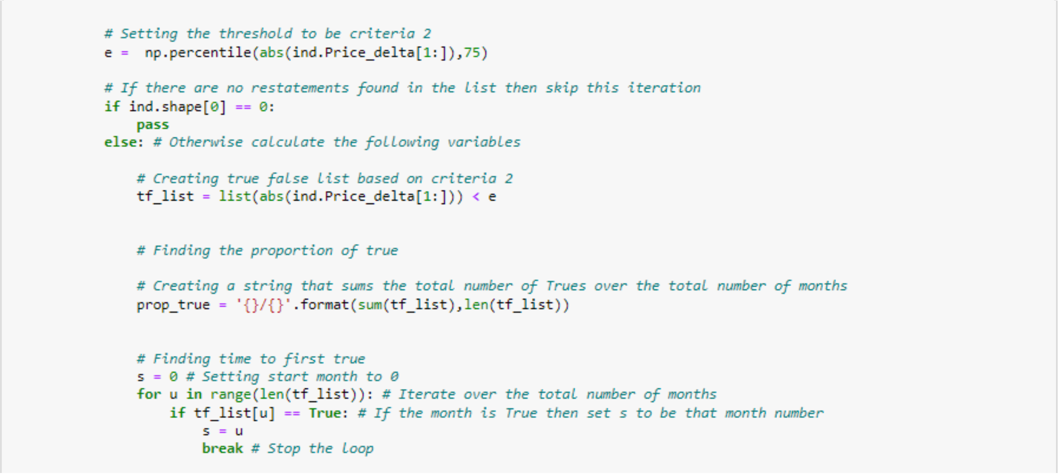
```
                # Finding time to max consecutive true
                t = 0 # Initializing variable
                for i in range(len(dict)): # Iterating over the length of list
                    # If the iteration is True and the value associated with it is the max number of consecutive True then break
                    if dict[i][0] == 'T' and dict[i][1] == m:
                        break
                    else:
                        t += dict[i][1] # Otherwise add the value associated with the True or False


                # Calculating the number of consecutive Trues from first
                try:
                    lc = lt[0]
                except:
                    lc = 0

                # Add all calculations to a list to be transformed into a dataset
                df_list.append([geography, ptype, month, st, prop_true, s, lc, m, t])


t1 = time.time() - t0
print('Time taken: ',t1)
```



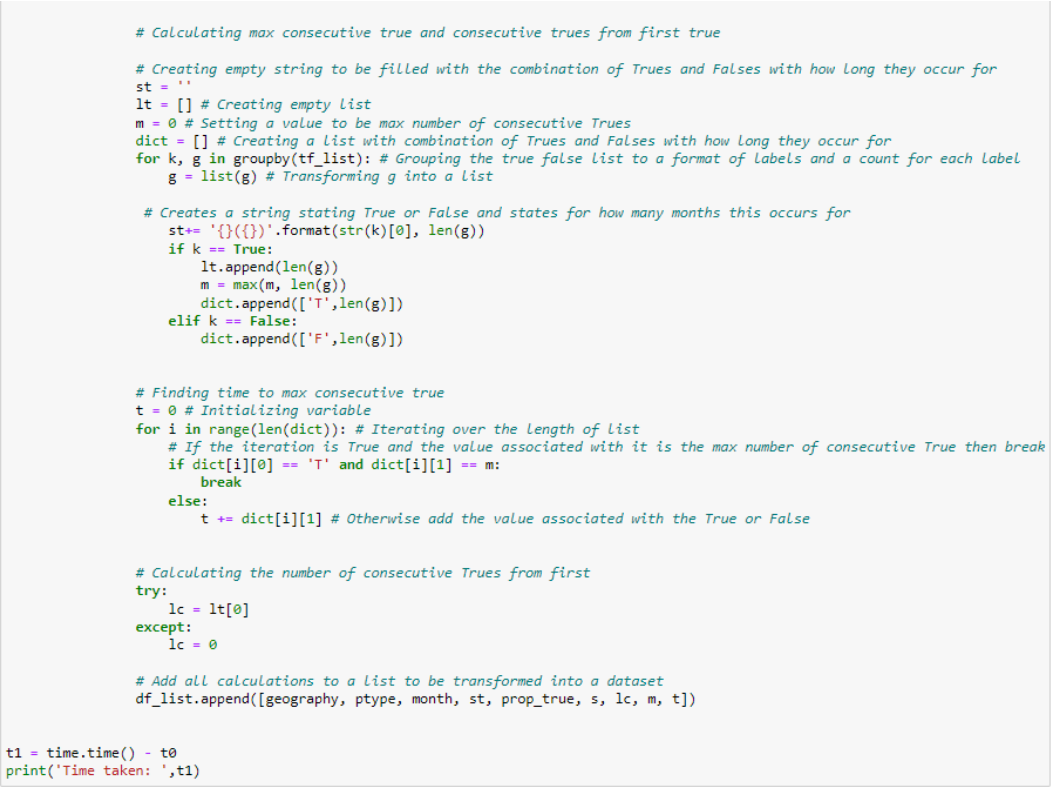
Criterion 2

In [ ]:

```
import time # Importing time library to calculate time taken to run code
from itertools import groupby # Importing groupby function t
import random # Importing a library to randomly sample geographies


t0 = time.time()

# Removing all null values from the index data
index_varNN = index_var[index_var['Price'].notnull()].reset_index().drop('index', axis=1)

# Creating a list of all geographies
geographies = list(index_varNN['Index_Geography'].unique())

# Creating a list of all property types
property_type = list(index_varNN['Property_Type'].unique())

# Filtering dates of transactions in index data
month_date = list(index_varNN[(index_varNN['Date'] >= '2018-01-31')]['Date'].unique())

month_date.sort() # Sorting the list of dates to use
geolist = random.sample(range(0, len(geographies)), 15) # Randomly sampling from the geographies list

df_list = [] # Creating an empty list which will be converted into the final dataset

for n in geolist: # Iterating over all geographies in the sampled list
    geography = geographies[n]
    for ptype in property_type: # Iterating over all property types
        for month in month_date[:15]: # Iterating over the list with months used

            # Filtering data based on location, date and proprty type
            ind = index_varNN[(index_varNN['Index_Geography'] == geography)&(index_varNN['Date'] == month)
                              &(index_varNN["Property_Type"] == ptype)].reset_index().drop("index",axis=1)

            # Creating restatement
            ind['Restatement'] = 0
            ind['Restatement'] = 100*ind['Price'].pct_change()
```

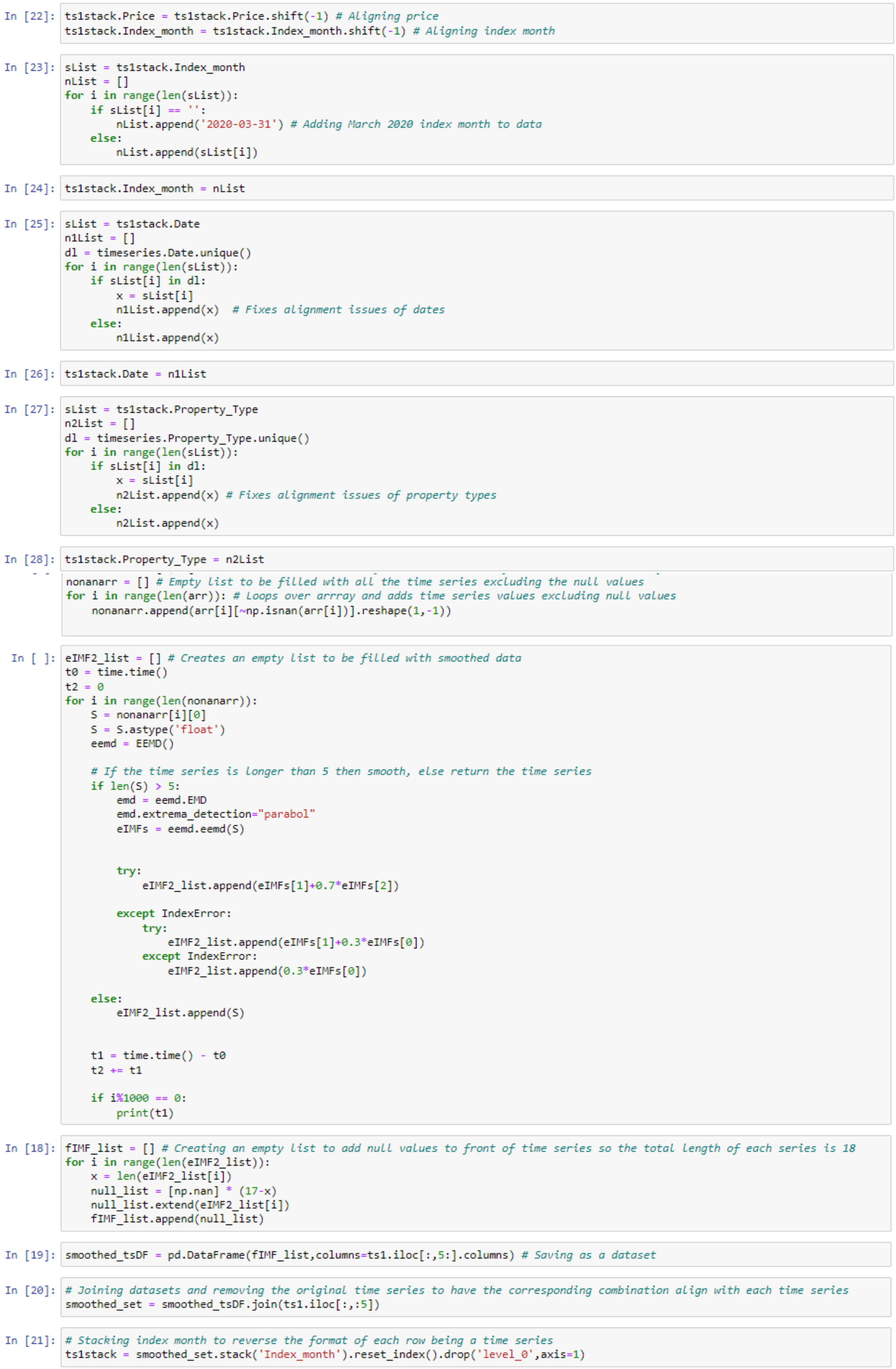


```
            # Setting the threshold to be criteria 2
            e =  np.percentile(abs(ind.Price_delta[1:]),75)

            # If there are no restatements found in the list then skip this iteration
            if ind.shape[0] == 0:
                pass
            else: # Otherwise calculate the following variables

                # Creating true false list based on criteria 2
                tf_list = list(abs(ind.Price_delta[1:])) < e


                # Finding the proportion of true

                # Creating a string that sums the total number of Trues over the total number of months
                prop_true = '{}/{}'.format(sum(tf_list),len(tf_list))


                # Finding time to first true
                s = 0 # Setting start month to 0
                for u in range(len(tf_list)): # Iterate over the total number of months
                    if tf_list[u] == True: # If the month is True then set s to be that month number
                        s = u
                        break # Stop the loop
```

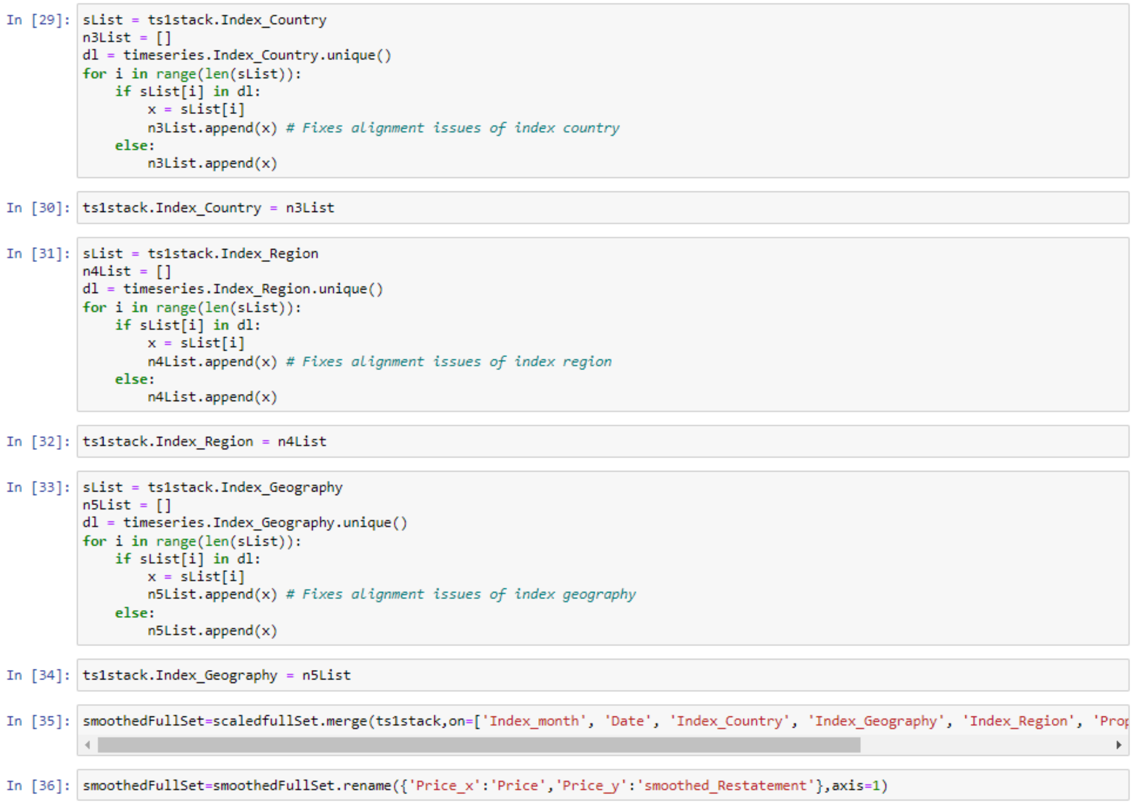


```
                # Calculating max consecutive true and consecutive trues from first true

                # Creating empty string to be filled with the combination of Trues and Falses with how long they occur for
                st = ''
                lt = [] # Creating empty list
                m = 0 # Setting a value to be max number of consecutive Trues
                dict = [] # Creating a list with combination of Trues and Falses with how long they occur for
                for k, g in groupby(tf_list): # Grouping the true false list to a format of labels and a count for each label
                    g = list(g) # Transforming g into a list

                 # Creates a string stating True or False and states for how many months this occurs for
                    st+= '{}({})'.format(str(k)[0], len(g))
                    if k == True:
                        lt.append(len(g))
                        m = max(m, len(g))
                        dict.append(['T',len(g)])
                    elif k == False:
                        dict.append(['F',len(g)])


                # Finding time to max consecutive true
                t = 0 # Initializing variable
                for i in range(len(dict)): # Iterating over the length of list
                    # If the iteration is True and the value associated with it is the max number of consecutive True then break
                    if dict[i][0] == 'T' and dict[i][1] == m:
                        break
                    else:
                        t += dict[i][1] # Otherwise add the value associated with the True or False


                # Calculating the number of consecutive Trues from first
                try:
                    lc = lt[0]
                except:
                    lc = 0

                # Add all calculations to a list to be transformed into a dataset
                df_list.append([geography, ptype, month, st, prop_true, s, lc, m, t])


t1 = time.time() - t0
print('Time taken: ',t1)
```

Appendix C – Code used to smooth data and transform structure into dataset



```
In [22]: ts1stack.Price = ts1stack.Price.shift(-1) # Aligning price
         ts1stack.Index_month = ts1stack.Index_month.shift(-1) # Aligning index month

In [23]: sList = ts1stack.Index_month
         nList = []
         for i in range(len(sList)):
             if sList[i] == '':
                 nList.append('2020-03-31') # Adding March 2020 index month to data
             else:
                 nList.append(sList[i])

In [24]: ts1stack.Index_month = nList

In [25]: sList = ts1stack.Date
         n1List = []
         dl = timeseries.Date.unique()
         for i in range(len(sList)):
             if sList[i] in dl:
                 x = sList[i]
                 n1List.append(x)  # Fixes alignment issues of dates
             else:
                 n1List.append(x)

In [26]: ts1stack.Date = n1List

In [27]: sList = ts1stack.Property_Type
         n2List = []
         dl = timeseries.Property_Type.unique()
         for i in range(len(sList)):
             if sList[i] in dl:
                 x = sList[i]
                 n2List.append(x) # Fixes alignment issues of property types
             else:
                 n2List.append(x)

In [28]: ts1stack.Property_Type = n2List

         nonanarr = [] # Empty list to be filled with all the time series excluding the null values
         for i in range(len(arr)): # Loops over arrray and adds time series values excluding null values
             nonanarr.append(arr[i][~np.isnan(arr[i])].reshape(1,-1))

In [ ]: eIMF2_list = [] # Creates an empty list to be filled with smoothed data
        t0 = time.time()
        t2 = 0
        for i in range(len(nonanarr)):
            S = nonanarr[i][0]
            S = S.astype('float')
            eemd = EEMD()

            # If the time series is longer than 5 then smooth, else return the time series
            if len(S) > 5:
                emd = eemd.EMD
                emd.extrema_detection="parabol"
                eIMFs = eemd.eemd(S)


                try:
                    eIMF2_list.append(eIMFs[1]+0.7*eIMFs[2])

                except IndexError:
                    try:
                        eIMF2_list.append(eIMFs[1]+0.3*eIMFs[0])
                    except IndexError:
                        eIMF2_list.append(0.3*eIMFs[0])

            else:
                eIMF2_list.append(S)


            t1 = time.time() - t0
            t2 += t1

            if i%1000 == 0:
                print(t1)

In [18]: fIMF_list = [] # Creating an empty list to add null values to front of time series so the total length of each series is 18
         for i in range(len(eIMF2_list)):
             x = len(eIMF2_list[i])
             null_list = [np.nan] * (17-x)
             null_list.extend(eIMF2_list[i])
             fIMF_list.append(null_list)

In [19]: smoothed_tsDF = pd.DataFrame(fIMF_list,columns=ts1.iloc[:,5:].columns) # Saving as a dataset

In [20]: # Joining datasets and removing the original time series to have the corresponding combination align with each time series
         smoothed_set = smoothed_tsDF.join(ts1.iloc[:,:5])

In [21]: # Stacking index month to reverse the format of each row being a time series
         ts1stack = smoothed_set.stack('Index_month').reset_index().drop('level_0',axis=1)
```



```
In [29]: sList = ts1stack.Index_Country
         n3List = []
         dl = timeseries.Index_Country.unique()
         for i in range(len(sList)):
             if sList[i] in dl:
                 x = sList[i]
                 n3List.append(x) # Fixes alignment issues of index country
             else:
                 n3List.append(x)

In [30]: ts1stack.Index_Country = n3List

In [31]: sList = ts1stack.Index_Region
         n4List = []
         dl = timeseries.Index_Region.unique()
         for i in range(len(sList)):
             if sList[i] in dl:
                 x = sList[i]
                 n4List.append(x) # Fixes alignment issues of index region
             else:
                 n4List.append(x)

In [32]: ts1stack.Index_Region = n4List

In [33]: sList = ts1stack.Index_Geography
         n5List = []
         dl = timeseries.Index_Geography.unique()
         for i in range(len(sList)):
             if sList[i] in dl:
                 x = sList[i]
                 n5List.append(x) # Fixes alignment issues of index geography
             else:
                 n5List.append(x)

In [34]: ts1stack.Index_Geography = n5List

In [35]: smoothedFullSet=scaledfullSet.merge(ts1stack,on=['Index_month', 'Date', 'Index_Country', 'Index_Geography', 'Index_Region', 'Prop

In [36]: smoothedFullSet=smoothedFullSet.rename({'Price_x':'Price','Price_y':'smoothed_Restatement'},axis=1)
```