UNIVERSITY OF CALIFORNIA
Los Angeles

Predicting Long-Term U.S. Housing Price Trends Using a Long Short-Term Memory Neural Network

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Harrison DiStefano

2022

ABSTRACT OF THE THESIS

Predicting Long-Term U.S. Housing Price Trends Using a Long Short-Term Memory Neural Network

by

Harrison DiStefano
Master of in Applied Statistics
University of California, Los Angeles, 2022
Professor Ying Nian Wu, Chair

Housing prices affect everyone. In this paper we establish a potential method of predicting long term home sale prices in the United States using an LSTM neural network model. We took publicly available macro economic data, then massaged it into a manageable form using cubic spline interpolation and logarithmic differencing. We then introduce the LSTM model using feature standardization, min-max normalization, and an Adam optimizer for backpropogation. After training the network on preceding data, we found that the network was able to provide predictions that coincided with similar market movements.

The thesis of Harrison DiStefano is approved.

Rick Schoenberg

Mark S. Handcock

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Housing prices affect everyone–whether you rent or buy, more expensive housing means a higher cost of living. In the last decade alone, the median sale price of a house in the United States has grown from $238,000 to $433,100, an increase of over 90%, while real median household income has increased by only 17%.[UU63a] This discrepancy between spending power and home prices has made it more difficult for the typical American to become a homeowner. The cost of a home is influenced by a number of macro economic factors–from those that affect consumer spending such as inflation and company earnings, to those that directly affect the cost of borrowing to purchase a home, such as long-term interest rates. These economic factors are publicly available and can potentially be used to predict long term trends in the housing market, helping prospective buyers to make more informed decisions when purchasing a home.

One current way to analyze these trends is by using Long Short-Term Memory (LSTM) recurrent neural networks. LSTMs are suited to time series data like price values because, unlike standard neural networks, they have feedback connections that allow for information to be stored over a period of time.

In this paper, we implement an LSTM to predict long term trends in U.S. housing sale prices. We take macroeconomic data compiled from various official sources, massage it into a format agreeable to the model, and train the LSTM on the years leading up to the last year of the data. We then use this model to predict housing prices for the final year of data and compare that to where prices actually went.

This manuscript is organized as follows. Chapter 2 covers the data overview, cleaning, and exploration. Chapter 3 demonstrates the theory behind the LSTM model. Chapter 4 covers the training of the neural network and the results. Chapter 5 is the conclusion statement to the paper.

CHAPTER 2

Data

2.1 Data Source

The United States housing price data and supply data used for training and testing is retrieved from the Federal Reserve Bank of St. Louis (FRED), sourced from the relevant official U.S. bureaus [UU63a][UU63b]. These time series data sets contain 236 quarterly median housing prices in the U.S. from January 1963 until January 2022 and 712 monthly new U.S. housing supply ratios from January 1963 until April 2022. Economic data is taken from Robert Shiller's data set used in his book *Irrational Exuberance*[Shi] and consists of 1,816 rows of monthly economic data from January 1871 until May 2022.

2.2 Data Details

Table 2.1: Data Attribute Information

#	Label	Description
1	DATE	First date of the period to which data corresponds, e.g. 01-01-1963 for first fiscal quarter of 1963 if quarterly data
2	MSPUS	Median U.S. housing prices over the given fiscal quarter in DATE. Target variable for this paper
3	MSACSR	Monthly supply of new U.S. houses, calculated by ratio of new houses for sale to new houses sold
4	S&P Comp.	Value of S&P Composite index on the given date
5	Dividend	Total dividends per share of the S&P Comp.
6	Earnings	Earnings per share of the S&P Comp.
7	CPI	Weighted average price of basket of consumer goods
8	Long Interest Rate GS10	Market yield on U.S. Treasury 10-Year Securities
9	Real Price	Value of S&P Comp. adjusted for inflation
10	Real Dividend	Dividend adjusted for inflation
11	Real Total Return Price	Cumulative total return of S&P Comp. since 1963
12	Real Earnings	Earnings adjusted for inflation
13	CAPE	CAPE ratio of S&P Comp.
14	Total Return CAPE	CAPE with dividends reinvested
15	Excess CAPE Yield	Inverse of CAPE ratio
16	Monthly Total Bond Returns	Shiller Monthly Total Bond Returns
17	Real Total Bond Returns	Inflation adjusted Shiller bond returns
18	10 Year Stock Real Return	Moving 10 year infl. adjusted S&P Comp. returns
19	10 Year Bonds Real Return	Moving 10 year infl. adjusted Shiller returns
20	10 Year Excess Returns	Inverse of 10 year stock real returns

The data consists of quarterly median U.S. housing prices, the monthly supply of new U.S. houses, and 17 monthly U.S. macroeconomic statistics for a total of 1816 months from January 1871 until May 2022. However, due to the price data being quarterly and the economic data being monthly, price information is missing from two out of every three months. We will fix this in the Data Cleaning section.

Figure 2.1 shows a graph of our target variable, the median sale price of houses in the U.S. (MSPUS) over time.

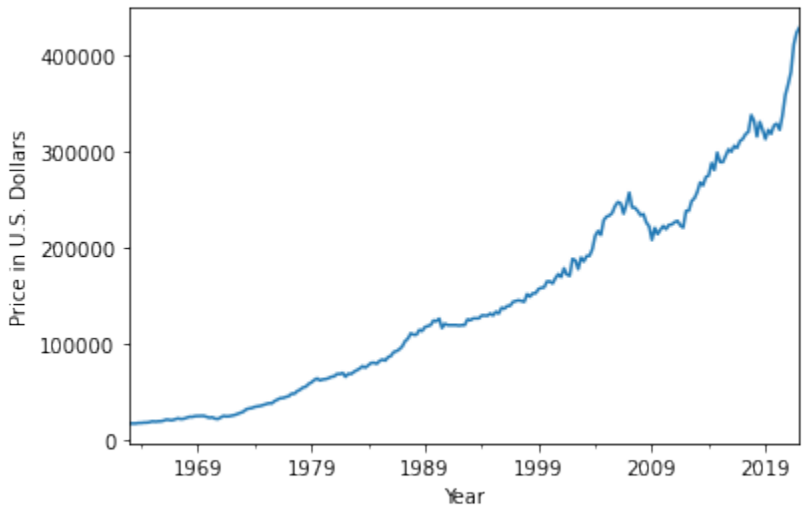

Figure 2.1: Median Sale Price of U.S. Houses Over Time

2.3 Data Cleaning

We begin by cleaning the data to prepare it for analysis and modeling. The data comes from several sources and so needs to be massaged into a cohesive unit so that it can be passed easily through our tools. The data also needs to be processed in order to match the assumptions of our models.

As mentioned previously, there are 1816 rows (months) of economic data and 712 of housing price data. Since there are fewer months of pricing data–the target variable–than economic data, we remove all rows with dates falling outside the range of the pricing data, leaving 712 rows.

2.3.1 Null Values

Part of massaging our dataset includes handling missing values, either by filling or removing them from the dataset in order to perform analysis. We will both fill through interpolation and remove some values, columns, and rows that are less necessary.

2.3.1.1 Interpolation

The economic data is monthly and the U.S. median house price data is quarterly as shown in Table 2.2

Table 2.2: Seven Months of U.S. Housing Supply and Price Data with Missing Values

DATE	MSACSR	MSPUS
1963-01-01	4.7	17800.00
1963-02-01	6.6	NULL
1963-03-01	6.4	NULL
1963-04-01	5.3	18000.00
1963-05-01	5.1	NULL
1963-06-01	6.0	NULL
1963-07-01	4.6	17900.00

To avoid removing all of these null values and with them two-thirds of the granularity of economic data, we interpolate between the known price values to fill what is missing. This will make all of the data monthly.

To do this, we use a natural cubic spline interpolation[Wol]. This is done by constructing a spline $f(x)$ on the data points $(x_1, ..., x_{n+1})$ consisting of n third degree piecewise polynomials:

$$f(x) = \begin{cases} a_1x^3 + b_1x^2 + c_1x + d_1 & \text{if } x \in [x_1, x_2] \\ a_2x^3 + b_2x^2 + c_2x + d_2 & \text{if } x \in (x_2, x_3] \\ \dots & \\ a_nx^3 + b_nx^2 + c_nx + d_n & \text{if } x \in (x_n, x_{n+1}] \end{cases}$$

Where a_i, b_i, c_i, and d_i are constant coefficients for the ith polynomial. These coefficients are calculated with the following set of constraints in a natural cubic spline: 1) the first and second derivatives of all polynomials are identical at the points where they touch adjacent polynomials and 2) the second derivative of the first and last polynomials are set to zero at the first and last points respectively. These constraints keep the interpolation smooth between points.

The results of interpolation are shown in Table 2.3 and Figure 2.2

Table 2.3: Seven Months of U.S. Housing Supply and Price Data with Interpolated Values

DATE	MSACSR	MSPUS
1963-01-01	4.7	17800.00
1963-02-01	6.6	18012.73
1963-03-01	6.4	18054.36
1963-04-01	5.3	18000.00
1963-05-01	5.1	17912.32
1963-06-01	6.0	17855.47
1963-07-01	4.6	17900.00

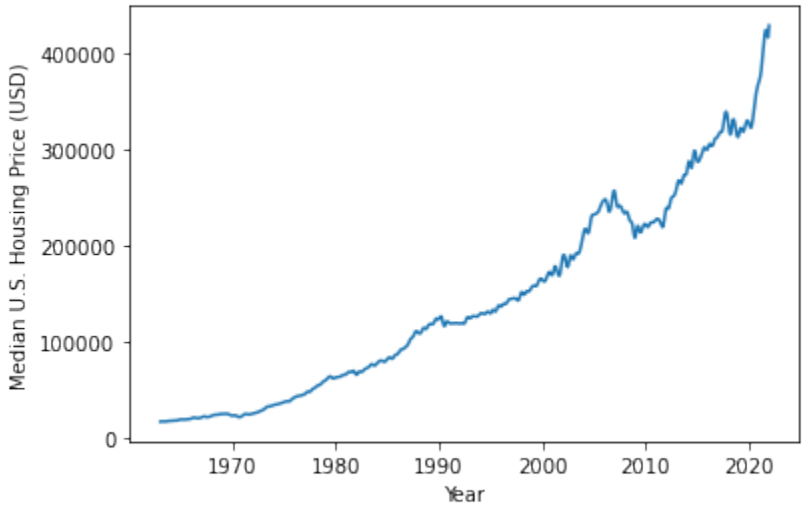

Figure 2.2: Cubic Spline Interpolation Results on Housing Prices

2.3.1.2 Removing Remaining Null Values

Figure 2.3 shows the remainder of null values are located mostly in the three columns '10 Year Stock Real Return', '10 Year Bonds Real Return', and '10 Year Excess Returns' with 120 missing values from the years 2012 through 2022 corresponding to the 10 years forward that haven't been 'completed.' The remaining missing values consist of 1 each from the 'Dividend' and 'Real Dividend' columns, as well as 4 from 'Earnings', 'Real Earnings', and 'Real TR Scaled Earnings' respectively.

Because the majority of null values come from only 3 of our 18 features, and there is a

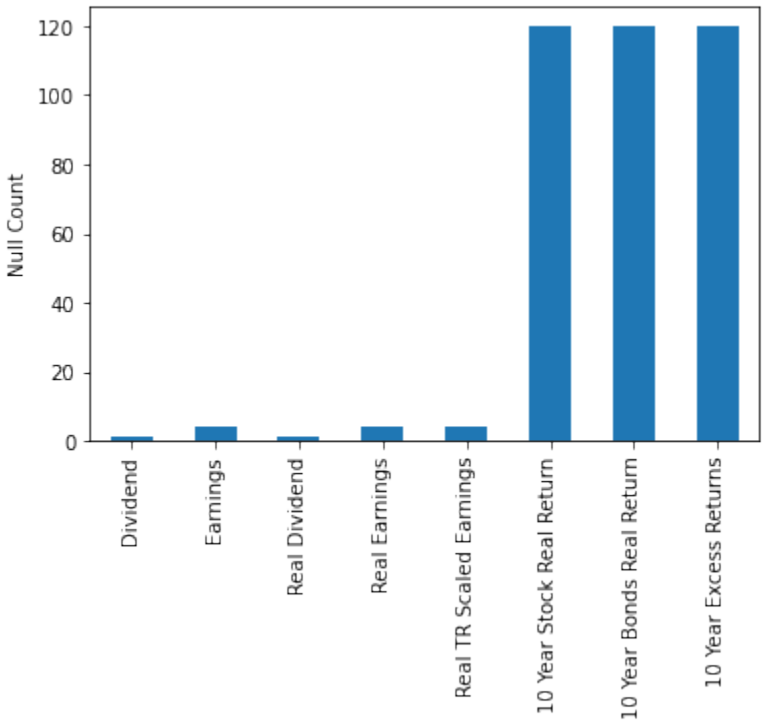

Figure 2.3: Missing Values by Column

possibility of some target variable leakage in these forward-looking values, we remove all 3 of the '10 Year' columns. This leaves only 14 more null values to be managed and maintains all 712 rows of data.

When the remaining null values are plotted by year in which they occur as in Figure 2.4, we see that they all are in the tail of the dataset in the year 2022.

All 4 null values are in the last 4 months of the dataset, meaning we can remove these rows without creating holes in the middle of the time series and only losing less than 1 percent of the data. This takes the total number of rows from 712 to 708.

2.4 Differencing

Figure 2.5 shows several features plotted against time. Notice the apparent upward trends in many of the features including CPI, S&P Composite, and our target variable MSPUS. We

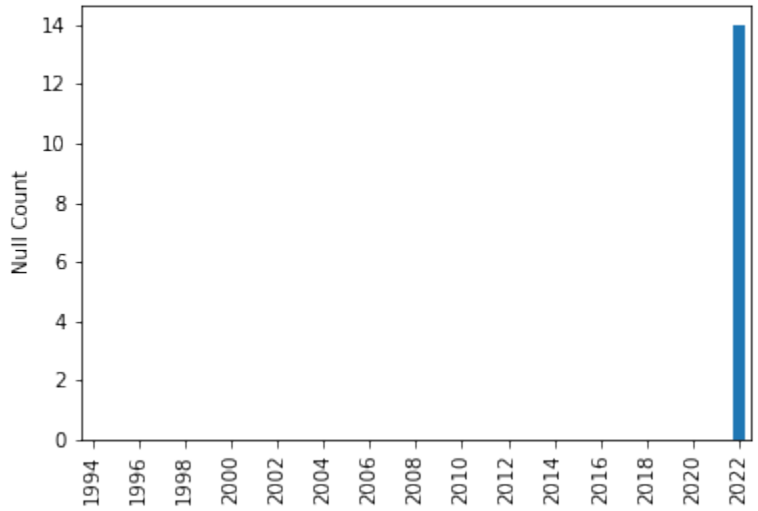

Figure 2.4: Missing Values by Year

apply differencing techniques to reduce this auto-correlation.[HA21]

2.4.1 First Order Differencing

A first order price difference calculated by subtracting each value by the previous value in the time series converts the data to a series of changes, one at each time-step x_t.

$$\hat{x}_t = x_t - x_{t-1}$$

The result of first order differencing on the median housing price data is shown in Figure 2.6, where each value represents the month to month change in median home prices. Note the heteroscedasticity, with variance increasing as time progresses. This is due to price swings tending to be larger as housing gets more expensive over time.

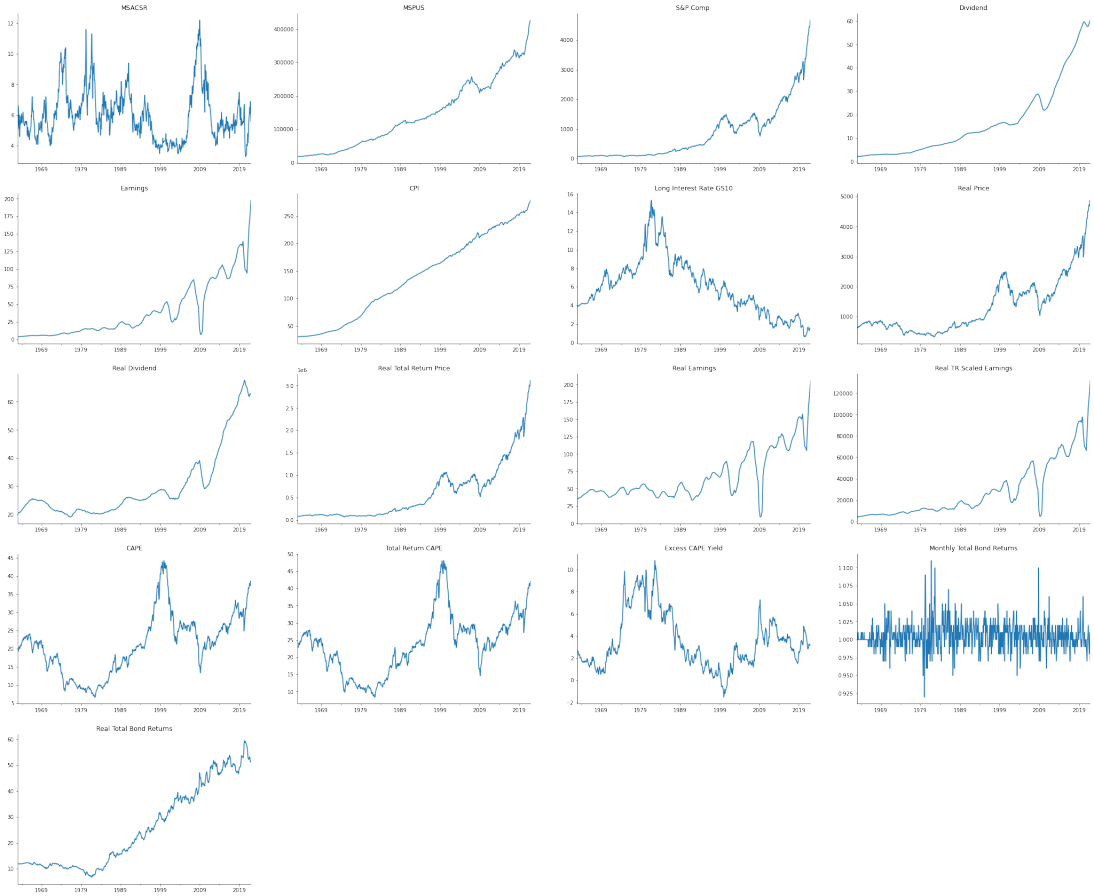

Figure 2.5: Select Timeseries Feature Plot

2.4.2 Logarithmic Differencing

We would like the variance to remain constant over the course of the time series, this is done by instead logarithmic (log) differencing, also known as converting the price data to log returns.

$$\hat{x}_t = \ln x_t - \ln x_{t-1}$$

Where $\hat{x}_t$ is the symmetric percent change from x at time $t-1$ to time t.

Note that the inverse can be calculated to restore the original data, to reproduce x_t from $\hat{x}_t$, as long as the first value x_0 of the original data vector is known, using the following

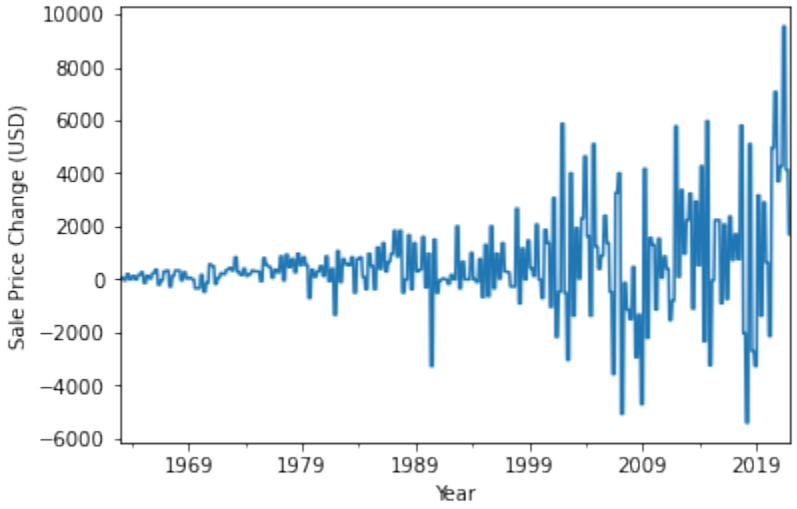

Figure 2.6: Monthly First Order Difference of Median U.S. House Prices

formula:

$$x_t = x_0 \exp \sum_{i=1}^{t} \hat{x}_i$$

This is used later to transform our log return results into price values.

Figure 2.7 shows median housing price data after log differencing. Taking the log has removed the trend just as first order differencing did, but also reduced apparent changes in variance and made the data more homoscedastic. An extra benefit of log differencing is that the resulting values represent a symmetric percent change. This means that negative and positive values change the direction, but not the value of the percent change.[CA17]

Additionally, as shown in Figure 2.8, most of the autocorrelation caused by trending has been reduced. The remaining autocorrelation is only strong in the first 3 or 4 months, with a small amount near the 10 month range. We will use these values when building our LSTM model.

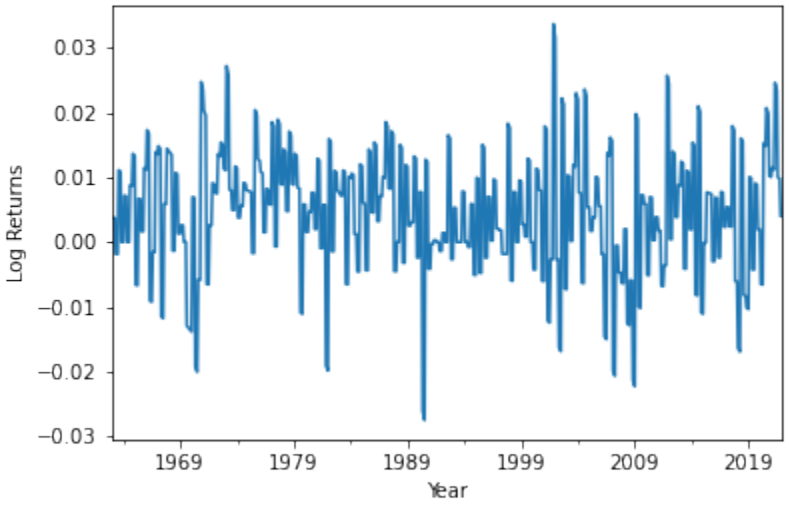

Figure 2.7: Log Differencing of Median Housing Prices

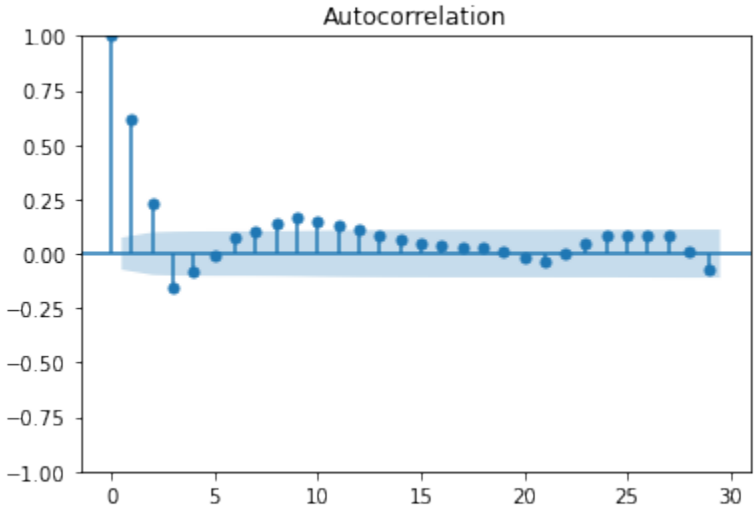

Figure 2.8: Autocorrelation of Transformed Housing Prices

2.4.3 Application to Data

We apply logarithmic differencing to each feature column except 'Monthly Total Bond Returns' and 'Long Interest Rate GS10'; Monthly total bond returns because it is already differenced, and we apply only first order differencing to the long term interest rate since the month to month changes are generally a fixed amount rather than dependent on the

previous month.

The results of these transformations are shown in Figure 2.9. These transformations removed trends, reduced heteroscedasticity, and made the data distributions more normal as shown in the data exploration section. One thing to note is the large swings around the financial crisis of 2008, leading to some outlier values.

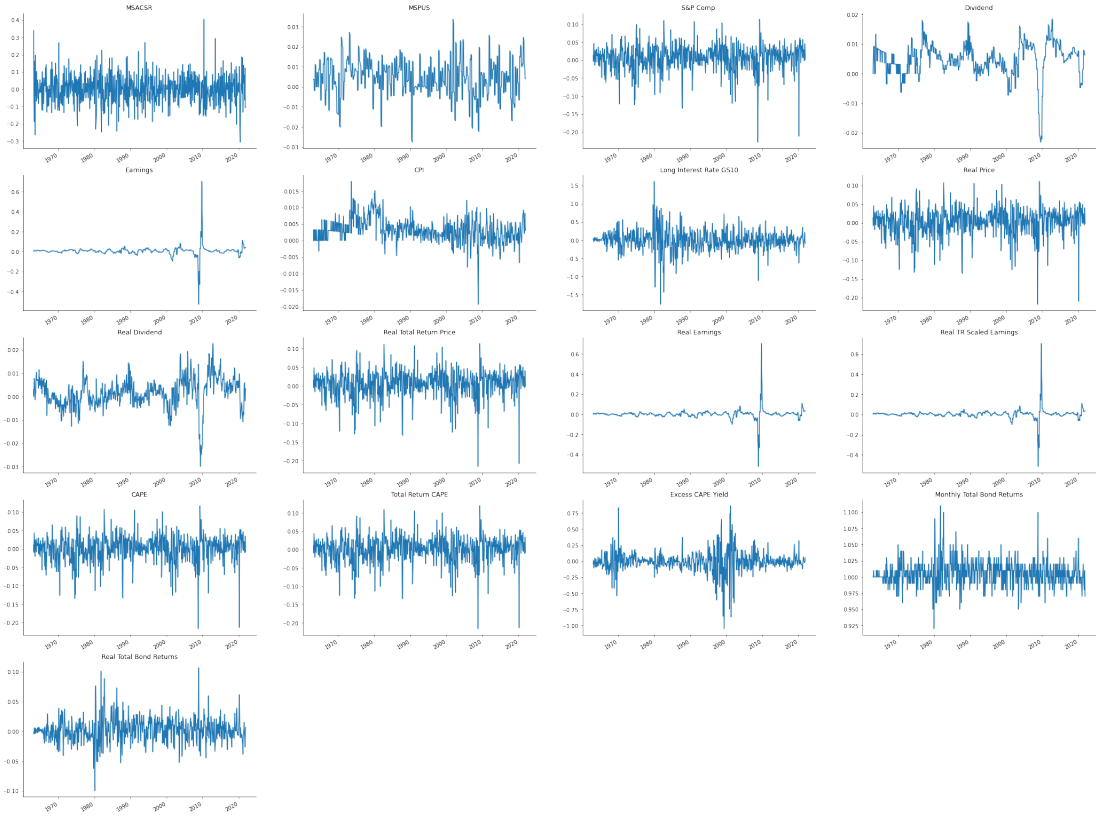

Figure 2.9: Transformed Feature Plot

2.5 Exploratory Analysis

In this section we look at the data more in depth in preparation for our model.

Figures 2.10 and 2.11 show kernel density estimation plots of the data columns before transformations, and after, respectively. These figures show a change from mostly right-

skewed distributions to more normally distributed data after differencing was applied. Having the data in a more normally distributed form allows us to use tests and models that assume more independent, normally distributed data.

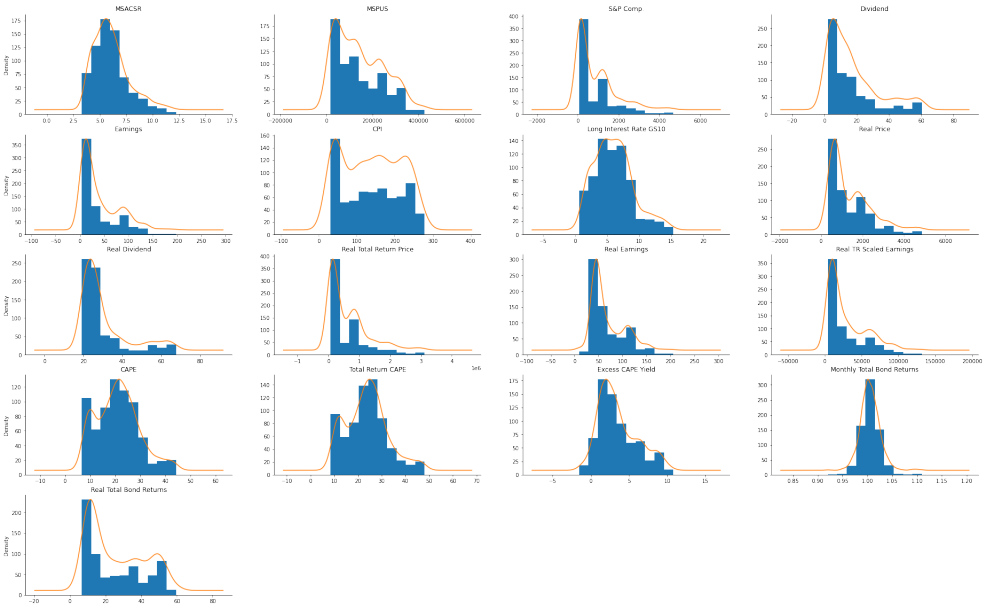

Figure 2.10: Feature KDE Plots Before Transformations

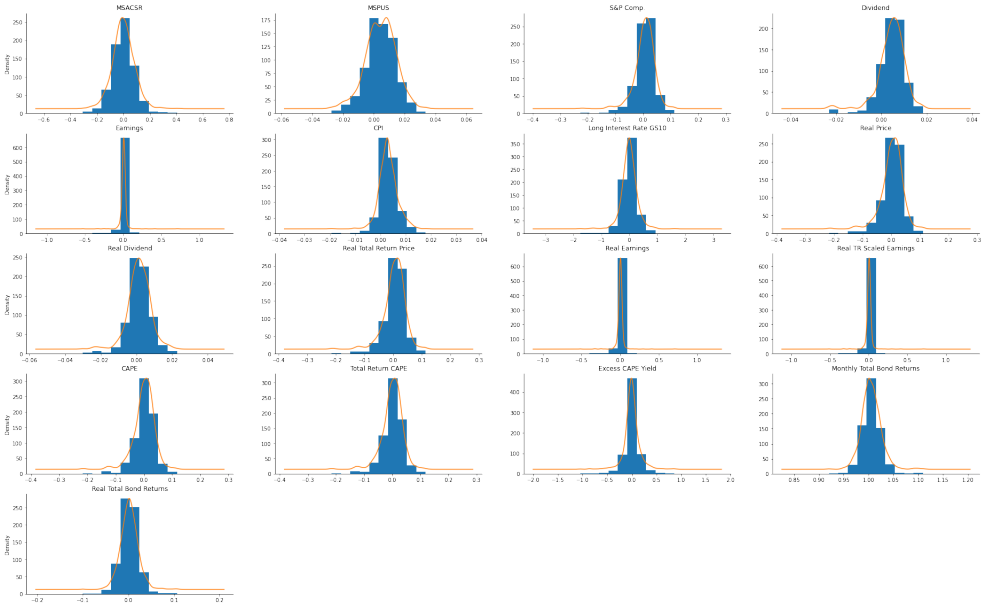

Figure 2.11: Feature KDE Plots After Transformations

The removal of trends by differencing helps to reduce spurious correlations caused by these trends in time series data.[Yul26]

We can then compute correlations between our features using the Kendall rank correlation coefficient. This is preferred over the typical Pearson correlation as it is more robust and makes fewer assumptions about the data.[Ken38] Kendall rank correlation is calculated by subtracting the number of discordant pairs between two sets of observations and the number of concordant pairs, then dividing by the total number of pairs. If $(x_1, y_1), ..., (x_n, y_n)$ are a set of observations of joint random variables X and Y, then a pair of observations (x_i, y_i) and (x_j, y_j) are concordant if either both $x_i > x_j$ and $y_i > y_j$ or both $x_i < x_j$ and $y_i < y_j$. The pairs are discordant otherwise. The Kendall's rank coefficient τ can be calculated by

$$\tau = \frac{2}{n(n-1)} \sum_{i<j} \operatorname{sgn}(x_i - x_j) \operatorname{sgn}(y_i - y_j)$$

Where sgn is the sign function

$$\operatorname{sgn} x := \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Figure 2.12 shows the Kendall rank correlation heatmap for all of the features. Most data columns appear to be largely uncorrelated, with the exception of those that are measuring the same thing (e.g. CAPE and Total Return CAPE are strongly correlated). This means that our transformations succeeded in removing spurious correlations. Of note, the strongest of the non-similar correlations is a strong negative correlation between the long term interest rate and real total bond returns; this is a well known phenomenon that bonds have an inverse relationship to interest rates.[Int]

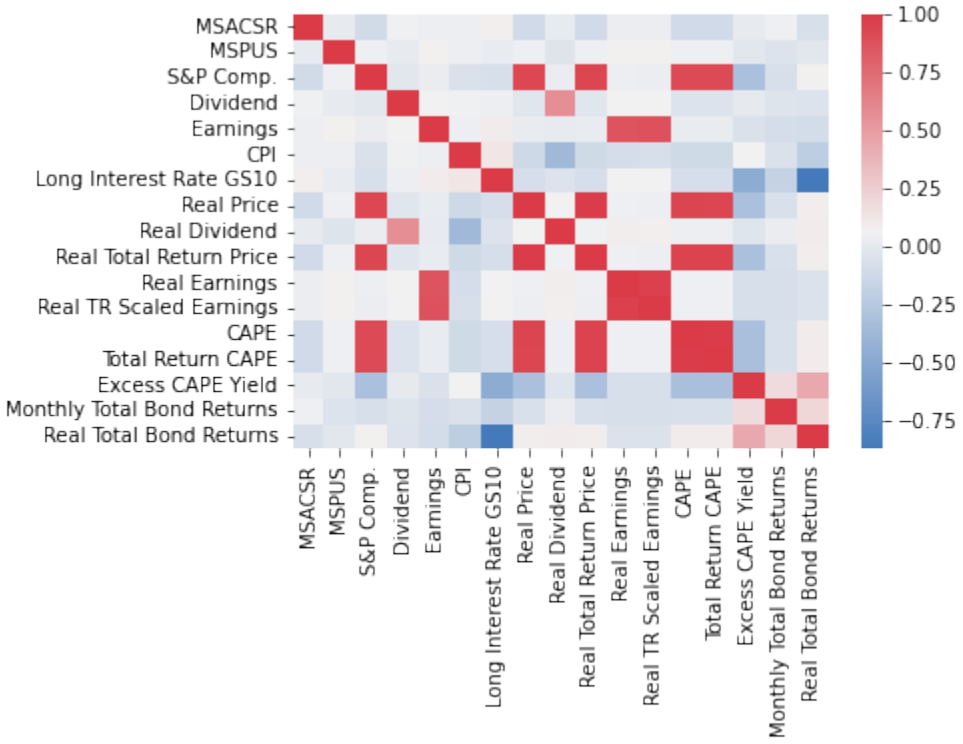

Figure 2.12: Correlation Heatmap for All Features

2.6 Data Pre-processing

Before training our model, we pre-process the data by splitting our data into training and testing sets and by scaling and normalizing our features.

2.6.1 Training & Testing Datasets

We split the dataset into separate training and testing sets to prevent the model from overfitting to the data we train on. If the model is trained and tested on the same data, it increases the risk that the model will be poor at generalizing to new data. Additionally, splitting data into training and testing sets before applying any scaling or normalization prevents leaking information between the sets, keeping them more isolated.

Figure 2.13 is a diagram of how our data is split. Specifically, we do a 95-5 train-test

split, where the first 95% of the dates in the ordered time series are put in the training set and the last 5% chronologically are the test set. We then process the training and testing sets separately before using them to train our LSTM model.

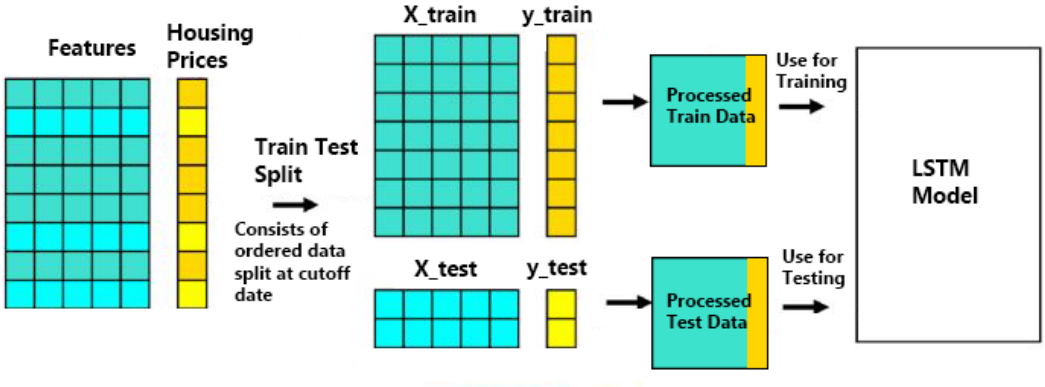

Figure 2.13: Diagram of Training and Testing Dataset Splits

2.6.2 Feature Standardization

Scaling reduces model bias when features have a variety of ranges by normalizing so the model is not dominated by features that have wider ranges. Additionally, gradient descent used by our model converges more quickly with feature scaling than without, allowing for faster training times.[IS15]

Since the ranges of features in the dataset vary, features with a wider range of values may contribute more to the model than those with a narrower range, creating bias. We use scaling to reduce the risk of this happening by standardizing the features to have a mean of zero and standard deviation of one. This is done by applying the following formula to each value x for each feature:

$$z = \frac{x - \bar{x}}{\sigma}$$

Where $\bar{x}$ is the mean of that feature vector

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

n is the number of values in the feature vector and σ is the sample standard deviation

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

2.6.3 Target Min-Max Normalization

For the target variable, the monthly percent change in Median U.S. Housing Prices, we apply min-max normalization so that values are in the range $[0, 1]$ by subtracting the lowest value from all data points and dividing that by the range of the values.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

CHAPTER 3

Models

Long Short-Term Memory (LSTM) neural networks have been found to be useful for predictions using time series data. In the following sections we describe the architecture of a typical LSTM model and explain how we apply it on our data to predict U.S. housing price trends.

Autoregressive Integrated Moving Average (ARIMA) models are widely used in time series forecasting so we will make use of one as a baseline comparison for our LSTM model.

3.1 ARIMA Model

ARIMA models consist of three components: the autoregressive (AR) component, the integrated (I) component, and the moving average (MA) component[HA21]. The AR component models the current value of the time series as a linear combination of its past values, while the MA component models the current value as a linear combination of the past forecast errors. The I component accounts for any non-stationarity in the time series, such as trends or seasonality.

Formally, the full ARIMA model for a differenced series y'_t at time-step t can be written as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Where the left hand side of the equation consists of the autoregressive portion of order p with constant drift c and parameters ϕ. The right hand side is the moving average portion of order q with parameters θ and ε as white noise.

3.1.1 ARIMA Information Criteria

To determine the orders that we should use for our ARIMA model, we need a way compare models using different order parameters p and q. Akaike's Information Criteria (AIC) is one commonly used measure for model comparison, for an ARIMA model it is defined as[HA21]:

$$\text{AIC} = -2\log(L) + 2(p + q + k + 1)$$

for likelihood of the data L and where $k = 1$ if $c \neq 0$ and $k = 0$ if $c = 0$. To avoid the potential for bias, we will use the corrected AIC for ARIMA which can be written as:

$$\text{AICc} = \text{AIC} + \frac{2(p + q + k + 1)(p + q + k + 2)}{N - p - q - k - 2}$$

Where N is the number of observations.

3.2 LSTM Model

An LSTM is a type of Recurrent Neural Network (RNN) that allows for information to accumulate for a long duration, until it has been used and can be 'forgotten.' Recurrent neural networks are a family of neural networks that process sequential data, maintaining a sort of 'memory' by having cyclic connections within the network structure.

A simple RNN architecture is shown in Figure 3.1[GBC16]. The left hand side shows the recurrent connection to the hidden layer h, and on the right is the structure unfolded across time. Let $x^{(1)}, ..., x^{(n)}$ be sequential inputs, $h^{(1)}, ..., h^{(n)}$ be the sequence of hidden

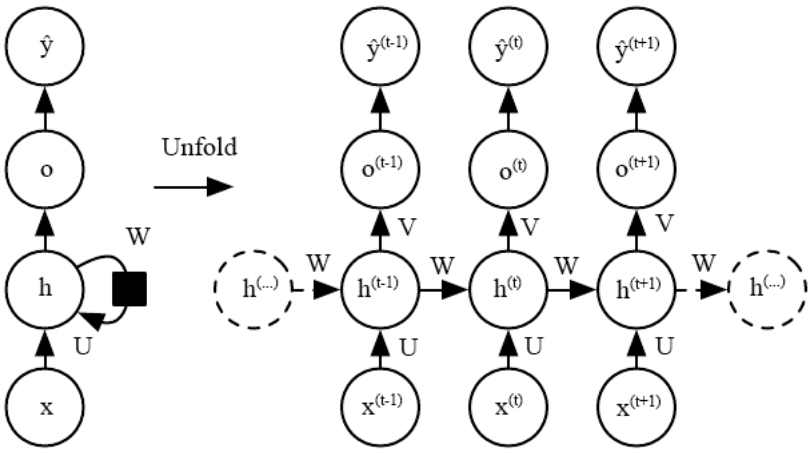

Figure 3.1: Simple Recurrent Neural Network Computational Graph

states generated at each time step, $o^{(1)}, ..., o^{(n)}$ be the outputs, and $y^{(1)}, ..., y^{(n)}$ be the model predictions. Then, given an initial state $h^{(0)}$, the hidden layer, output layer, and predictions are updated at each time step $1, , , n$ by the following equations:

$$h^{(t)} = \tanh(b + Wh^{(t-1)} + Ux^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$y^{(t)} = g(o^{(t)})$$

Where U, V, and W are the weight matrices for input-to-hidden, hidden-to-output, and hidden-to-hidden connections respectively. Additionally, b and c are the bias vectors and $g(z)$ is the softmax function defined as:

$$g(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Recurrent neural networks have difficulty with longer sequences because backpropogated error signals from further back in the sequence tend to vanish or blow up. [HS97] This is called the *vanishing gradient* problem and is improved in Long Short-Term memory networks by introducing the *memory cell* to the RNN architecture. These memory cells allow the network

to accumulate information, then forget it after it has been used.

Figure 3.2[GBC16] shows a diagram of an LSTM memory cell, where the black square indicates a delay of a single time step. These cells are connected recurrently to each other, and replace the hidden units of recurrent networks. Each cell has three gates that regulate the flow of information into and out of the cell: an input gate, a forget gate, and an output gate.

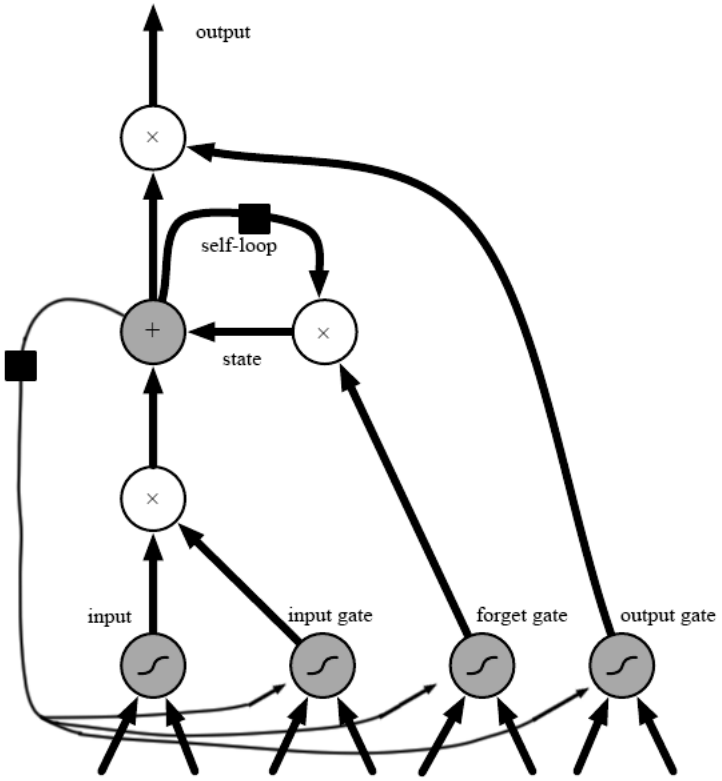

Figure 3.2: Architecture of an LSTM Memory Cell

Formally, these cells are defined by the following equations:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$
$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$
$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

Where $f_i^{(t)}$, $g_i^{(t)}$, and $q_i^{(t)}$ represent the forget, input, and output gates respectively at time step t and for cell i. The weight matrices U, V, W and bias vectors b have superscripts denoting which gate they belong to, e.g. U^f for a forget gate and b^o for an output gate. The sigmoid activation function $\sigma(z)$ is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The LSTM cell internal state $s_i^{(t)}$ at time step t for memory cell i is updated by:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

The output $h_i^{(t)}$ of the LSTM cell is calculated with:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. A forget gate controls the linear inner self-loop within the memory cell, and the output gate can shut off the output for the entire cell.

The overall architecture of the LSTM (Figure 3.3) is similar to a simple RNN, but with the hidden layer replaced with LSTM memory cells. The cross-hatched circles in the diagram

represent LSTM memory cells.

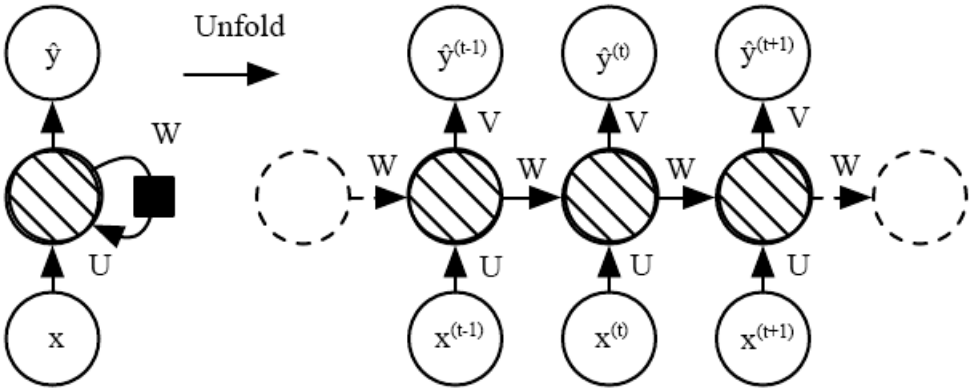

Figure 3.3: Architecture of LSTM Network

Due to this structure, LSTM networks can learn long-term dependencies within time series data such as ours, allowing for changes in the macro economic situation to accumulate over time before culminating in housing price changes.

3.2.1 LSTM Loss Function

The loss function we will be using to measure the LSTM model's error is Mean Squared Error (MSE) loss, which represents the average squared difference between the predicted values and the actual values. This is calculated by the following equation:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where $\hat{y}_i$ is the ith predicted value and y_i is the corresponding actual value.

3.2.2 Backpropogation Optimizer

After the inputs are forward propogated through the LSTM network, the loss between the actual and predicted values needs to be backpropogated back through the network to update the weight matrices. Gradient descent is one way to update these weights, we will be using a more advanced technique that builds on gradient descent, Adagrad, and RMSProp, called the Adam optimizer.

3.2.2.1 Gradient Descent

Gradient descent is an algorithm used to minimize a loss function by iteratively updating weights in the direction of steepest descent, i.e. the gradient. Suppose we have a loss function for our parameters θ defined by $L(\theta)$, then gradient descent updates these parameters iteratively by:

$$\theta_{t+1} = \theta_t - \eta L'(\theta)$$

Where η is the learning rate and $L'(\theta)$ is the gradient of the loss function at θ. After running many iterations of this algorithm, the loss will converge to a local minimum. This is simple training method but can be slow to converge.

3.2.2.2 AdaGrad

One problem with regular gradient descent is that parameters all use the same learning rate, regardless of the size of their search space. AdaGrad, short for Adaptive Gradient, accelerates the optimization process by tailoring the learning rate to each individual parameter.[DHS11] This way, dimensions with consistently steep gradients can have larger steps and those with less steep gradients can have smaller steps. At each time step t, AdaGrad uses a different learning rate for each parameter with gradient g_t, and parameters are updated by:

$$\theta_{t+1} = \theta_t - \eta \frac{g_t}{\sqrt{G_t + \epsilon}}$$

Where ϵ is a small value to prevent division by zero and G_t is a diagonal matrix where each entry is the cumulative sum of the squares of the gradients

$$G_t = G_{t-1} + g_t^2$$

3.2.2.3 RMSProp

Root Mean Square Propogation (RMSProp) is an extension of AdaGrad that uses a momentum value to put more weight on more recent time steps when updating individual gradients.[Rud47] Parameters are updated by the same algorithm as AdaGrad, but the diagonal gradient matrix G_t is updated by:

$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2$$

Where γ is the momentum constant and is usually set to 0.9.

3.2.2.4 Adam

The Adam optimizer builds on both AdaGrad and RMSProp by updating the learning rates for each iteration using estimations of both the first and second moments (mean and uncentered variance) of the gradients.[KB] The algorithm does this by calculating an exponentially decaying moving average of the gradient and squared gradient as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 m_{t-1} + (1 - \beta_2) g_t^2$$

Where m_t and v_t are the estimates of the first and second moments of the gradients respectively. To avoid a bias towards zero, these moment estimates are bias-corrected by:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

And finally parameters are updated by:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

The Adam optimization method is useful because it improves performance on problems with sparse gradients and noisy or non-stationary problems which can often be the case with financial data.

CHAPTER 4

Model Training and Results

4.1 ARIMA Model Selection

To select an ARIMA model for forecasting we use auto_arima grid search from the pmdarima Python library. The auto_arima function uses a stepwise search procedure to iteratively fit the ARIMA models with different combinations of p, d, and q. At each step, it fits a new model using the parameters that result in the lowest corrected AIC.

We define the search space by starting with p and q equal to 0, up to a maximum of 15. We set d equal to 0 because the data has already been differenced. Using this search space we find the optimal parameters to be $p = 10$ and $q = 0$ with a corrected AIC of -6467.47 as shown in Figure 4.1. This ARIMA(10,0,0) model is the one we use in the results section.

4.2 LSTM Training

The following pseudocode (Algorithm 1) shows what is done in our training loop for the LSTM model. This procedure is repeated for each training epoch.

Our actual code is done in Python, with the Pytorch library being used for the LSTM neural network. Because the dataset is relatively small, training could be done on a laptop in a few minutes with the Adam optimizer.

Figure 4.2 plots the training and testing losses for each epoch in the training loop. The losses quickly decrease in the first 50 epochs, and then slowly trend downward for the re-

```
Performing stepwise search to minimize aicc
 ARIMA(0,0,0)(0,0,0)[0]             : AICC=-4335.457, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AICC=-5079.127, Time=0.09 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AICC=-5087.466, Time=0.10 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AICC=-5585.135, Time=0.22 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AICC=-5973.655, Time=0.36 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AICC=-5776.541, Time=0.10 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AICC=-6137.875, Time=0.53 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AICC=-6185.921, Time=0.09 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AICC=-6188.893, Time=0.20 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AICC=-6275.477, Time=0.23 sec
 ARIMA(6,0,0)(0,0,0)[0]             : AICC=-6401.526, Time=0.28 sec
 ARIMA(7,0,0)(0,0,0)[0]             : AICC=-6429.407, Time=0.12 sec
 ARIMA(8,0,0)(0,0,0)[0]             : AICC=-6433.079, Time=0.24 sec
 ARIMA(9,0,0)(0,0,0)[0]             : AICC=-6436.036, Time=0.80 sec
 ARIMA(10,0,0)(0,0,0)[0]             : AICC=-6453.477, Time=0.39 sec
 ARIMA(11,0,0)(0,0,0)[0]             : AICC=-6455.072, Time=0.47 sec
 ARIMA(12,0,0)(0,0,0)[0]             : AICC=-6453.008, Time=0.46 sec
 ARIMA(11,0,1)(0,0,0)[0]             : AICC=-5909.552, Time=1.66 sec
 ARIMA(10,0,1)(0,0,0)[0]             : AICC=-5923.850, Time=1.49 sec
 ARIMA(12,0,1)(0,0,0)[0]             : AICC=-5985.627, Time=1.92 sec
 ARIMA(11,0,0)(0,0,0)[0] intercept   : AICC=-6467.299, Time=0.96 sec
 ARIMA(10,0,0)(0,0,0)[0] intercept   : AICC=-6467.467, Time=0.79 sec
 ARIMA(9,0,0)(0,0,0)[0] intercept   : AICC=-6455.484, Time=0.69 sec
 ARIMA(10,0,1)(0,0,0)[0] intercept   : AICC=-5958.270, Time=2.86 sec
 ARIMA(9,0,1)(0,0,0)[0] intercept   : AICC=-6098.268, Time=2.56 sec
 ARIMA(11,0,1)(0,0,0)[0] intercept   : AICC=-5979.631, Time=2.99 sec

Best model:  ARIMA(10,0,0)(0,0,0)[0] intercept
Total fit time: 20.687 seconds
```

Figure 4.1: ARIMA Grid Search Results

Algorithm 1 LSTM Training Loop

1: *loop*:
2: **for** each epoch **do**
3: LSTM Forward Pass
4: Calculate Gradient
5: LSTM Backpropogate Training Loss, Updating Weights
6: Predict on Test Data
7: Store Training and Test Loss
8: **end for**

mainder of the training duration with the test loss being slightly higher than training loss for the vast majority of iterations. We stop training after 1000 epochs to reduce the chances of overfitting to our training data.

Our model uses a single LSTM layer, with a hidden layer size of 10. We use an Adam optimizer for backpropogation as mentioned above, with an initial learning rate of 0.001.

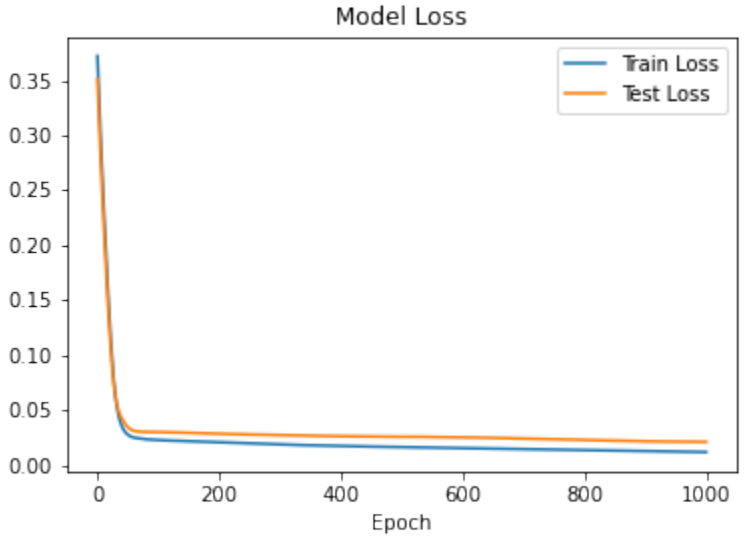

Figure 4.2: Training Loss by Iteration

4.3 Results

Our goal for this experiment was to use the model to predict longer term trends of U.S. housing prices based on economic data. To see how our trained LSTM model performs, we use one final forward pass of the test data, only feeding it the 24 months before the start of the 12 month time period we wish to predict.

Figure 4.3 shows our predicted values versus the actual median housing prices. The 12 months predicted begin after the dashed red line. The predictions look to follow the trends fairly well, predicting a jump up but initially faster than the actual values, then more slowly, and missing the slight downturn at the end.

In figure 4.4 we plot the results of an ARIMA model forecast for the same 12 month period. This model also does well at predicting the direction of the trends and the 95% prediction interval captures most of the actual values with the exception of expecting higher prices immediately at the beginning of the forecast.

Finally, Table 4.1 has error root-mean-squared and standard error values for comparison

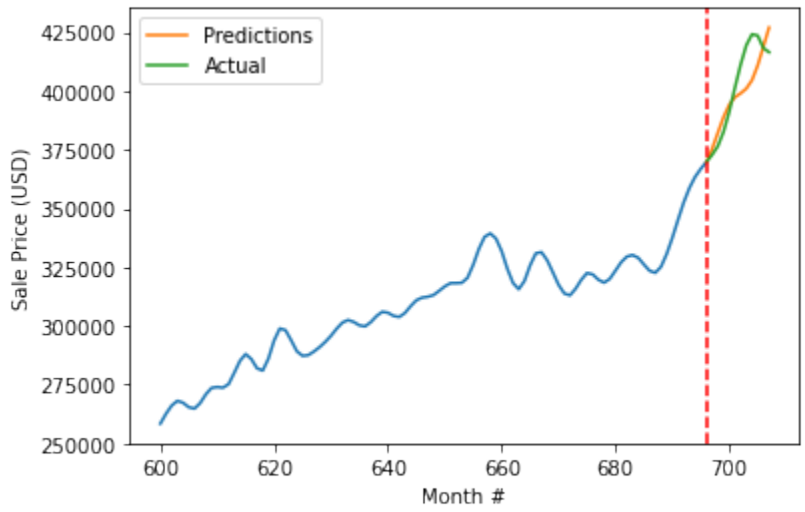

Figure 4.3: LSTM Model Forecast Actual vs. Predicted

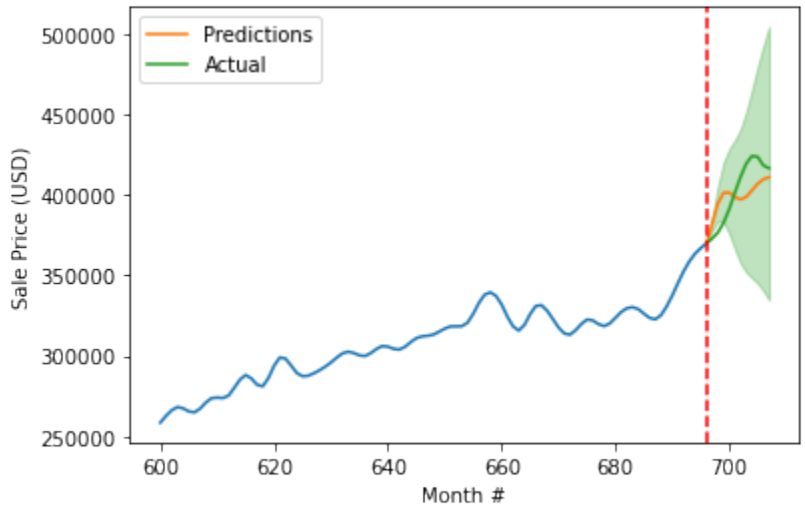

Figure 4.4: ARIMA Model Forecast with 95% Forecast Interval

between the two models. The LSTM model shows a roughly 20% improvement for both RMSE and standard error, suggesting an improved performance over this time interval.

Table 4.1: Model Error Comparison

Model	**RMSE**	**Standard Error**
ARIMA	12,927.55	14,161.42
LSTM	10,204.81	11,178.81

CHAPTER 5

Conclusion & Further Discussion

We presented in this paper a method for predicting potential trends in U.S. housing prices using macro economic data and an LSTM neural network. We outlined a procedure for wrangling the data into a manageable form using cubic spline interpolation and logarithmic differencing. We found that an LSTM was able to produce reasonable predictions of trends for housing prices in the past year.

The model was somewhat limited in scope given the relatively small dataset and could be improved with the plethora of financial and economic data available related to housing. For example, incorporating employment, income data, or consumer sentiment would expand on the purchasing power of buyers; data regarding property taxes or tax incentives could provide information on home-buying friction. Additionally, we could explore other promising models such as Latent State Space or Transformer models to compute predictions.

The results offered a promising display of the potential of using LSTM models for financial data, however we must remember that prediction is difficult and the housing market is the sum of thousands more variables than were used in this study. To take the words of statistician George Box: "All models are wrong, but some are useful." As with any financial model, extrapolating too much could get one into trouble, but perhaps it could be used in conjunction with other tools and domain knowledge to aid prospective home buyers in their search.

REFERENCES

[CA17] Tim J. Cole and Douglas G. Altman. "Statistics Notes: Percentage differences, symmetry, and natural logarithms." *British Medical Journal Publishing Group*, **doi:10.1136/bmj.j3683**:1–2, August 2017.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *Journal of Machine Learning Research*, **12**(61):2121–2159, 2011.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[HA21] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice (3rd ed)*. OTexts, 2021.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation*, **9**(8):1735–1780, 1997.

[Int] "Interest Rate Risk." *U.S. Securities and Exchange Commission*, 5 June 2022, https://www.sec.gov/files/ib_interestraterisk.pdf.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *CoRR*, **abs/1502.03167**, 2015.

[KB] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *ICLR 2015, San Diego,, 7-9 May.*

[Ken38] M. G. Kendall. "A New Measure of Rank Correlation." *Biometrika*, **30**(1-2):81–93, 1938.

[Rud47] Sebastian Ruder. "An Overview of Gradient Descent Optimization Algorithms." *arXiv Pre-print*, 2016, doi: arXiv:1609.04747.

[Shi] Robert Shiller. "Online Data - Robert Shiller." 1 April 2022, http://www.econ.yale.edu/ shiller/data.htm.

[UU63a] U.S. Census Bureau and U.S. Department of Housing and Urban Development. "Median Sales Price of Houses Sold for the United States.", January 1963. Publisher: FRED, Federal Reserve Bank of St. Louis.

[Last updated: 2022-05-26 09:01:01-05.]

[UU63b] U.S. Census Bureau and U.S. Department of Housing and Urban Development. "Monthly Supply of New Houses in the United States.", January 1963. Publisher: FRED, Federal Reserve Bank of St. Louis.

[Last updated: 2022-05-26 09:01:03-05.]

[Wol] George Wolberg. "Cubic Spline Interpolation: A Review." Department of Computer Science, Columbia University. doi:10.7916/D82Z1DMQ, 1988.

[Yul26] G. Udny Yule. "Why Do We Sometimes Get Nonsense-Correlations Between Time-Series?–A Study in Sampling and the Nature of Time-Series." *Journal of the Royal Statistical Society*, **89**(1):1–63, 1926.