TIME SERIES FORECASTING

I. INTRODUCTION

This report will discuss about various time series models used for the price prediction of Raspberry and will compare the results of the same. Systemic patterns or trends throughout time can be better comprehended by time series analysis. Time series analysis has two basic objectives: figuring out what phenomenon is being represented by the sequence of observations and forecasting (predicting future values of the time series variable). A forecasting method is the procedure for computing forecast from present and past values. Forecasting methods may be broadly classified into 3 types: a. *Judgemental forecast*, b. *Univariate methods* (which is analysed in this report) and c. *Multivariate methods*.[2] The basic time series models for forecasting includes Autoregression (AR), Moving Average (MA), Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA), and Seasonal Autoregressive Integrated Moving-Average (SARIMA). Deep learning methods, such as Multilayer perceptron, Convolutional Neural Networks and Long Short-Term Memory Networks can be used to automatically learn the temporal dependence structures for challenging time series forecasting problems. Neural networks may not be the best solution for all time series forecasting problems, but for those problems where classical methods fail, and machine learning methods require elaborate feature engineering; deep learning methods can be used with great success. In this report we will implement SARIMA, MLP, CNN and LSTM based time series model on Raspberry dataset to analyse their performance.

The rest of this report is organized as follows: Section II describes the dataset used and the pre-processing steps performed on it to make it desirable for the analysis. Section III discusses about the working/concepts of various models used for the price prediction. In Section IV, we present the details about the discussed model implementation and in section V evaluation and the experimental results of the models are briefed. Finally, Section VI concludes the analysis by comparing the performance of all the models.

II. DATA SOURCE

In this analysis, univariate model is developed using the weekly price data of raspberry at Santa Maria, California extracted from November 2010 to February 2022. The obtained data cannot be directly used for the analysis. It must undergo pre-processing steps such as Data Cleaning (Identifying and correcting mistakes or errors in the data), Feature Selection (Identifying those input variables that are most relevant to the task), Data Transforms (Changing the scale or distribution of variables), Feature Engineering (Deriving new variables from available data) and Dimensionality Reduction (Creating compact projections of the data). Below data cleaning operations are performed on the Raspberry dataset to obtain the final cleaned dataset,

- Identification of normal data and outliers.
- Removing the columns that have the same value or no variance.
- Removing duplicate rows of data.
- Marking empty values as missing and inputting them by using the average value of the data.

Once the data set is prepared suitable time series models are zeroed in for the analysis.

III. TIME SERIES MODELS

1. SARIMA Model

ARIMA model is one of the widely used time series models but the problem with it is that it doesn't support seasonal data, time series with a repeating cycle. ARIMA expects data that is either not seasonal or has the seasonal component removed. So, for time series with seasonal patterns, SARIMA model is preferred.

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, that explicitly supports univariate time series data with a seasonal component. It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality. Configuring a SARIMA requires selecting hyperparameters for both the trend and seasonal elements of the series.

Trend Elements

There are three trend elements that require configuration. They are the same as the ARIMA model; specifically:

- p: Trend autoregression order.
- d: Trend difference order.
- q: Trend moving average order.

Seasonal Elements

There are four seasonal elements that are not part of ARIMA that must be configured; they are:

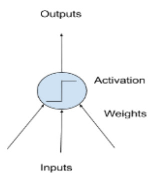
- P: Seasonal autoregressive order.
- D: Seasonal difference order.
- Q: Seasonal moving average order.
- m: The number of time steps for a single seasonal period.

Together, the notation for an SARIMA model is specified as: **SARIMA(p,d,q)(P,D,Q)m**

2. Deep Learning Models

2.1 MLP Model [4]

A perceptron is a single neuron model that was a precursor to larger neural networks. The power of neural networks comes from their ability to learn the representation in your training data and how best to relate it to the output variable you want to predict. The predictive capability of neural networks based on the multi-layered structure of the networks. The data structure can pick out (learn to represent) features at different scales or resolutions and combine them into higher-order features. The building blocks for neural networks is artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.



Each neuron has a bias which can be thought of as an input that always has the value 1.0, and it too must be weighted. The weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron. Non-linear activation functions are widely used as it allows the network to combine the inputs in more complex ways and, in turn, provide a richer capability in the functions they can model. Non-linear functions like the logistic, also called the sigmoid function, were used to output a value between 0 and 1 with an s-shaped distribution. The hyperbolic tangent function, also called tanh, outputs the same distribution over the range -1 to +1. Currently, rectifier activation function is widely used as it is fast and tend to converge more quickly.

Neurons are arranged into networks of neurons. A row of neurons is called a layer, and one network can have multiple layers. The bottom layer that takes input from your dataset is called the visible layer / Input layer because it is the exposed part of the network. Layers after the input layer are called hidden layers because they are not directly exposed to the input. The final hidden layer is called the output layer, and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

Once configured, the neural network needs to be trained on the dataset and the algorithm used is stochastic gradient descent. One row of data is exposed to the network at a time as input. The network processes the input upward, activating neurons as it goes to finally produce an output value (Forward pass). The output of the network is compared to the expected output, and an error is calculated. This error is then propagated back through the network, one layer at a time, and the weights are updated according to the amount they contributed to the error (Backpropagation algorithm). The process is repeated for all the examples in your training data. One round of updating the network for the entire training dataset is called an epoch. A network may be trained for tens, hundreds, or many thousands of epochs. The weights in the network can be updated from the errors calculated for each training example (online learning). Alternatively, the errors can be saved across all the training examples, and the network can be updated at the end (Batch learning). Once a neural network has been trained, it can be used to make predictions.

2.2 CNN Model

Feedforward neural networks can learn a single feature representation of the image but in the case of complex images, ANN will fail to give better predictions, this is because it cannot learn pixel dependencies present in the images.

A Convolutional neural network (CNN, or ConvNet) is another type of neural network that can be used to enable machines to visualize things. These classes of neural networks can input a multi-channel image and work on it easily with minimal preprocessing required. CNN takes an image as an input, processes it, and classifies it under certain categories. CNN can learn multiple layers of feature representations of an image by applying filters, or transformations. In CNN, the number of parameters for the network to learn is significantly lower than the multilayer neural networks since the number of units in the network decreases, therefore reducing the chance of overfitting. Also, CNN considers the context information in the small neighborhood and due to this feature, these are very important to achieve a better prediction in data like images.

The different layers involved in the architecture of CNN are as follows:

1. Input Layer: The input layer in CNN should contain image data. Image data is represented by a three-dimensional matrix. We must reshape the image into a single column.

2. Convolutional Layer: To perform the convolution operation, this layer is used which creates several smaller picture windows to go over the data. Convolution is a linear operation of a smaller filter to a larger input that results in an output feature map. Convolutional units go by many names: Filters, kernels, and conv units. Unlike a fully connected unit, whose output is a single scalar, the output of a convolutional unit is a tensor (i.e., a multidimensional array), usually called a feature map.

The hyperparameters of convolution layer include:

Filter size: The most common sizes are: 3X3 and 5X5.

Number of Filters in a layer: This controls the depth (the number of feature maps) of the output of the layer.

Stride: The step size used when sliding the filters. As the stride increases, the output shrinks. This is usually set to 1.

Padding: This refers to how the CNN should handle the edge cases (when part of the filter extends outside the input). One way to handle this is to zero-pad the input, so that, if the stride is 1, the output spatial size is the same as the input. This is usually referred to as "same padding" in popular libraries. The other option is to discard the output of such cases. This is usually called "valid padding".

3. ReLU Layer: This layer introduces the non-linearity to the network and converts all the negative pixels to zero. The final output is a rectified feature map.

4. Pooling Layer: Pooling is a down-sampling operation that reduces the dimensionality of the feature map so that it speeds up the computation of the network. The hyperparameters for a pooling layer are Filter size, Stride, Max or average pooling.

Max pooling: Once we obtain the feature map of the input, we will apply a filter of determined shapes across the feature map to get the maximum value from that portion of the feature map. It is also known as subsampling because from the entire portion of the feature map covered by filter or kernel, we are sampling one single maximum value.

Average pooling: Computes the average value of the feature map covered by kernel or filter and takes the floor value of the result.

After a series of convolution and pooling operations on the feature representation of the image, we then flatten the output of the final pooling layers into a single long continuous linear array or a vector. The process of converting all the resultant 2-d arrays into a vector is called Flattening. Flatten output is fed as input to the fully connected neural network having varying numbers of hidden layers to learn the non-linear complexities present with the feature representation.

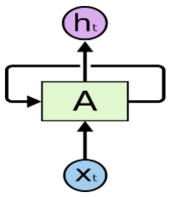
5. Fully Connected Layer: This layer identifies and classifies the objects in the image. The aim of the Fully connected layer is to use the high-level feature of the input image produced by convolutional and pooling layers for classifying the input image into various classes based on the training dataset. It works like an ANN, assigning random weights to each synapse, the input layer is weight-adjusted and put into an activation function. The output of this is then compared to the true values and the error generated is back-propagated, i.e., the weights are re-calculated and repeat all the processes. This is done until the error or cost function is minimized.

6. SoftMax / Logistic Layer: The SoftMax or Logistic layer is the last layer of CNN. It resides at the end of the FC layer. Logistic is used for binary classification problem statement and SoftMax is for multiclassification problem statement.

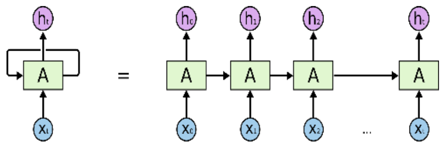
7. Output Layer: This layer contains the label in the form of a one-hot encoded vector.

2.3 LSTM Model [3]

RNN are networks with loops allowing information to persist. In the below diagram, a chunk of neural network, (A), looks at some input (xt) and outputs a value (ht). A loop allows information to be passed from one step of the network to the next.



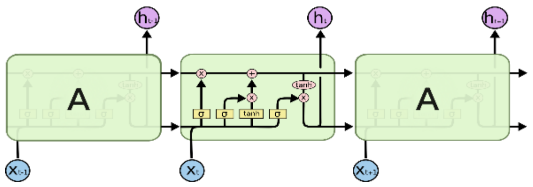
A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task. But can they? It depends. In certain cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information. But in cases where we need more context, i.e., the gap between the relevant information and the point where it is needed to become very large RNN’s unable to learn to connect the information. So, LSTM became more persistent in those cases.

One of the most advanced models to forecast time series is the Long Short-Term Memory (LSTM) Neural Network. “*The LSTM cell adds long-term memory in an even more performant way because it allows even more parameters to be learned. This makes it the most powerful [Recurrent Neural Network] to do forecasting, especially when you have a longer-term trend in your data.*” [1]

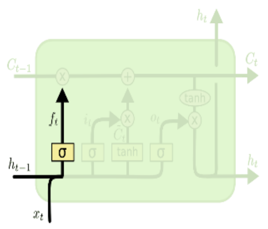
Long Short Term Memory networks (LSTM) are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The key to LSTMs is the cell state. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

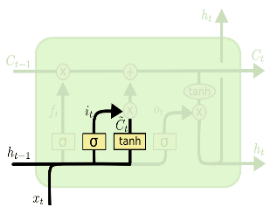
An LSTM has three gates, to protect and control the cell state.

The first step in LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at (ht-1) and (xt), and outputs a number between (0) and (1) for each number in the cell state (Ct-1). A (1) represents "completely keep this" while a (0) represents "completely get rid of this."



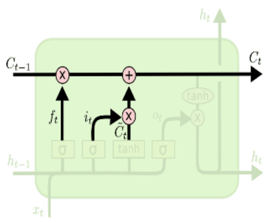
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information is going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values to update. Next, a tanh layer creates a vector of new candidate values, (~Ct), that could be added to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, (Ct-1), into the new cell state (Ct). The old state is multiplied by (ft), forgetting the things decided to forget earlier. Then (it*~Ct) is added to it. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, the output will be decided. This output will be based on the cell state but will be a filtered version. First, sigmoid layer runs which decides what parts of the cell state is going to output. Then, the cell state is put through (tanh) (to push the values to be between (-1) and (1)) and multiply it by the output of the sigmoid gate, so that the decided parts will only be outputted.

IV. MODEL IMPLEMENTATION

1. SARIMA Model

Before trying to model and forecasting given time series, it is desirable to have a preliminary look at the data to get a feel for them and to get some idea of their main properties. The time plot is the most important tool used to get insights from the data and it showed that the data has seasonal properties. So, SARIMA model is preferred for the analysis.

Price forecasting is performed by building the model, fitting the defined model, and making the prediction with the fit model. Python is used as the programming language for this analysis. Based on the ACF and PACF plots order of the MA and AR model is decided upon and the seasonality analysis of the data provided the seasonality elements order for the SARIMA model. The data set is divided into training (# data 424), validation (# data 74), and test (# data 91) subsets and the designed model is trained using the training subset. Once the model is trained forecasting is performed with the validation and test subsets.

A rolling forecast scenario is used, also called walk-forward model validation. Each time step of the test dataset will be walked one at a time. A model will be used to make a forecast for the time step (one week), then the actual expected value from the test set will be taken and made available to the model for the forecast on the next time step. The designed SARIMA model forecasted the raspberry price one week in advance using the past data. The performance of the forecast is reported using the Mean absolute Percentage Error, Root Mean Square Error and R2 scores.

2. Deep Learning Models

2.1 MLP

MLPs can be used for time series forecasting by taking multiple observations at prior time steps, called lag observations, and using them as input features and predicting one or more-time steps from those observations. Walk-forward validation approach is used, where the model makes a forecast for each observation in the Val/test dataset one at a time. After each forecast is made for a time step in the Val/test dataset, the true observation for the forecast is added to the Val/test dataset and made available to the model [4]. That true observation for the time step can then be used apart of the input for making the prediction on the next time step. First, the dataset is split into training (# data 424), validation (# data 74), and test (# data 91) subsets and the designed model is trained using the training subset.

A model will be fit once on the training dataset for a given configuration. It takes the training dataset and the model configuration and returns the fit model ready for making predictions. A single one-step prediction is made by taking the fit model, the history, and the model configuration. The prediction is added to a list of predictions and the true observation from the Val/test set is added to a list of observations that was seeded with all observations from the training dataset. This list is built up during each step in the walk-forward validation, allowing the model to make a one-step prediction using the most recent history. All the predictions can then be compared to the true values in the test set and an error measure (MSE, RMSE and R squared) calculated.

2.2 CNN

Convolutional Neural Networks, or CNNs, are a type of neural network developed for two-dimensional image data, although they can be used for one-dimensional data such as sequences of text and time series. When operating on one-dimensional data, the CNN reads across a sequence of lag observations and learns to extract features that are relevant for making a prediction. For this analysis CNN model is defined with two convolutional layers for extracting features from the input sequences. Each will have a configurable number of filters and kernel size and will use the rectified linear activation function. The number of filters determines the number of parallel fields on which the weighted inputs are read and projected. The kernel size defines the number of time steps read within each snapshot as the network reads along the input sequence. A max pooling layer is used after convolutional layers to distil the weighted input features into those that are most salient, reducing the input size by 1/4. The pooled inputs are attended to one long vector before being interpreted and used to make a one-step prediction. [4]

Like all the models, the dataset is split into training (# data 424), validation (# data 74), and test (# data 91) subsets. The fit function of the CNN model is fitted on the training dataset and the prediction on the fit model is very much like the prediction performed by the MLP model. Once the prediction is made the performance of the model is analysed and compared against the other models.

2.3 LSTM

Like SARIMA model, the data set is divided into training (# data 424), validation (# data 74), and test (# data 91) subsets. Forecasting is performed by building the model, fitting the defined model, and training it using training dataset. Once the model is trained, validation and test subsets are used to predict the price of Raspberry. Before we can fit an LSTM model to the dataset, we must transform the data as below

- Transform the time series into a supervised learning problem

  The LSTM model assumes that your data is divided into input (X) and output (y) components. For a time series problem, we can achieve this by using the observation from the last time step (t-1) as the input and the observation at the current time step (t) as the output.

- Transform the time series data so that it is stationary

  Data is said to be stationary if its mean and variance is constant and doesn’t have seasonality and trend in it. This is achieved by using transformations and most used transformation is differencing. Stationarity of the data can be verified by several tests; most widely used test is Augmented Dickie Fuller test (ADF)

- Transform the observations to have a specific scale

LSTMs expect data to be within the scale of the activation function used by the network. The default activation function for LSTMs is the hyperbolic tangent (tanh), which outputs values between -1 and 1. The scaling coefficients (min and max) values are calculated on the training dataset and applied to scale the VAL/test dataset and any forecasts. MinMaxScaler is used to transform the dataset to the range [-1, 1].

Finally, the values are returned to the original scale by inverting the operation and undoing the transformations so that the results can be interpreted, and a comparable Mean absolute Percentage Error, Root Mean Square Error and R2 score can be calculated. Like SARIMA model, rolling forecast or walk-forward method is used in LSTM to predict one time step by walking over each time step one at a time. However, the model is fitted using the entire training data set and the prediction length can be adjusted by updating the batch size.

V. RESULTS

1. Raspberry dataset price prediction - SARIMA model:

First, we analyse how the price of Raspberry have changed during the extracted time. Fig 1 shows the price vs time plot of Raspberry

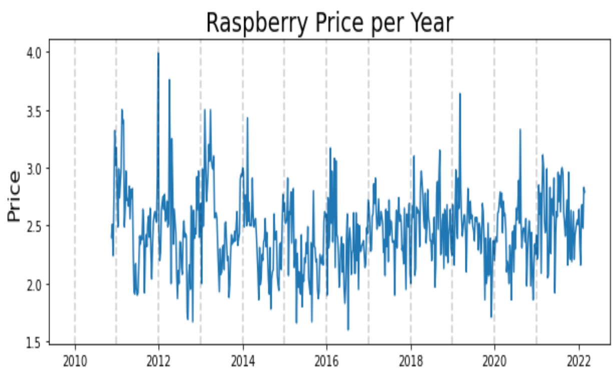


Fig 1: Price Vs Time plot of Raspberry

This is a typical time series problem; hence we apply seasonal ARIMA to forecast the upcoming prices. Seasonal decompose function is used to find the trend, seasonality, and residuals of the dataset. Fig 2 shows the plot of the same.

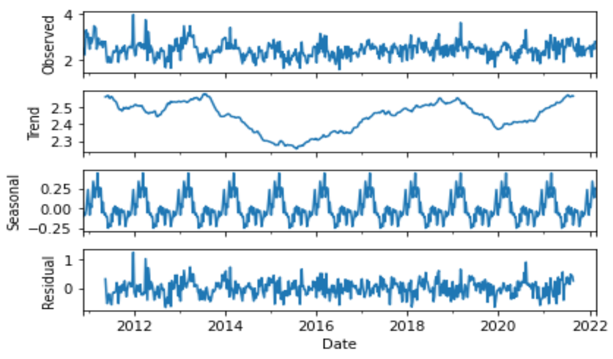
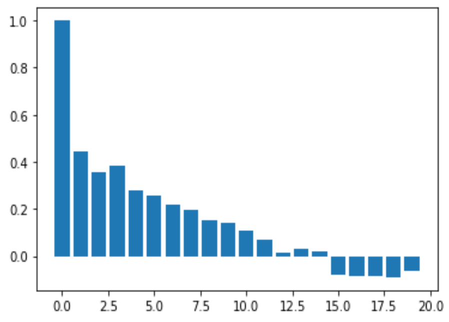


Fig 2: Data set Trends

ACF and PACF plots in Fig 3 provided the order for MA and AR components of the SARIMA model (4,0,0) whereas the seasonality of the data helped us to round in on the seasonal components of order (0,1,0,52). Hence SARIMA (4,0,0) (0,1,0,52) is used for the analysis.



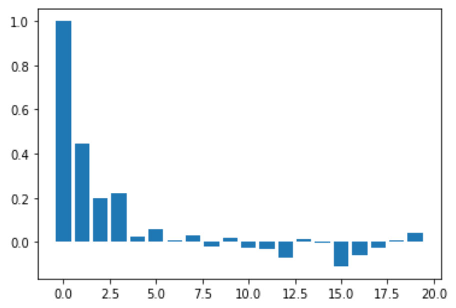


Fig 3: ACF and PACF plots

Fig 4 shows the comparison of the data and predictions made by the model once its trained and fed into the test dataset.

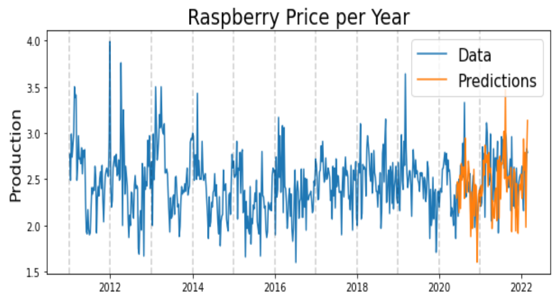


Fig 4: Actual VS Predicted price plot – SARIMA model

Performance of the model is then calculated which shows 10% Mean absolute percent error, 34 cents price variation from the actual and predicted value as per RMSE and 0.981 R2 value.

2. Raspberry dataset price prediction - Deep learning Models

2.1 MLP

MLP model is designed with 4 neurons connected sequentially with one dense layer with ReLU activation function. Model is trained for 1500 epochs with 424 training data like SARIMA model and forecast is done on the VAL (# 74 data) and test (# 91 data) datasets. Model is complied using MSE loss and adam optimizer. The computational cost of MLP model is higher than SARIMA model but lesser than that of CNN and LSTM models. Performance statistics of the model shows 7.7% MAE, 27 cents price variation from actual and predicted values per RMSE and 0.985 R2 value. Fig 5 shows the actual vs predicted price of the Raspberry.

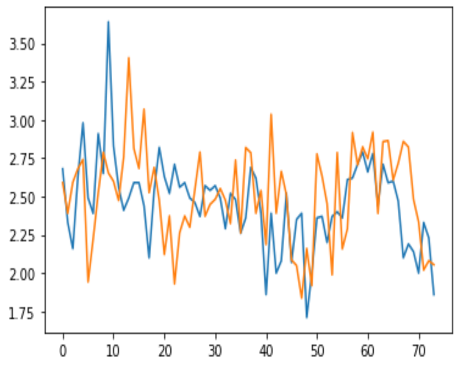


Fig 5: Actual VS Predicted price plot – MLP model

2.2 CNN

CNN model is designed with 2 Convolution Layer consisting of 256 Filters, 1 kernel and ReLU activation function. A max pooling layer with is pool size 2 which is then flattened by the flatten layer and compiled using MSE loss and adam optimizer. Model is trained for 1500 epochs with 424 training data just like other models and forecast is done on the VAL (# 74 data) and test (# 91 data) datasets. RMSE shows that there is 36 cents price variation, 12% MAE and 0.88 R2 value. CNN performed the least in comparison to all the other models for the Raspberry dataset. Below line graph (Fig 6) depicts the actual vs predicted price of the Raspberry obtained using CNN model.

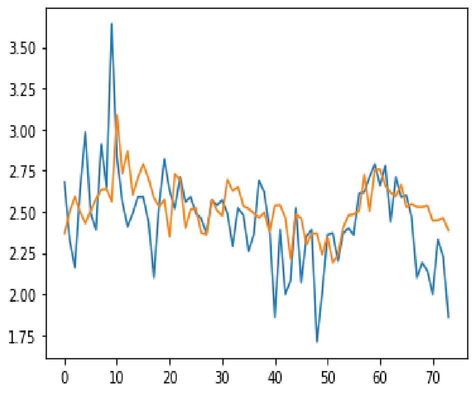


Fig 6: Actual VS Predicted price plot – CNN model

2.3 LSTM

LSTM model is designed with 4 neurons connected sequentially with one dense layer with ReLU activation function. Model is trained for 1500 epochs with 424 training data like SARIMA model and forecast is done on the VAL (# 74 data) and test (# 91 data) datasets. Fig 7 shows the actual vs predicted price of the Raspberry.

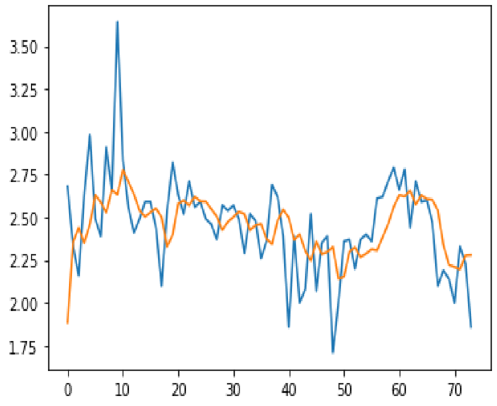


Fig 7: Actual VS Predicted price plot – LSTM model

RMSE shows that there is only 26 cents price variation from the actual and predicted value of LSTM model and 0.986 R2 value.

VI. CONCLUSION

The report analysed the Time series forecasting on the 12-year Raspberry dataset extracted from November 2010 to February 2022 of SARIMA and multiple deep learning models like MLP, CNN and LSTM. As per the results and from the below performance statistics, it is evident that the LSTM model performed better in comparison to all the other analysed models for the provided dataset, whereas CNN model performed the least. In terms of the computational cost, SARIMA model is the best amongst all the models.

Model	RMSE (cents)	MSE (%)	R2
SARIMA	34	10	0.981
MLP	27	7.7	0.985
CNN	36	12	0.88
LSTM	26	7.0	0.986

Table 1: Performance statistics of the models

References:

[1] Korstanje,Joos "Advanced Forecasting with Python With State-of-the-Art-Models Including LSTMs, Facebook's Prophet, and Amazon's DeepAR"

[2] Chatfield, C. (2001). Time-series forecasting. Chapman & Hall/CRC.

[3] Olah, C. (2015, August 27). Understanding LSTM Networks -- colah's blog.

[4] Brownlee, Jason. (2018 v1.4). Deep Learning for Time Series Forecasting