**FORECASTING PASSENGER COUNT FOR A TRANSPORTATION MEDIUM USING ARIMA AND**

**LSTM NEURAL NETWORK.**

*Name: Sagar Sharma*

*UTA ID: 1001626958*

**INTRODUCTION/PROBLEM DESCRIPTION**

This problem is taken from an online challenge available on Analytics Vidhya Website. Unicorn Investors wants to make an investment in a new form of transportation - Jet Rail. Jet Rail uses Jet propulsion technology to run rails and move people at a high speed! The investment would only make sense, if they can get more than 1 Million monthly users with in next 18 months. In order to help Unicorn Ventures in their decision, we need to forecast the passenger count on Jet Rail for the next 7 months. We are provided with traffic data of Jet Rail since inception in the train.csv file. URL: <https://datahack.analyticsvidhya.com/contest/practice-problem-time-series-2/>

**DATA DESCRIPTION**

Data Set includes test and training data in separate files. The training data file has 18,289 instances while the test file has 5,113 instances. The input variables/predictor variables are Date (dd/mm/yyyy) and Time (hh:mm) with respective formats and the output/response/target variable is the count of passengers travelled. This is a univariate time series forecast problem. We have ID, Datetime and corresponding count of passengers in the train file. For test file we have ID and Datetime only, so we must predict the Count for test file. ID is the unique number given to each observation point. Datetime is the time of each observation. Count is the passenger count corresponding to each Datetime. ID and Count are in integer format while the Datetime is in object format for the train file. ID is in integer and Datetime is in object format for test file.

**PREPROCESSING**

***Hypothesis Generation***

Hypothesis Generation is the process of listing out all the possible factors that can affect the outcome. Below are some of the hypotheses that we will be validating, which I think can affect the passenger count (dependent variable for this time series problem) on the Jet Rail:

1. There will be an increase in the traffic as the years pass by.

* **Explanation** - Population has a general upward trend with time, so I can expect more people to travel by Jet Rail. Also, generally companies expand their businesses over time leading to more customers travelling through Jet Rail.

1. The traffic will be high from May to October.

* **Explanation** – This is a naïve assumption just to check the seasonality in the time series data.

1. Traffic on weekdays will be more as compared to weekends/holidays.

* **Explanation** - People will go to office on weekdays and hence the traffic will be more.

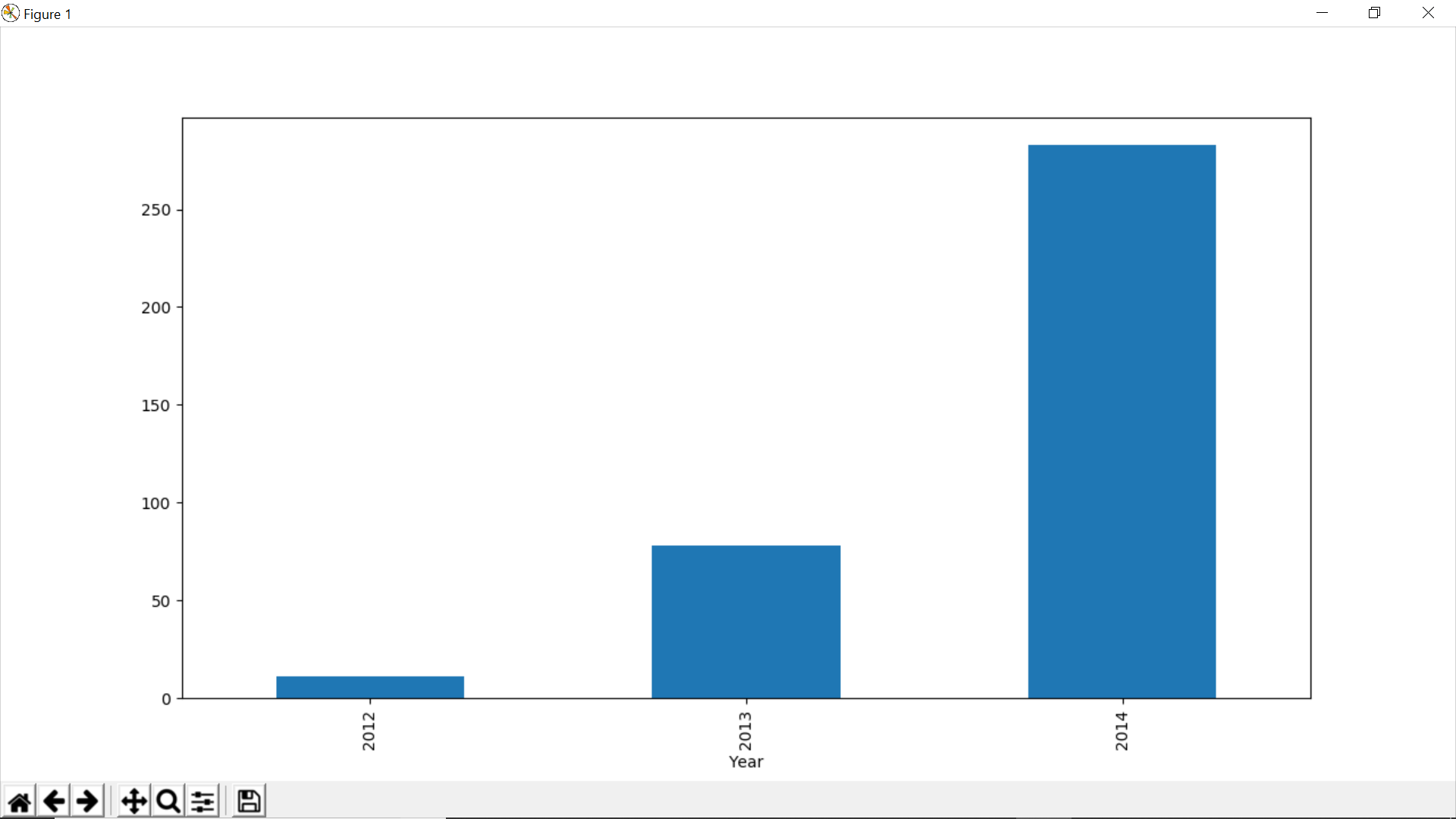
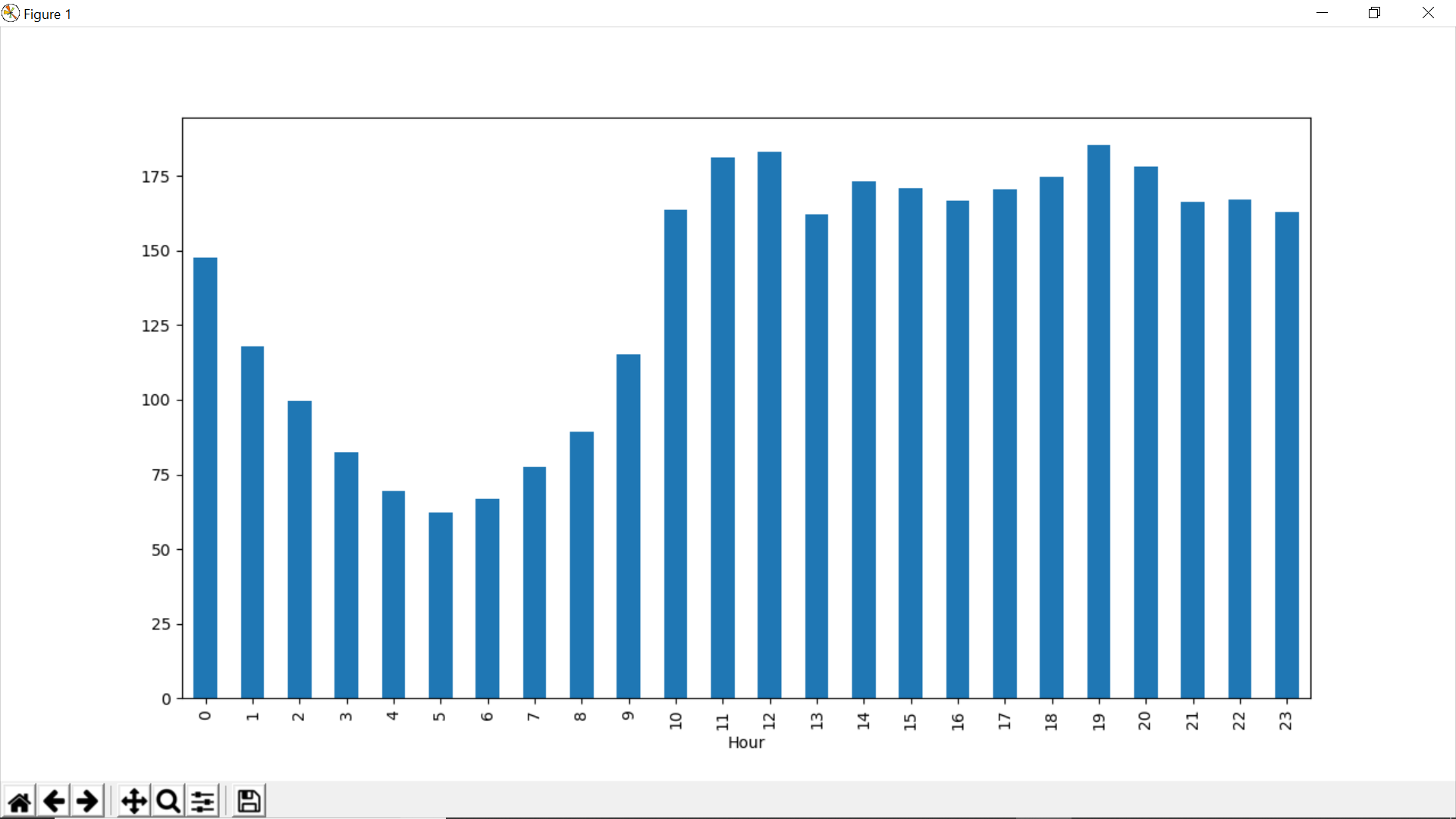
1. Traffic during the peak hours will be high.

* **Explanation** - People will travel to work, college etc.

***Feature Extraction***

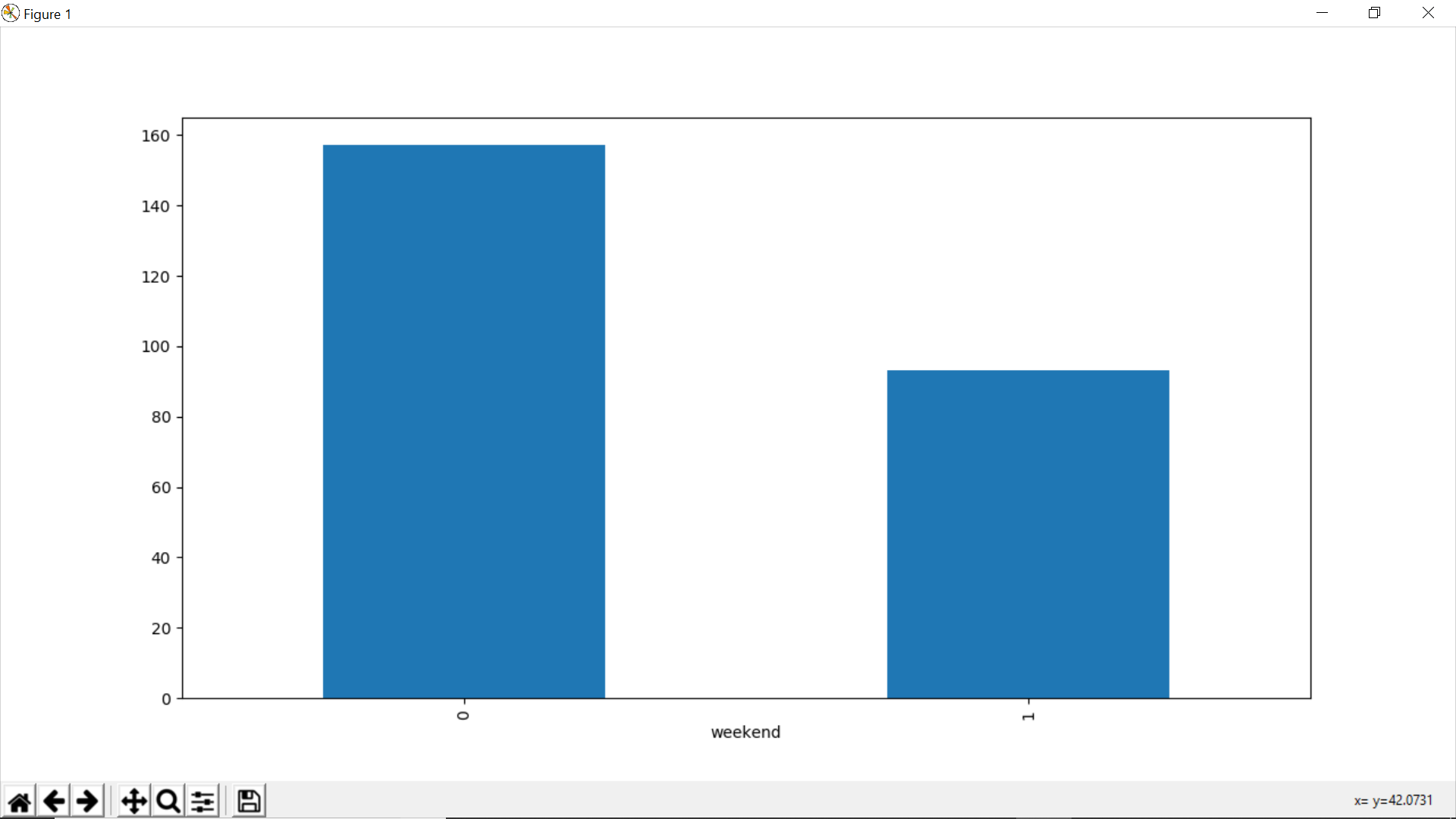
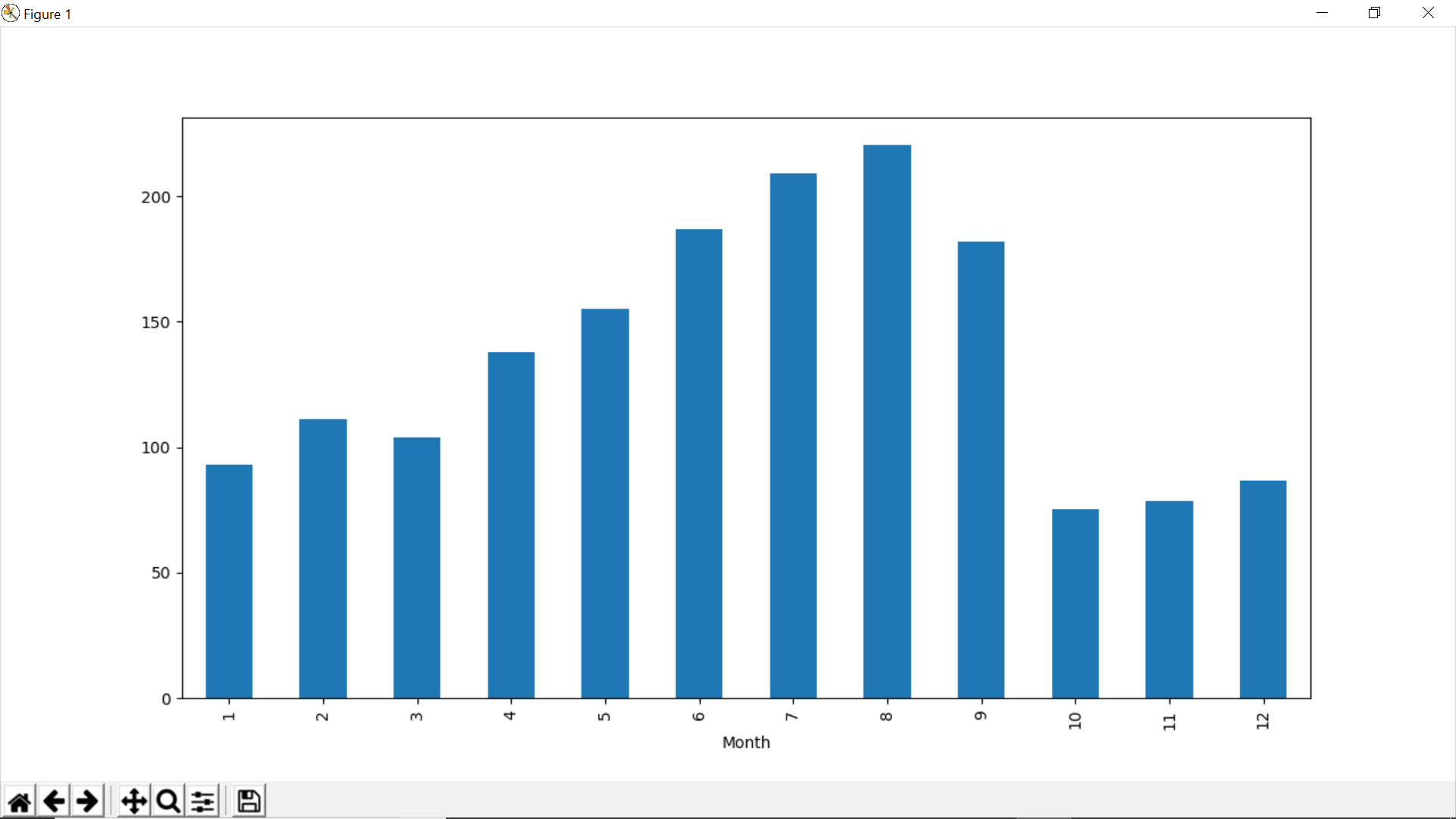
We will extract more features to validate our hypothesis. Change the data type of Datetime to datetime format from object format, otherwise we cannot extract features from it. We will extract the time and date from the Datetime. Then extract the year, month, day and hour from the Datetime to validate our hypothesis. We made a hypothesis for the traffic pattern on weekday and weekend as well. So, we will extract the day of week from Datetime and then based on the values we will assign whether the day is a weekend or not.

***Exploratory Analysis***

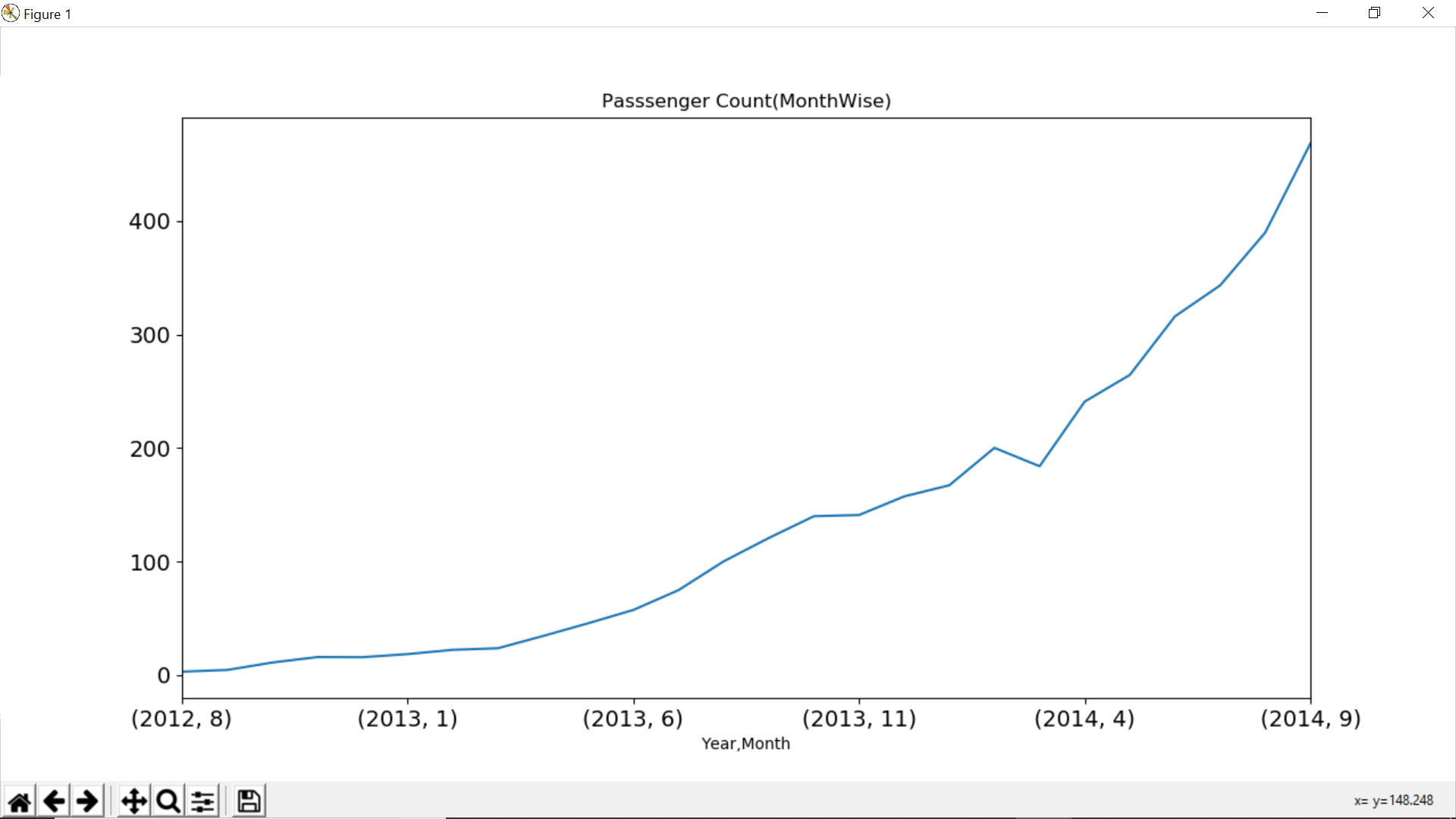
* For hypothesis 4, the peak traffic is at 7 PM and then we see a decreasing trend till 5 AM. After that the passenger count starts increasing again and peaks again between 11AM and 12 Noon.

After Plotting for hypothesis 1, we observe that there is an exponential growth in the traffic with respect to years which validates my hypothesis.

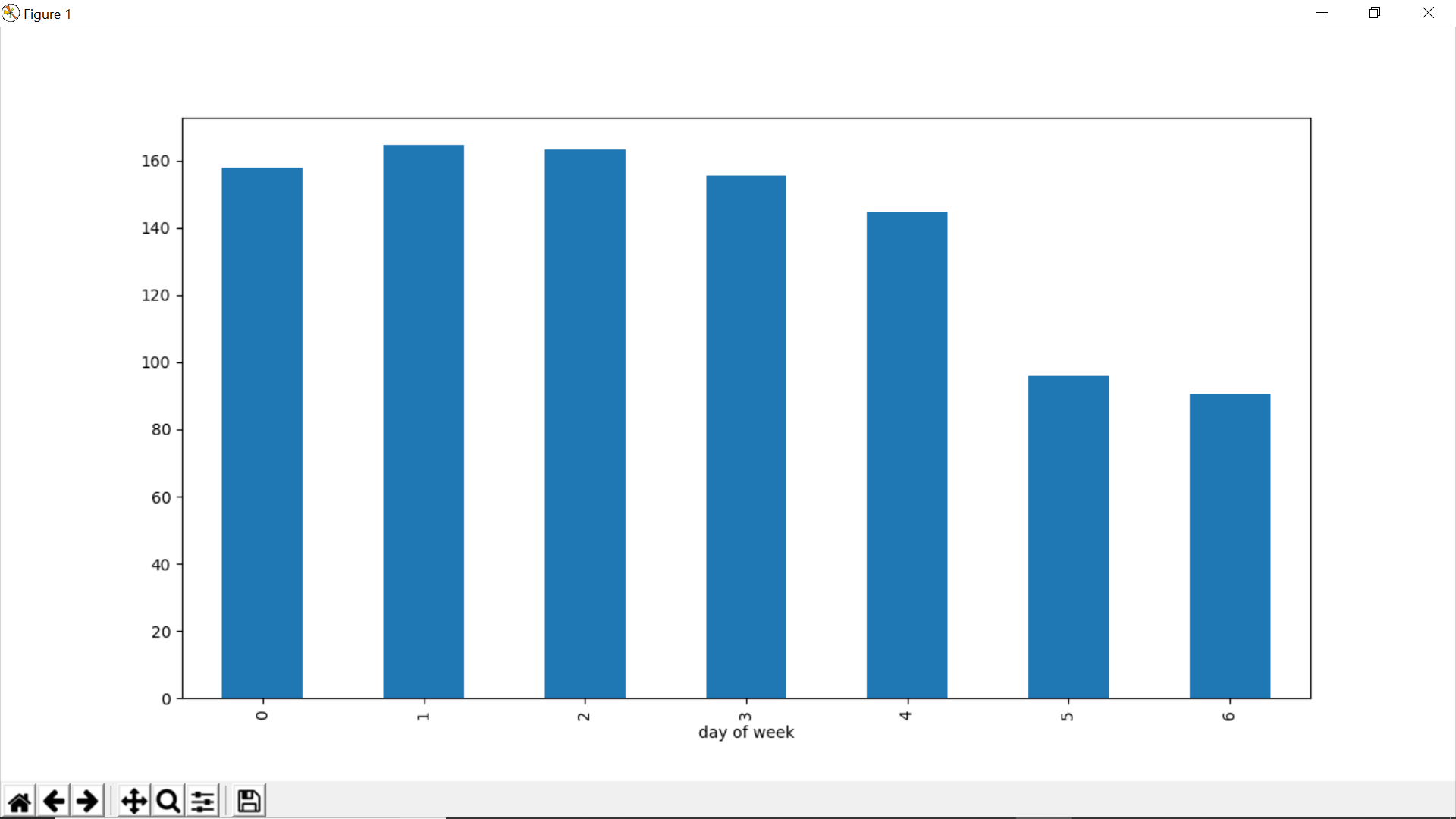
 

For hypothesis 2, we see a decrease in the mean of passenger count in last three months. This does not look right. Let’s look at the monthly mean of each year separately.

For hypothesis 3, the traffic is more on weekdays as compared to weekends which validates my hypothesis.



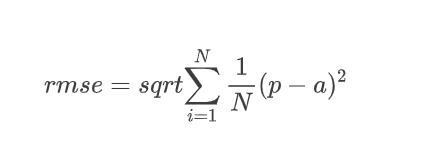
* We see that the months 10, 11 and 12 are not present for the year 2014 and the mean value for these months in year 2012 is very less. Since there is an increasing trend in our time series, the mean value for rest of the months will be more because of their larger passenger counts in year 2014 and we will get smaller value for these 3 months. In the line plot to the side, we can see an increasing trend in monthly passenger count and the growth is approximately exponential.



Additional observation, this is a day wise passenger count plot, we can infer that the passenger count is less for Saturday and Sunday as compared to the other days of the week.

***Splitting the train.csv file into training data set and validation data set.***

I did this split for both ARIMA and LSTM models. I used a time-based split. If I choose to split randomly it will take some values from the starting and some from the last years as well. It is like predicting the old values based on the future values which is not the case in real world scenarios. In case of ARIMA, the first 21 months are used for training and the last 3 months for validation as the trend will be the most in the recent data. In case of LSTM, the first 15k samples are used for training and the last 3.2k samples for validation out of the approx. 18.2k samples in the train.csv file.



Used **RMSE** as the performance measure to compute the accuracy of the predictions made by the models. It is the standard deviation of the residuals. Residuals are a measure of how far from the regression line, data points are. The formula is to the left.

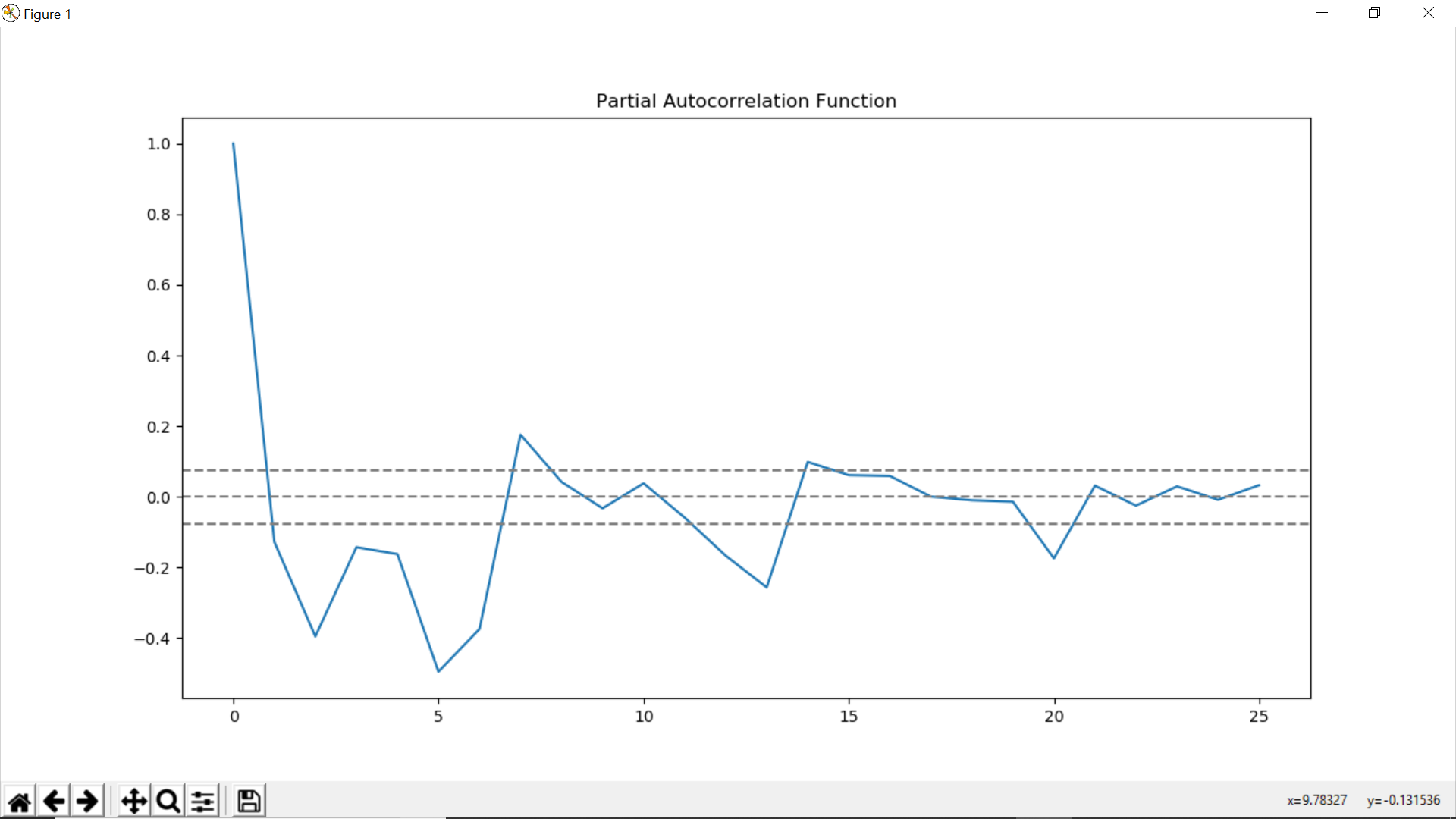
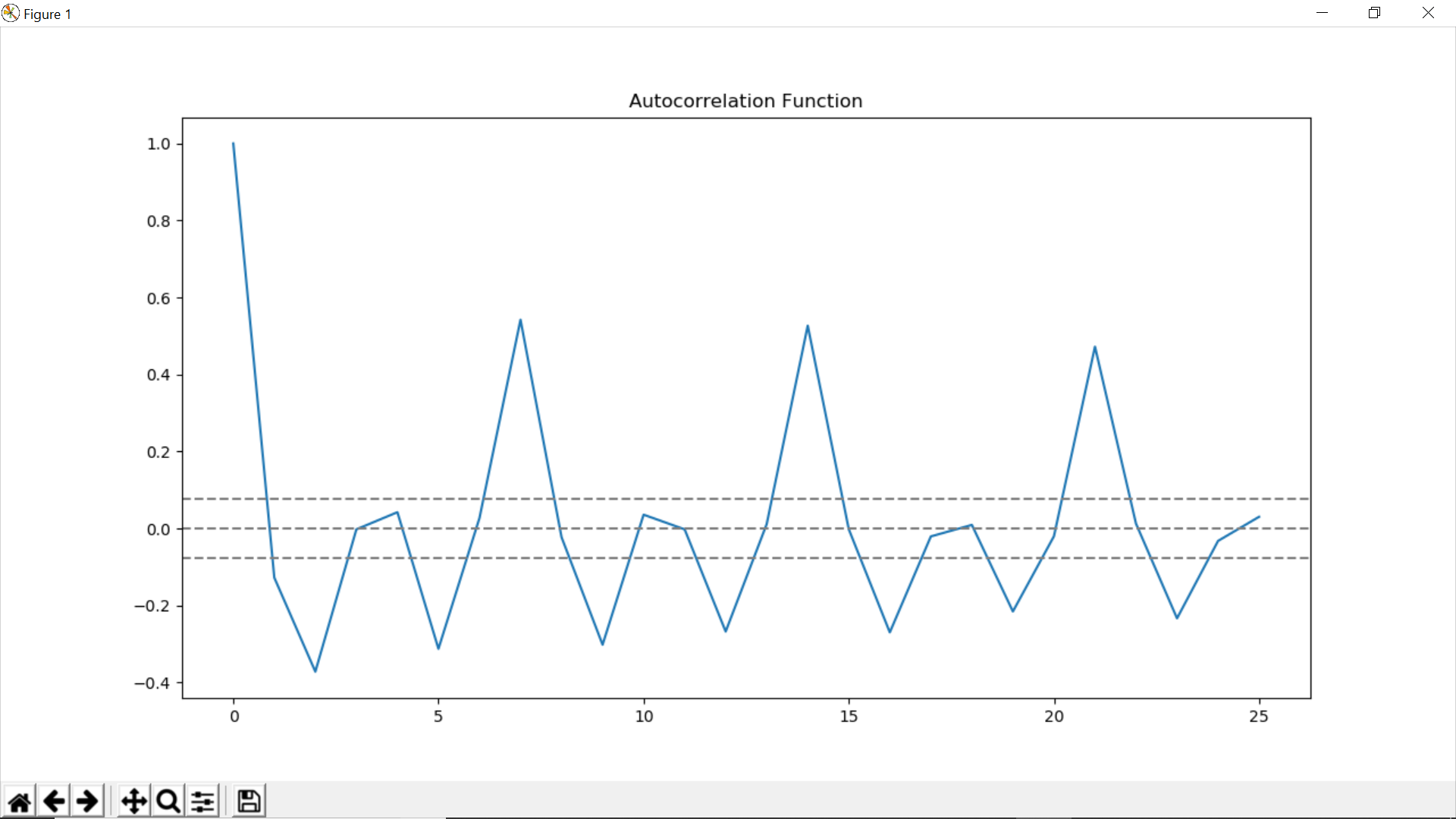
***Transforming the time series to be used by both ARIMA and LSTM models.***

By making the series stationary the skill of the predicting model increases. To do so I need to remove the trend and the seasonality in the series. **For ARIMA:** To remove the increasing trend, take log transformation which penalizes higher values more than smaller ones. Then, take rolling average on it with the window size of 24 since each day has 24 hours. Take a difference between the log transformed version and the moving average version and drop the first 23 null values. Perform the Ad fuller test to check for stationarity (explained in the code so as how to interpret it). Stabilize the mean of the time series by differencing to remove the trend. Now, to remove the stationarity we decompose the series into trend, seasonal and residuals using seasonal decompose. Then check for the stationarity of residuals using AD fuller test. They turned out to be stationary. **For LSTM:** Before making the series stationary, we need to transform the series into a supervised learning problem because Keras LSTM model assumes that the data is divided into input and output parts. This can be done 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. Use shift () function in Pandas that will push all values in a column/vector of a data frame down by a specified value for places, to become the input component to the model, this value is 1 in my case. The time series without any shift will be the output component for the model. The NaN value in the input vector of the data frame willbe replaced with a 0, which the LSTM model will learn as the starting point of the training data set. This data frame is now ready for supervised learning. *timeseries\_to\_supervised ()* function in the code implements this functionality. Now, to transform the time series into stationary series we need to remove the trend as it usually results in better forecast skills. This removed trend can be added back to forecasts later for the prediction to be in the original scale and thus we can calculate a comparable error score as well. The trend removal can be achieved by differencing the data. The observation from the previous time step (t-1) is subtracted from the current observation (t). We now have a differenced series, which includes the changes to the observations from one-time step to the next. Difference () function in the code implements this functionality. The first observation in the series is not used as there is no prior observation than it with which to calculate a differenced value. This process is inverted to get the predictions back to their original scale. *Inverse\_difference ()* function achieves this functionality. Now, we need to scale the time series. Hyperbolic tangent is the default activation function for LSTMs, they output between the range of -1 and 1. It is better to give the data to the nets in the range of the activation functions used by them. We will scale our data in the range of -1 to 1. To avoid data leakage or any sort of biasness, the scaling coefficients (min and max) values are taken from the training dataset and applied to scale the validation/test dataset and forecasts are also made in the same scale. Use the MinMaxScaler class to do so. It is a scikit-learn class and requires data to be provided in a matrix format with rows and columns. I used data frames for this. *scale ()* function in the code implements this functionality. Scale needs to be inverted back to original for the forecasts made so that we can compute comparable error score. The *invert\_scale ()* function in the code implements this functionality.

**PROCESSING**

***Arima Model (Auto Regression Integrated Moving Average)***

Arima model takes into consideration both the trend and the seasonality in the time series data to make predictions. There are three parameters (p, d, q) to specify an ARIMA model. p is the order of the autoregressive model (number of time lags), d is the degree of differencing (number of times the data have had past values subtracted), q is the order of moving average model. Therefore, we need to find the optimized values for these parameters before fitting the model on our training data set of the time series. Use ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) to find those values. ACF is a measure of the correlation between the time series with a lagged version of itself. PACF measures the correlation between the time series with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons.



* p value is the lag value where the PACF chart crosses the upper confidence interval for the first time. It can be noticed that in this case p=1.
* q value is the lag value where the ACF chart crosses the upper confidence interval for the first time. It can be noticed that in this case q=1.

ACF

PACF

We will fit the AR model and the MA model on the training data separately using the p and q values respectively using the stats models library. The AR (autoregressive model) specifies that the output variable depends linearly on its own previous values. The MA (moving-average model) specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

***LSTM (Long Short-term memory) Network***

***Model Development***

LSTM can learn and remember long sequences. Keras identify this as being stateful, we need to set the stateful argument to “True” when defining an LSTM layer. In Keras, an LSTM layer maintains state between data within one batch. A batch of data is a fixed no of samples from the training dataset which are used to compute the gradient descent and make updates to the parameters. LSTM layer clears the state between batches by default, we need to maintain the state for within each epoch and reset them among epochs using the reset\_states() function. Also, Input to the LSTM layers are given in a 3D matrix: [*samples, time steps, features*]. **Samples** is an observation in the data set. **Time steps** are separate time steps of a given feature for a given observation. **Features** are separate features observed at the time of observation. My input would be simple as each time step in the domain is one separate sample, with one timestep and one feature. I have the training data into supervised form, now it must be reshaped into samples/timestep/ features format for the LSTM layer to use it. “batch\_input\_shape” argument is used to specify the shape of the input data to be used by the LSTM layer. It is a tuple that specifies number of samples to read in each batch, the number of time steps, and the number of features. The batch size along with the number of epochs, defines how fast the gradient descent converges and the parameters are learned. There is one more parameter in the LSTM layer, it is the number of neurons, also called the number of memory units or blocks. It is a user parameter which could be set between 1 and 5. We use a single neuron in the output layer with a linear activation to make predictions at the next time step. We need to use a backend mathematical library such as Theano or Tensor Flow for compiling the net. For compiling the net, I used “mean\_squared\_error” as the objective function as it is helpful in computing RMSE scores, and ADAM as the optimization algorithm for computing the gradient descent. After compiling, the model is be fit to the training data. Because the network is stateful, we will maintain the state within an epoch and reset it among the epochs for the desired number of epochs. The samples within an epoch are shuffled prior to being exposed to the net, by default. We do not want this; we need to maintain the state of the net as it learns the sequence of the samples. We can disable this by setting shuffle to false.

Set the verbose to 0 to disable the default debug info sent by the model as it learns. fit\_lstm () function in the code implements all these functionalities mentioned above. Its arguments are training dataset in a supervised learning format, a batch size, a number of epochs, and a number of neurons. The batch\_size is set to 1 because we want online training for our model so that it converges faster and thus computes the optimal parameter values for making predictions faster. The *predict ()* function on the model will also use batch size of 1 because we want to make one-step forecasts on the test/validate data.

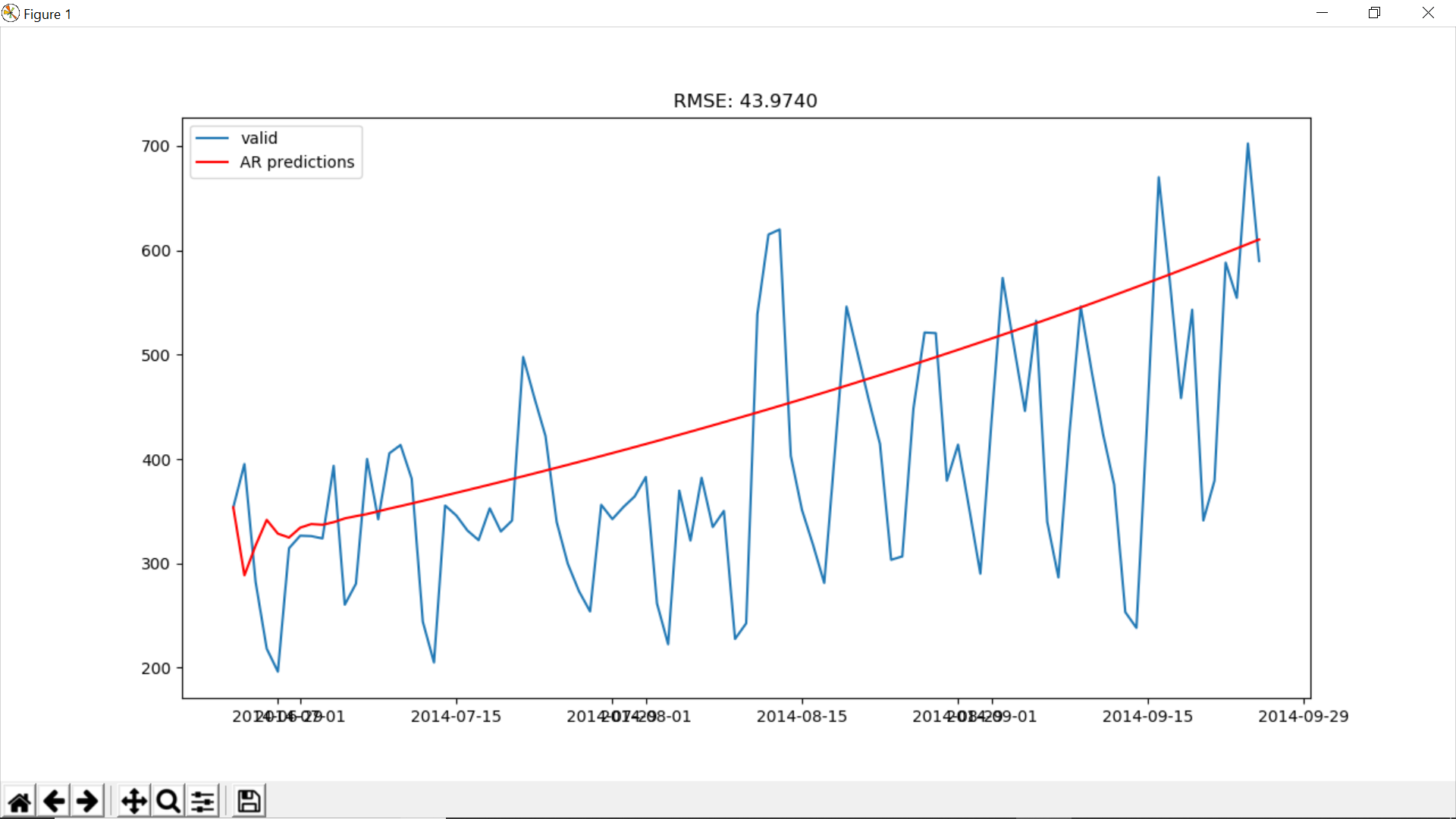
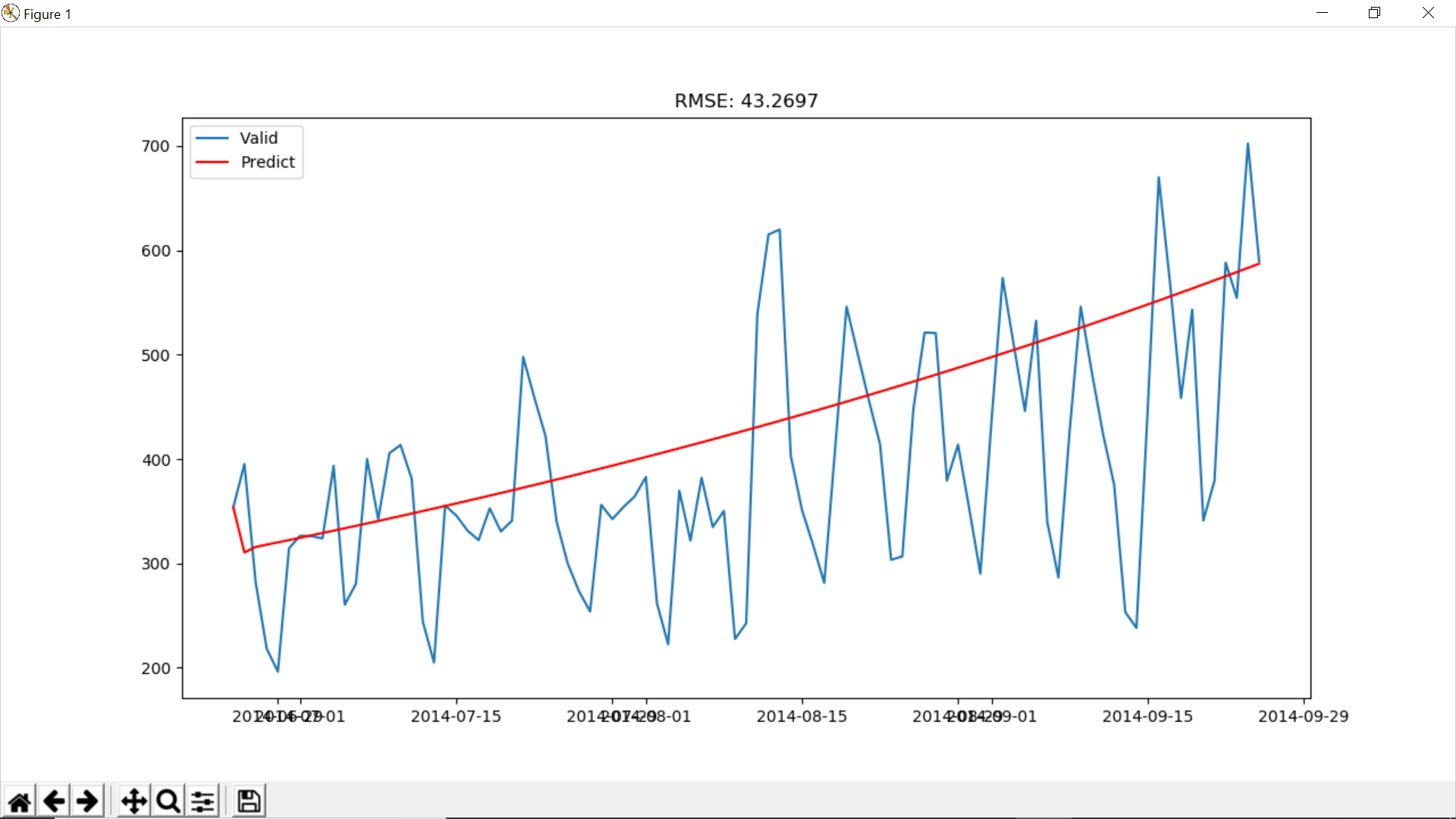
Used these following configurations, found with a little trial and error; Batch Size: 1, Epochs: 200, Neurons: 1.

***LSTM Forecast***

After fitting the LSTM model to the training data, we use it to make forecasts. Since there were computational limitations on my end, I decided to fit the model once on all of the training data, and then predict each new time step one at a time from the validation/test data, I call this fixed approach. Call the predict () function on the model to make forecasts. 3D inputs are required to be passed as an argument for making predictions as well. Thus, we use the observation at the previous time step to make prediction at the current time step of the validation/test data set. The input also needs to be reshaped into a 3D NumPy array. For every input provided to the predict () function an array of output is received. The output is a 2D NumPy array with one value for a single 3D input. forecast\_LSTM () function in the code implements this functionality. The arguments to this function are a fitted model, batch size of 1 as we need to provide observation at the previous time step only, the function we reshape the input into a 3D NumPy array and return a 2D array with a floating point in it as the prediction for it. We reset the state of the model after every epoch during training. We do not want to do the same while forecasting because we would like the model to build up state as we forecast each time step in the validation/test dataset. Need to set a good initial state for the net prior to forecasting on test dataset. I seeded the state for the net by making a prediction on all samples in the training dataset which we used for training the model. Later half of the code implements this functionality of seeding the state and making predictions on the test/validation data.

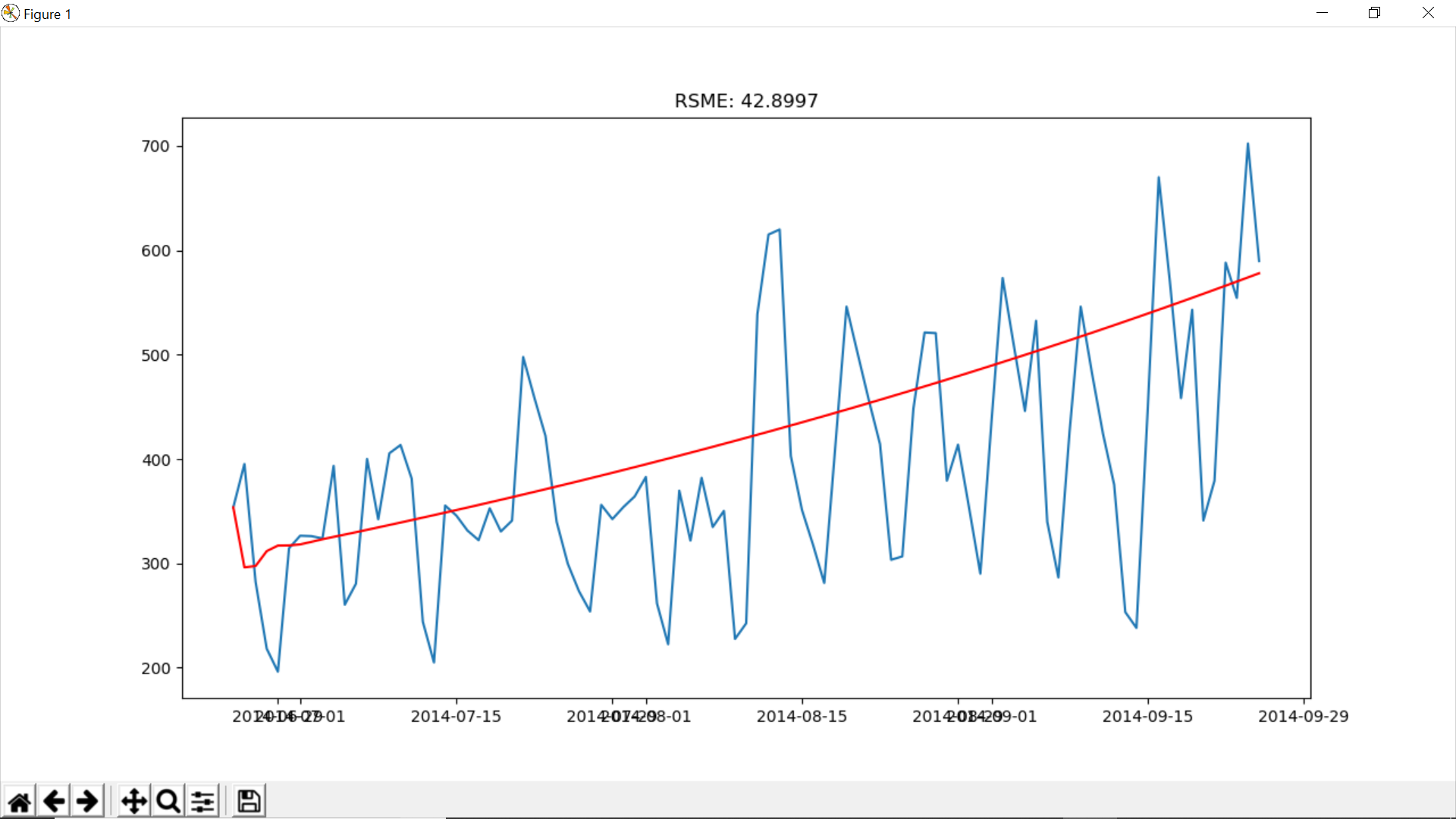
**RESULTS:**

***Arima Model:*** Validation Curves for AR model and MA model for predictions on the validation set after rescaling it back to the original scale.

AR Model validation curve. RMSE Score 43.97 on validation data set.

Combine both the AR and MA models to fit the ARIMA model onto the training data set using the stats models library and plot the validation curve after predicting for the validation set and rescaling it back to the original scale.

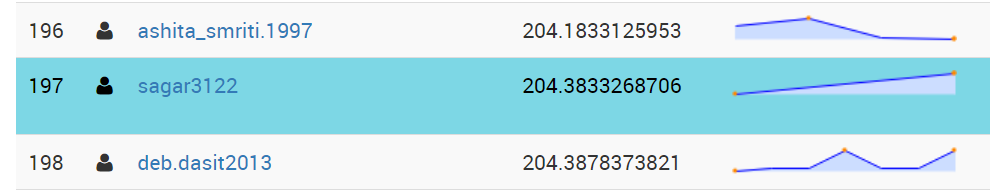


ARIMA Model validation curve. RMSE Score 42.90 on validation data set.

Now, I made daily predictions on the test data set using the ARIMA model and converted them into hourly predictions and submitted these on the analytics Vidhya website for this online challenge and received a RMSE score of 208.

MA Model validation curve. RMSE Score 43.27 on validation data set.

***LSTM Network Model:*** This model gave a **RMSE** score of **75.03** on the validation set and a **RMSE** score of **204.38** on the test data set which is available through the public leaderboard on the analytics Vidhya website.



**ANALYSIS OF RESULTS/APPROACH:**

The RMSE scores achieved by this implementation of ARIMA model are the best that any other implementation of ARIMA model would get on this time series data. Also, this implementation of LSTM yields in RMSE scores which are worse than what I achieved using ARIMA. I suppose there are multiple points that need to be addressed which explains the performance of this implementation of a LSTM model. Firstly, neural nets require high computational power to train over a relatively huge training data set to yield any useful results. Here, I have taken the epochs to be 200 which is relatively less for a neural net. This is a direct effect of limitations of computational power that both my machine and google Colab with GPU impose for a proper implementation of this LSTM model. Provided the appropriate computational power I would like to see the results this model returns for a different combination of epochs and neurons; I would like to set the epochs to around 1500 at least. Also, neural nets provide better results when the training data set is big. I would suppose the results would be better if the training data set is for a couple of previous decades rather than just 25 months. Secondly, I am not so sure about the initial state set for the net prior to performing forecasting for validation/test data set. I have seeded it by making a prediction on all samples in the training dataset which we used for training the model. There would be some other good ways of setting up the state for the net before predicting for validation/test data. Thirdly,I haveused the fixed approach for making predictions for the validation/test data set as mentioned earlier.A little better but computationally expensive approach is a dynamic approach. We can re-fit the model each time when the predictions are made for a time step of the testing/validation set are made available and use the re-fitted model to make predictions for the next time steps. I suppose, this will increase the skill of the model in making accurate predictions and decrease the RMSE scores as well.

**CONCLUSION**

I conclude that with proper tuning of the hyper parameters and adapting the approaches mentioned in the analysis of result/approach section LSTM will surely outperform any implementation of a stats model for time series data.

**REFERENCES**

<https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/>

<https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>

<https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>

<https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/>