Forecasting a Weekly Indoor Room Temperature Using Statistical Methods and Machine Learning Techniques

Jamal Ahmadov
Petroleum Engineering
University of Louisiana at Lafayette
Lafayette, LA, USA
jamal.ahmadov1@louisiana.edu

Muhibbur Rahman
Mechanical Engineering
University of Louisiana at Lafayette
Lafayette, LA, USA
muhibbur.rahman1@louisiana.edu

***Abstract*— Forecasting indoor temperature of buildings has many potential applications. Most important of them is forecasting of energy consumption of a building. Using energy consumption forecasting, a better energy management plan can be made to achieve energy efficiency. The weekly indoor temperature was forecasted using machine learning and classical methods. The Root Mean Squared Error (RMSE) is reported and the best model is recommended. The results showed that machine learning techniques, such as Long Short-Term Memory and multiple linear regression outperform the classical time series forecasting methods such as ARIMA and Vector Autoregression. Moreover, adding seasonality component to the ARIMA model improved the forecasting results.**

Keywords—Forecasting, Machine Learning, Root Mean Squared Error

I. INTRODUCTION

Electricity and electronics changed the human life forever. And Internet of things (IoT) have been changing our life without us even noticing [1-3]. Sensors and actuators have been around for quite a long time. Until few years ago these sensors and actuators have acting as individual entities. Internet of thing is making all these sensors as connected nodes. It is the advent of IoT that gives way for us to design a smart home, smart hospital or even a smart city.

Smart buildings of today are packed with variety of sensors including temperature, humidity sensors, motion sensors, smoke sensors, carbon monoxide sensors as well as accessibility and security sensors. Data from all these sensors are stored in a cloud and analyzed to make intelligent decisions regarding the building.

Energy efficiency of large buildings and comfort of the people inside the buildings are some important goals that were being achieved by analyzing the data from all the sensors. Heating Ventilation and Air Conditioning (HVAC) is one of the most energy extensive unit of a building. Analyzing the historical data from the temperature and humidity sensors and motion sensors it is possible to make decisions which can ensure better energy consumption [4, 5] as well as individual comfort in the buildings. Information coming from different buildings can then be aggregated. This aggregated data can be used to forecast the energy consumption of the whole city.

Predicting indoor temperature have been done before with scenarios from individual houses to offices and buildings [6, 7]. Methods followed in these literatures range from Neural Network to customarily methods. Research works was also found focusing on the energy efficiency in IoT context where indoor room temperature was used [8-11]. Less work had been devoted where outside temperature is used to predict the indoor temperature [12].

Accuracy of prediction has important role while planning to design a smart building or a smart city. Better forecasting makes it possible for better estimate of energy consumption which is one of the main purposes. Good forecasting accuracy will ensure a good plan to operate the HVAC machinery. Accurately forecasted temperature can be combined with individual stress level data to regulate indoor temperature.

In the present work, indoor room temperature of different rooms in the building of the University of Louisiana at Lafayette was predicted based on the historical data. There are temperature and humidity sensors and motion sensors in almost every room in the building. And the data from all these sensors have been recorded in the server for one and half years [13]. These historical data from indoor sensors as well as outside historical temperature data was used in this work to predict the indoor temperature of different rooms of the building.

II. METHODOLOGY

A. Theory

1) Univariate forecasting

Most of the time series models work on the assumption that the time series is stationary. Intuitively, if the time series has a particular behavior over time, there is a very high probability that it will follow the same in the future. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series. Stationarity is defined using very strict criterion. However, for practical purposes it can be assumed that the series is stationary if it has constant statistical properties, such as mean, variance and an autocovariance over time.

Dickey-Fuller test is used to check for stationarity. The null hypothesis of this test is that the time series is non-stationary. The test results comprise of a test statistic and critical values for different confidence levels. If the 'test statistic' is less than the 'critical value', the null hypothesis can be rejected, and the series is considered to be stationary.

The trend and seasonality are two major reasons behind non-stationarity of a time series. The trend is a variation of mean over time while seasonality corresponds to the variations at specific time-frames. The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to obtain a stationary series. After this, statistical forecasting techniques can be implemented on this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.

One of the first steps in reducing the trend is a transformation which penalize higher values more than smaller values. The

log transform was applied in this analysis. Some other common trend reduction techniques include "aggregation" which is taking average for a time period like monthly/weekly averages; "smoothing" that is taking rolling averages, such as simple moving average or exponentially weighted moving average and "polynomial fitting". However, these simple trend reduction techniques do not work in all cases. One of the most common methods of dealing with both trend and seasonality is "differencing". In this technique, the difference of the observation at a particular instant with that at the previous instant was estimated. Depending on the degree of trend and seasonality, first or second order differencing was applied.

Auto Regressive Integrated Moving Average (ARIMA) model is a popular technique to perform univariate time series forecasting. It explains a given time series based on its own past values, including its own lags and the lagged forecast errors, so that equation can be used to forecast future values. The first step to build an ARIMA model is to make the time series stationary. Because, term "Auto Regressive" in ARIMA means it is a linear regression model that uses its own lags as predictors. Linear regression models, as you know, work best when the predictors are not correlated and are independent of each other. The ARIMA forecasting for a stationary time series is a linear equation and the predictors depend on the parameters (p,d,q) of the ARIMA model:

1. Number of AR (Auto-Regressive) terms (p): AR terms are the lags of dependent variable. For instance, if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).
2. Number of MA (Moving Average) terms (q): MA terms are lagged forecast errors in prediction equation. For instance, if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.
3. Number of Differences (d): the minimum number of differencing needed to make the series stationary.

Therefore, ARIMA model is one where the time series is differenced at least once to make it stationary and the AR and the MA terms are combined.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \ldots + \phi_q \epsilon_{t-q}$$

"q" and "p" values are determined from Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) charts, respectively. They are equal to the lag value where the charts cross the upper confidence interval for the first time.

2) Multivariate forecasting

a) Vector Autoregression (VAR)

Vector Autoregression (VAR) model is used to forecast a temperature using humidity, outside temperature, and temperature at previous two days as the predictors. VAR is a multivariate forecasting algorithm that is used when two or more time series influence each other. It is considered as an Autoregressive model because each variable (or time series) is modeled as a linear combination of past values of itself and the past values of other variables in the system. For example, the system of equations for a VAR model with two time series (variables "Y1" and "Y2") and containing up to one lag of each of the predictors is as follows:

$$Y_{1,t} = \alpha_1 + \beta_{11,1} Y_{1,t-1} + \beta_{12,1} Y_{2,t-1} + \epsilon_{1,t}$$
$$Y_{2,t} = \alpha_2 + \beta_{21,1} Y_{1,t-1} + \beta_{22,1} Y_{2,t-1} + \epsilon_{2,t}$$

The basis behind Vector Autoregression is that each of the time series in the system influences each other. That is, the series can be predicted with past values of itself along with other series in the system. Granger's causality test is used to test this relationship before building the model. Granger's causality tests the null hypothesis that the coefficients of past values in the regression equation is zero. In simpler terms, the past values of time series (X) do not cause the other series (Y). So, if the p-value obtained from the test is lesser than the significance level of 0.05, then, the null hypothesis can be rejected.

Cointegration test helps to establish the presence of a statistically significant connection between two or more time series. When two or more time series are cointegrated, it means they have a long run, statistically significant relationship. This is the basic premise on which Vector Autoregression (VAR) models are based on.

The VAR model requires the time series you want to forecast to be stationary. Dickey-Fuller test is used to check for stationarity. If a series is found to be non-stationary, the "differencing" was applied until it becomes stationary. To select the right order of the VAR model, increasing orders of VAR model were fit iteratively and the order that gives a model with least AIC was picked. Though the usual practice is to look at the AIC, other best fit comparison estimates of BIC, FPE and HQIC can also be checked.

A serial correlation of residuals is used to check if there is any leftover pattern in the residuals (errors). Checking for serial correlation is to ensure that the model is sufficiently able to explain the variances and patterns in the time series. Durbin Watson's Statistic is used for this purpose. The value of this statistic can vary between 0 and 4. The test statistic around 2 means there is no significant serial correlation of errors.

b) Long Short-Term Memory (LSTM)

A Recurrent Neural Networks (RNNs) are networks with loops in them, allowing information to persist. RNNs process a time series step-by-step, maintaining an internal state summarizing the information that have been seen previously. Therefore, they are a type of neural networks well-suited to time series data. Long Short-Term Memory (LSTM) is a specialized RNN layer and explicitly designed to avoid the long-term dependency problem of standard RNNs. Like RNNs, LSTMs also have 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.

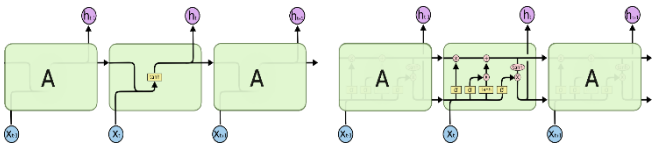


Fig. 1. Standard RNN structure (left) and LSTM structure (right).

To perform LSTM, the supervised learning problem is framed as predicting the temperature at the current time (t) given the temperature measurement, humidity, outside

temperature, and temperature at previous two days at the prior time steps. The variables were scaled. The LSTM is defined with 50 neurons in the first hidden layer and 1 neuron in the output layer for predicting temperature. The prediction will be performed for different number of previous time steps, namely 3, 6, 9, 12. Providing more than 1 lag of input time steps is important given the use of backpropagation through time by LSTMs when learning sequence prediction problems. Hence, the number of features will be 16, 31, 46 and 61 given the presence of 5 features at each previous time step and a temperature at the current time step. The Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent will be used. The model will be fit for 30 training epochs with a batch size of 32. The scaling was inverted at the end of forecasting.

c) Multiple Linear Regression (MLR)

Multiple linear regression is an extension of linear (OLS) regression that uses just one explanatory variable. MLR attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data.

$$y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n + \varepsilon$$

B. Data Collection

The temperature and humidity data for each day is collected between January 15 and February 15 of the year 2020. Also, sensor data from January 13 and January 14 were collected to introduce the features such as temperature of the previous day and temperature of 2 days before. The temperature data were sampled every 2 hours that sum up to 12 data points per day. Data was collected for the rooms 130, 138, 212, 214, 269, 276A.

The outside temperature was collected from the weather data online. Several online sources were considered. Outside temperature was collected for the same time stamp of the room temperature [14]. These historical temperatures were also from January 15 to February 15 of the year 2020. After collecting the indoor sensor data and outside temperature data, they were combined in s single spreadsheet for every room and prepared for the analysis.

While collecting the indoor sensor data there were some challenges. The battery level for some data during some timestamp were zero even though the data was consistent. So, data was considered ignoring the battery level. Initially motion sensor data was considered to be included as one of the attributes. But there were missing data for majority of the days considered. Therefore, motion sensor data was not included in the work. Most of the online source provided the historical temperature in terms of highest and lowest temperature for a day. One source was found to provide a continuous graph of temperature over hours in a day. So, the outside temperature was collected after interpreting the graph which may introduce 2/3F˚ error for some values.

III. RESULTS & DISCUSSION

The results from the analysis will be shown and discussed for one of the rooms in this section. The corresponding plots and analysis data for other 5 rooms are given in Appendix A. In addition, the python codes used to run all the models are given in Appendix B.

A. ARIMA model

The stationarity of the data was tested first using Dickey-Fuller test. The test statistic is smaller than the 10% critical values so we can say with 90% confidence that this is a stationary series. To have a more stationary time series, we will apply first order differencing and check the stationarity again. After differencing, the test statistic became smaller than the 1% critical values and the time series can be considered stationary with a high confidence. Therefore, the minimum number of differencing needed to make the series stationary or "d" value of ARIMA model is 1.

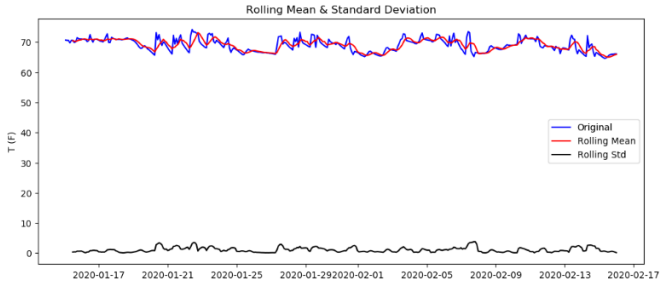


Fig. 2. Rolling mean and std for original time series.

Results of Dickey-Fuller Test:
Test Statistic -2.611693
p-value 0.090605
#Lags Used 12.000000
Number of Observations Used 371.000000
Critical Value (1%) -3.448100
Critical Value (5%) -2.869362
Critical Value (10%) -2.570937

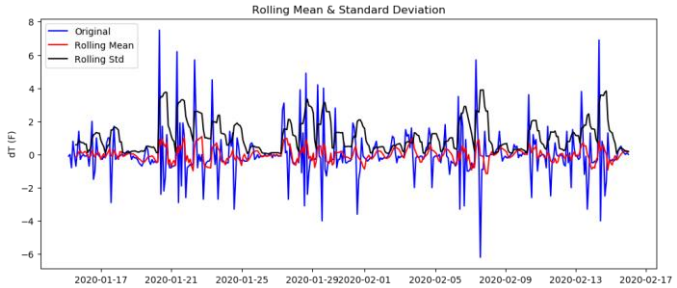


Fig. 3. Rolling mean and std for time series after differencing.

Results of Dickey-Fuller Test:
Test Statistic -7.109460e+00
p-value 3.973825e-10
#Lags Used 1.100000e+01
Number of Observations Used 3.710000e+02
Critical Value (1%) -3.448100e+00
Critical Value (5%) -2.869362e+00
Critical Value (10%) -2.570937e+00

Next, "q" and "p" values of ARIMA model are determined from Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) charts, respectively. Both charts cross the upper confidence interval for the first time at the lag value of 1 (i.e. q=1, p=1).

Next, ARIMA model was trained on the 3-week temperature data which spans from January 15 to February 7. The

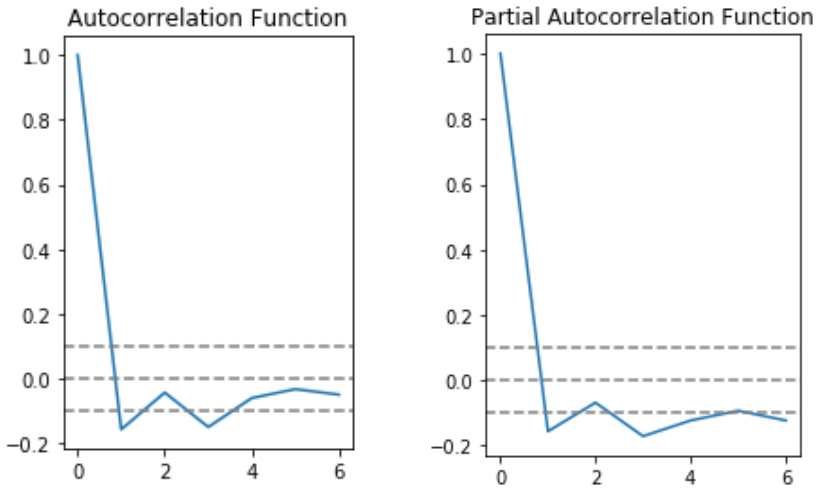


Fig. 4. ACF and PACF charts are given.

temperature forecast was performed between February 7 and February 15.

As it is seen in the figure below the performance of ARIMA model is not very good and it is not able to capture the seasonality in the time series. Root Mean Squared Error (RMSE) of the model is equal to 2.078. Although the method can handle data with a trend, it does not support time series with a seasonal component.

To incorporate the seasonality, Seasonal Autoregressive Integrated Moving Average (SARIMA) model was applied. It is an extension to ARIMA that supports the direct modeling of the seasonal component of the series.

SARIMA adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) (P, D, Q) for the seasonal component of the series, as well as an additional parameter for the period of the

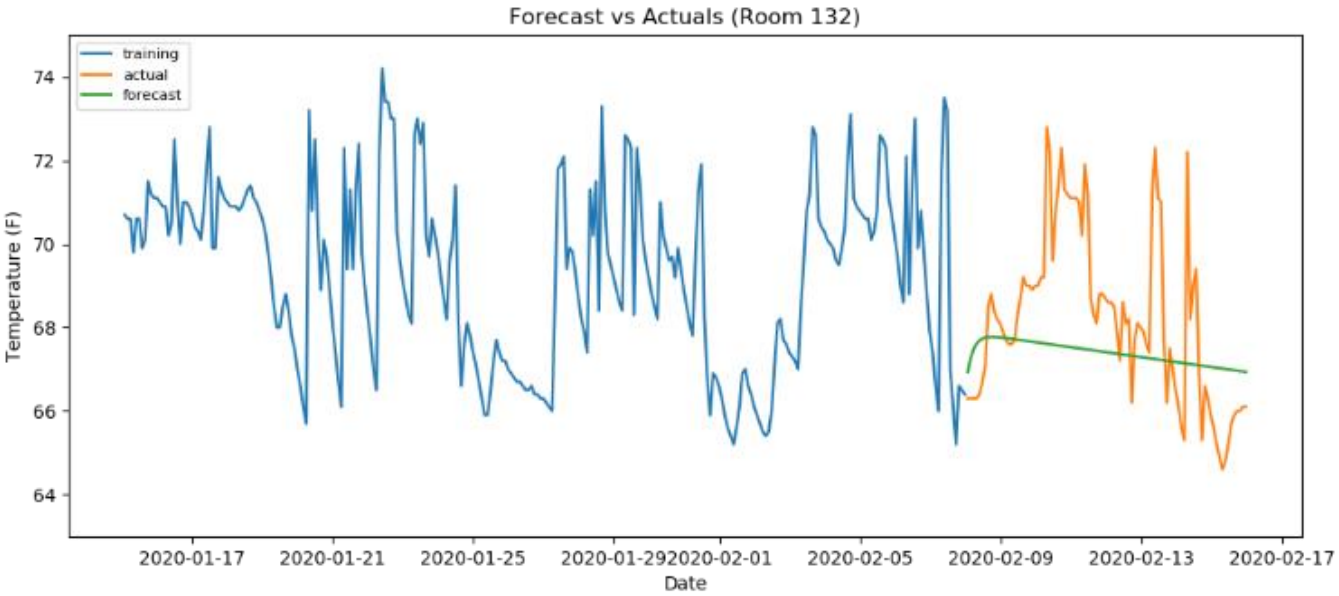


Fig. 5. Forecasting results of ARIMA model for Room 132.

We decomposed the time series into the trend, seasonality, and residuals elements. The seasonality of time series results from the day and night temperatures.

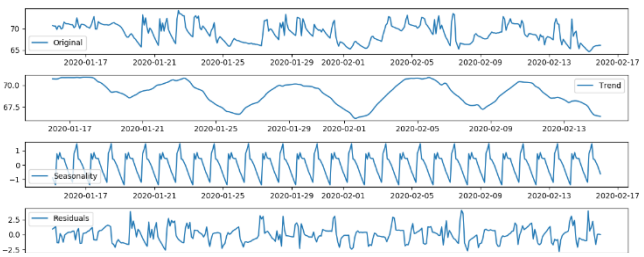


Fig. 6. Decomposition of original time series.

seasonality (m).

We used the “auto_arima” function from “pmdarima” library that identifies the most optimal parameters for an ARIMA model and returns a fitted ARIMA model. Also, the “seasonal” parameter is enabled in which “auto_arima” also seeks to identify the optimal P and Q hyperparameters after conducting the statistical test to determine the optimal order of seasonal differencing, D. The m parameter is the number of observations per seasonal cycle. Since we had 12 measurements per day, the m value was taken as 12. The figure below shows the forecasting results of SARIMA

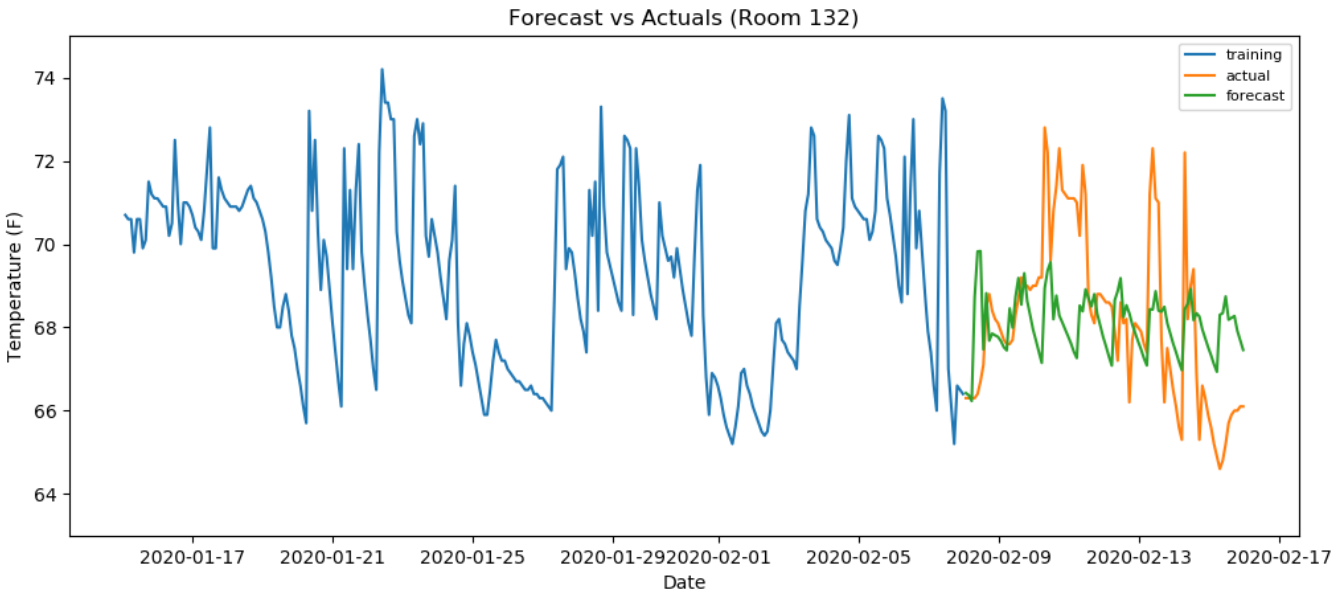


Fig. 7. Forecasting results of SARIMA model for Room 132.

model. It shows a better performance than ARIMA model with a reduced RMSE value of 1.940.

In the table below, RMSE values obtained from ARIMA and SARIMA forecasting are shown for all 6 rooms.

Table 1. RMSE values for ARIMA and SARIMA models.

	Room number					
Model	***132***	***138***	***212***	***214***	***269***	***276***
ARIMA	2.078	2.248	1.772	2.556	3.262	1.663
SARIMA	1.940	2.115	1.518	2.370	2.067	1.618

B. VAR model

Granger's causality test is first applied to test the relationship between the model variables. In other words, it was checked that if the past values of time series (X) cause the series (Y). The p-values obtained from the test are given in the table below. For most of the cases, the p-values are lesser than the significance level of 0.05 which allows us to reject the null hypothesis that the coefficients of past values in the regression equation is zero.

Name :: Test Stat > C(95%) => Signif

Temperature :: 153.69 > 60.0627 => True
Humidity :: 82.34 > 40.1749 => True
Outside Temperature :: 39.08 > 24.2761 => True
Temperature on Previous day :: 15.01 > 12.3212 => True
Temperature 2 days before :: 0.2 > 4.1296 => False

Next, we performed a first order differencing to make the variables stationary. To select the right order (p) of the VAR model, we iteratively fit increasing orders of VAR model and pick the order that gives a model with the lowest estimates of AIC, BIC, FPE or HQIC. In the table below, the order (p) of the VAR models developed for each room is given.

Table 2. Optimum order of VAR models for each room.

	Room number					
	132	***138***	***212***	***214***	***269***	***276***
p	14	12	14	13	13	13

The serial correlation of residuals was checked using Durbin Watson's statistic. The test statistic of each variable is close to 2 which means that there is no significant serial correlation

	Temperature_x	Humidity_x	Outside Temperature_x	Temperature on Previous day_x	Temperature 2 days before_x
Temperature_y	1.0000	0.0140	0.0010	0.0001	0.0117
Humidity_y	0.0000	1.0000	0.0000	0.0677	0.0577
Outside Temperature_y	0.0451	0.0001	1.0000	0.0058	0.0002
Temperature on Previous day_y	0.0000	0.0209	0.0011	1.0000	0.0000
Temperature 2 days before_y	0.0000	0.3302	0.0000	0.0000	1.0000

Fig. 8. Results of Granger's causality test.

Johanssen's cointegration test was applied next to test the presence of a statistically significant connection between variables. The results of this test are given below. It showed that all of the variables, except a temperature recorded 2 days ago, have a long run, statistically significant relationship.

of errors. In other words, the model is able to explain the variances and patterns in the time series sufficiently.

Temperature : 1.98
Humidity : 1.93

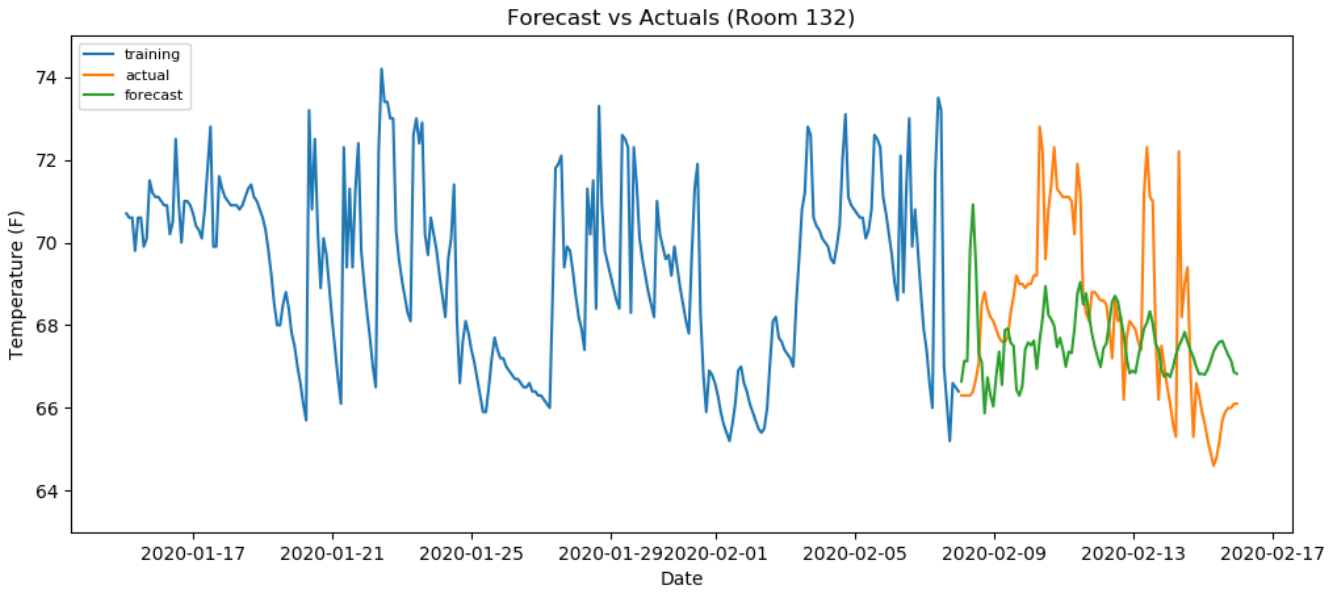


Fig. 9. Forecasting results of VAR model for Room 132.

Outside Temperature : 1.97
Temperature on Previous day : 2.44
Temperature 2 days before : 2.35

After forecasting temperature for the test dataset, the results were inverted back to the original scale. As it can be seen the VAR model did not show improved performance when compared to ARIMA and SARIMA models. RMSE value is equal to 2.112 for the temperature forecasting of Room 132.

In the table below, RMSE values obtained from VAR forecasting are shown for all 6 rooms.

Table 3. RMSE values for VAR model.

	Room number					
Model	***132***	***138***	***212***	***214***	***269***	***276***
VAR	2.112	2.146	1.645	2.802	2.206	1.117

C. LSTM model

It was observed that as we provided more time steps the performance of LSTM model improved. It was decided to use a model with 12 previous time steps (or 1 day) as it performs better in comparison to the models with less time steps. The figure below demonstrates the dataset when the forecasting of current temperature is based on the values of 5 variables over the last 12 time steps. Hence, there are 60 predictors and 1 response variable for this setup.

The LSTM model showed better performance than all the previous models. RMSE value for forecasting of temperature values in Room 132 dropped to 1.332. The forecasting results for other rooms are given in Appendix A.

In some cases, the training loss was slightly higher than a test loss. It can be attributed to the fact that the training loss is the average of the losses over each batch of training data. Since the model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss. Another possible reason for this could be that our test set is simpler in comparison to the training set. Hence, the network performs better on the test set than it did on the training set. Finally, our test set might be very small as such, the high accuracy we see on it is not indicative of the real accuracy and would have increased if we had a larger test set. RMSE values obtained from LSTM forecasting are shown for all 6 rooms in the table below.

Table 4. RMSE values for LSTM model.

	Room number					
Model	***132***	***138***	***212***	***214***	***269***	***276***
LSTM	1.332	0.580	1.147	1.878	1.048	0.728

	var1(t-12)	var2(t-12)	var3(t-12)	var4(t-12)	var5(t-12)	var1(t-11)	var2(t-11)	var3(t-11)	var4(t-11)	var5(t-11)	...	var2(t-2)	var3(t-2)	var4(t-2)	var5(t-2)	var1(t-1)
12	0.635417	0.726937	0.804348	0.511111	0.322222	0.625000	0.759475	0.760870	0.511111	0.311111	...	0.644112	0.760870	0.622222	0.544444	0.677083
13	0.625000	0.759475	0.760870	0.511111	0.311111	0.625000	0.783139	0.804348	0.500000	0.288889	...	0.681087	0.717391	0.622222	0.522222	0.666667
14	0.625000	0.783139	0.804348	0.500000	0.288889	0.541667	0.654835	0.891304	0.733333	0.166667	...	0.740802	0.717391	0.611111	0.511111	0.656250
15	0.541667	0.654835	0.891304	0.733333	0.166667	0.625000	0.598817	0.956522	0.811111	0.377778	...	0.768904	0.760870	0.600000	0.511111	0.656250
16	0.625000	0.598817	0.956522	0.811111	0.377778	0.625000	0.598262	0.956522	0.822222	0.411111	...	0.799963	0.760870	0.600000	0.500000	0.583333

5 rows × 61 columns

Fig. 10. Dataset of variables for LSTM model.

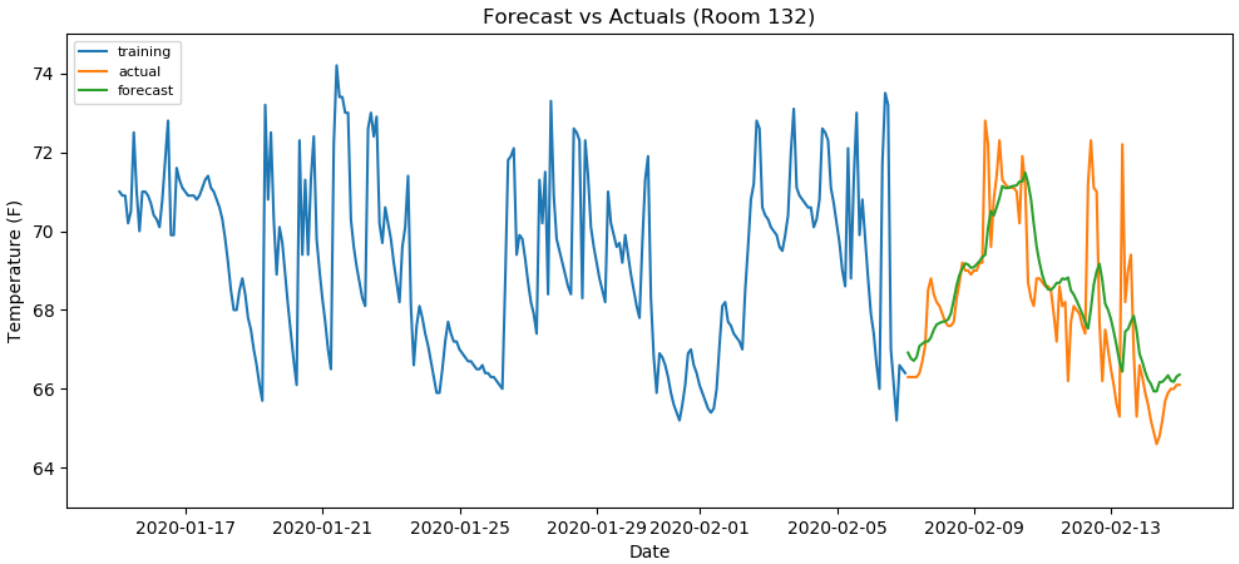


Figure 11. Forecasting results of LSTM model for Room 132.

D. Multiple Linear Regression

The dataset from the previous section where the predictors are the values of 5 variables over the last 12 time steps is used to perform linear regression. That is to say, there are 60 independent variables and one dependent variable (i.e. the forecasted temperature) in the linear regression equation. The beta coefficients were obtained from fitting on training set and used to forecast temperature values given the input of test set. The linear regression showed a similar performance to the LSTM model. RMSE value is equal to 1.433.

RMSE values obtained from linear regression forecasting are shown for all 6 rooms in the table below.

Table 5. RMSE values for linear regression model.

	Room number					
Model	***132***	***138***	***212***	***214***	***269***	***276***
MLR	1.433	0.433	0.950	1.368	0.756	0.550

IV. Conclusions

The temperature forecasting was performed for 6 rooms in the building of the University of Louisiana at Lafayette. Both univariate forecasting methods, such as ARIMA and SARIMA and multivariate forecasting methods such as VAR, LSTM and linear regression were applied. For latter methods, temperature, humidity, and outside temperature at previous time steps were used as the predictors of current temperature. The conclusions of this study are given below:

- The addition of seasonality to the univariate forecasting process improved the results. Hence, SARIMA model performed better than ARIMA model with average RMSE of 1.94 vs. 2.26.
- VAR model showed a better prediction than univariate ARIMA model with an average RMSE of 2.00. It also displayed a decent capture of trends in the test dataset.

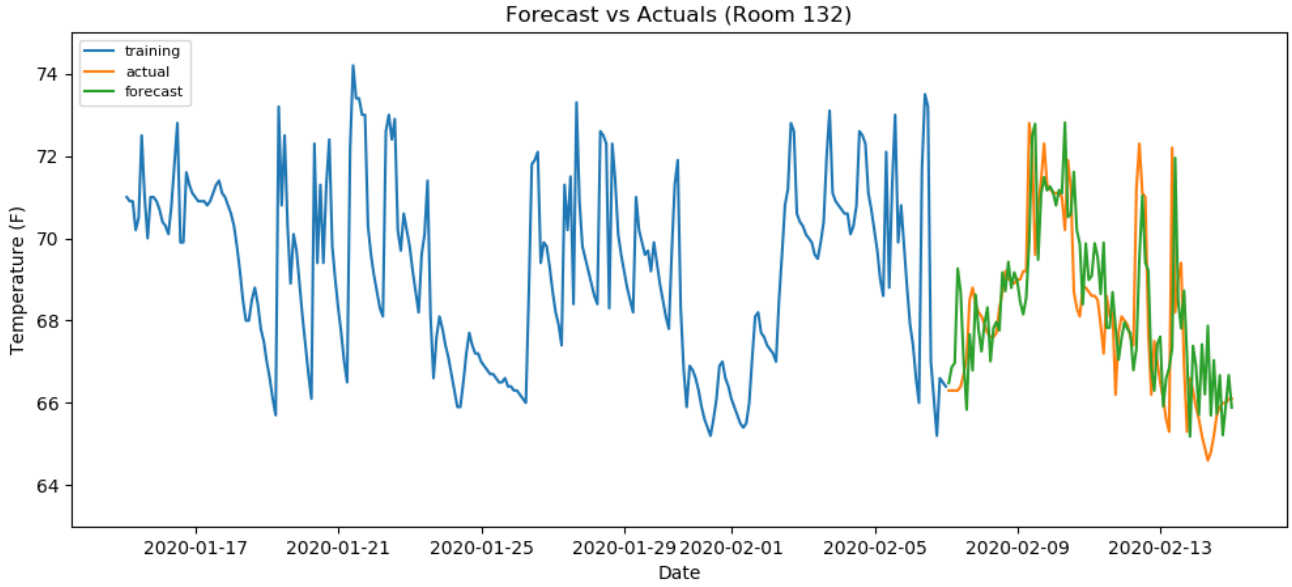


Fig. 12. Forecasting results of linear regression for Room 132.

- LSTM and Multiple Linear Regression performs better than classical techniques with an average RMSE values of 1.12 and 0.92, respectively.
- For our analysis, LSTM is proved to be powerful technique to capture the temperature trends. Including features from more previous time steps as the inputs led to the better forecasting performance.
- The results of study can be improved by adding additional features such as occupancy data. Also, the forecasting period can be extended to the monthly temperature predictions. This study serves as a basis for a future work where the energy management of particular building can be optimized when combined with different type of sensor data in an IoT framework.

V. REFERENCES

[1] P. Wang, C. Sohail, and L. Ling, "Introduction: advances in IoT research and applications," Internet Research (2016).

[2] G Mastorakis, C.X. Mavromoustakis, E. Pallis, "Beyond the Internet of Things" Batalla JM, editor. Switzerland, Springer, 2017.

[3] R. Gravina, C. E. Palau, M. Manso, A. Liotta, G. Fortino, editors, "Integration, interconnection, and interoperability of IoT systems, " New York, NY, USA, Springer International Publishing, 2018.

[4] Z. O'Neill, C. O'Neill, "Development of a probabilistic graphical model for predicting building energy performance," Applied Energy, 2016 Feb 15, vol. 164, pp. 650-8.

[5] L. Mba, P. Meukam, A. Kemajou, "Application of artificial neural network for predicting hourly indoor air temperature and relative humidity in modern building in humid region," Energy and Buildings. 2016 Jun 1, vol. 121, pp. 32-42.

[6] K. Matsui, "An information provision system to promote energy conservation and maintain indoor comfort in smart homes using sensed data by IoT sensors," Future Generation Computer Systems, 2018 May 1, vol. 82, pp. 388-94.

[7] A. Quinn, J. D. Tamerius, M. Perzanowski, J. S. Jacobson, I. Goldstein, L. Acosta, J. Shaman, 'Predicting indoor heat exposure risk during extreme heat events," Science of the total environment, 2014 Aug 15, vol. 490, pp. 686-93.

[8] M. V. Moreno, A. F. Skarmeta, A. Venturi, M. Schmidt, A. Schuelke, "Context sensitive indoor temperature forecast for energy efficient operation of smart buildings," In2015 IEEE 2nd World Forum on Internet of Things (WF-IoT) 2015 Dec 14 (pp. 705-710), IEEE.

[9] B. Spencer, F. Al-Obeidat, O. Alfandi, "Selecting sensors when forecasting temperature in smart buildings," Procedia Computer Science, 2017 Jan 1, vol. 109, pp. 777-84.

[10] B. Spencer, O. Alfandi, F. Al-Obeidat F, "A refinement of lasso regression applied to temperature forecasting" Procedia computer science, 2018 Jan 1, vol. 130, pp. 728-35.

[11] B. Thomas and M. Soleimani-Mohseni, "Artificial neural network models for indoor temperature prediction: investigations in two buildings," Neural Computing & Applications, vol. 16, pp. 81–89, 2007

[12] D. Meana-Llorián, C. G. García, B. C. G-bustelo, J. M. Lovelle, N. Garcia-Fernandez, "IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions," Future Generation Computer Systems, 2017 Nov 1, vol. 76, pp. 275-84.

[13] https://www.imonnit.com last accessed on May 8, 2020

[14] https://www.timeanddate.com/weather/usa/lafayette/historic?month=2&year=2020 last accessed on May 8, 2020

[15] https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/

[16] https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

[17] https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

[18] https://medium.com/pursuitnotes/multiple-linear-regression-model-in-7-steps-with-python-c6f40c0a527

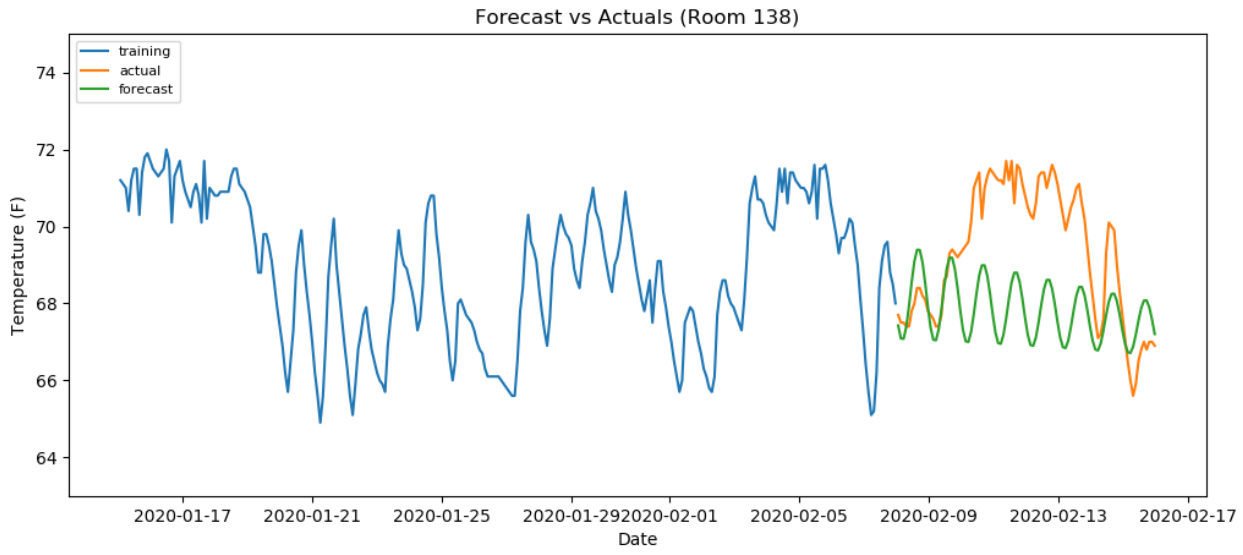
[19] https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/#

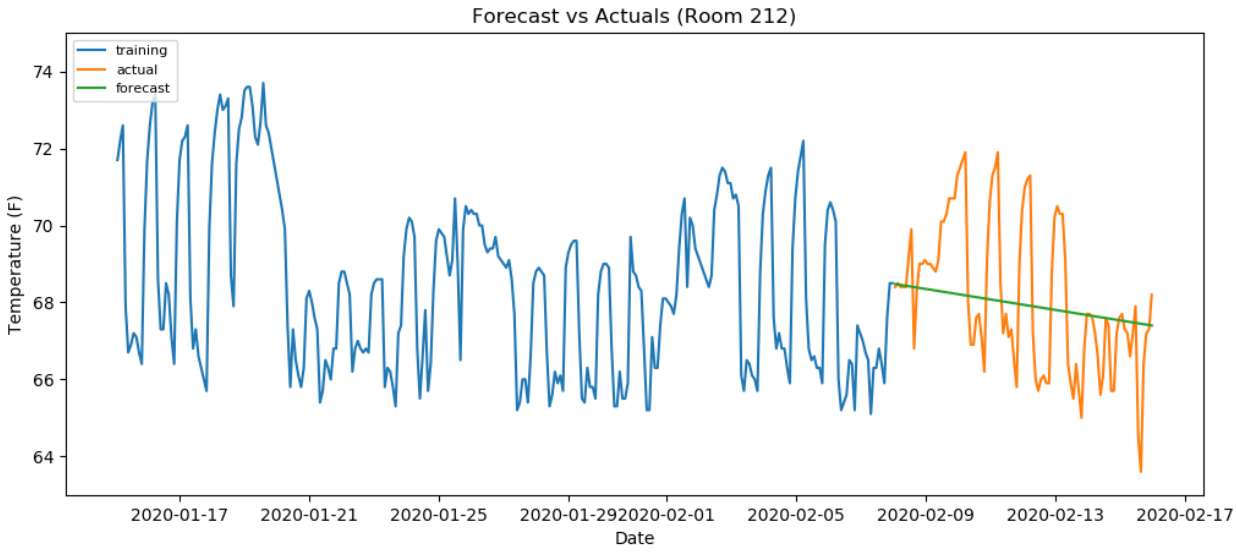
[20] https://towardsdatascience.com/time-series-machine-learning-regression-framework-9ea33929009a

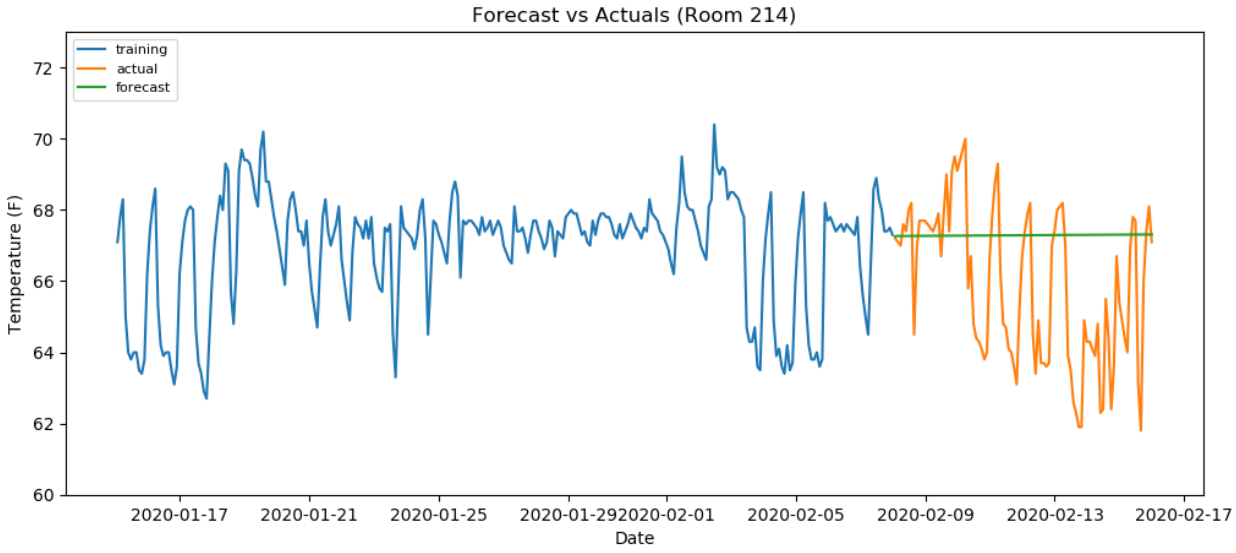
VI. APPENDIX

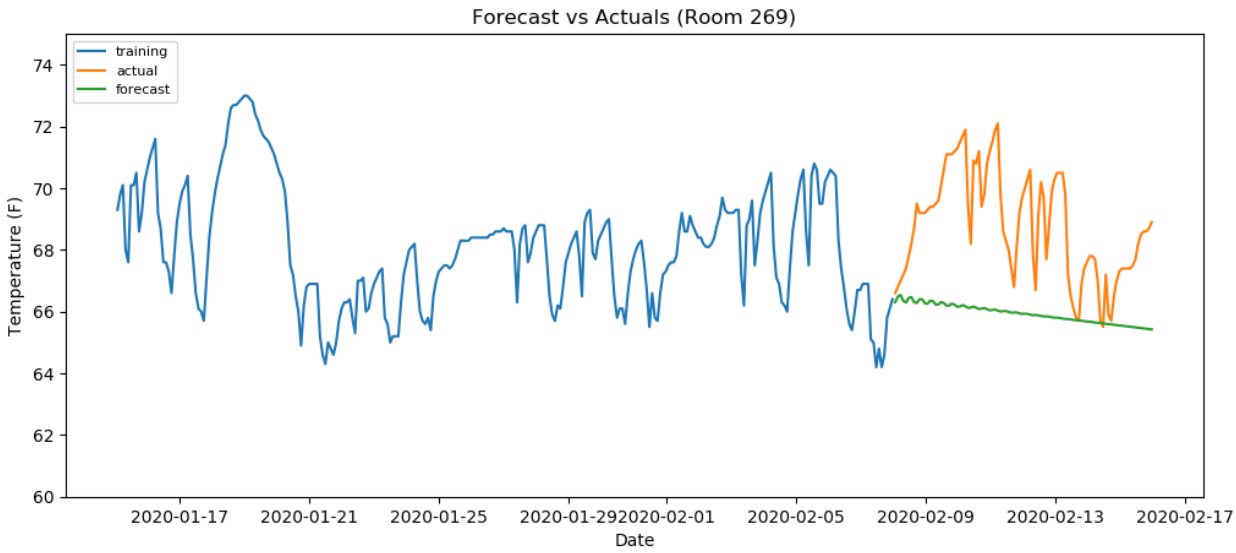
A. *Appendix A*

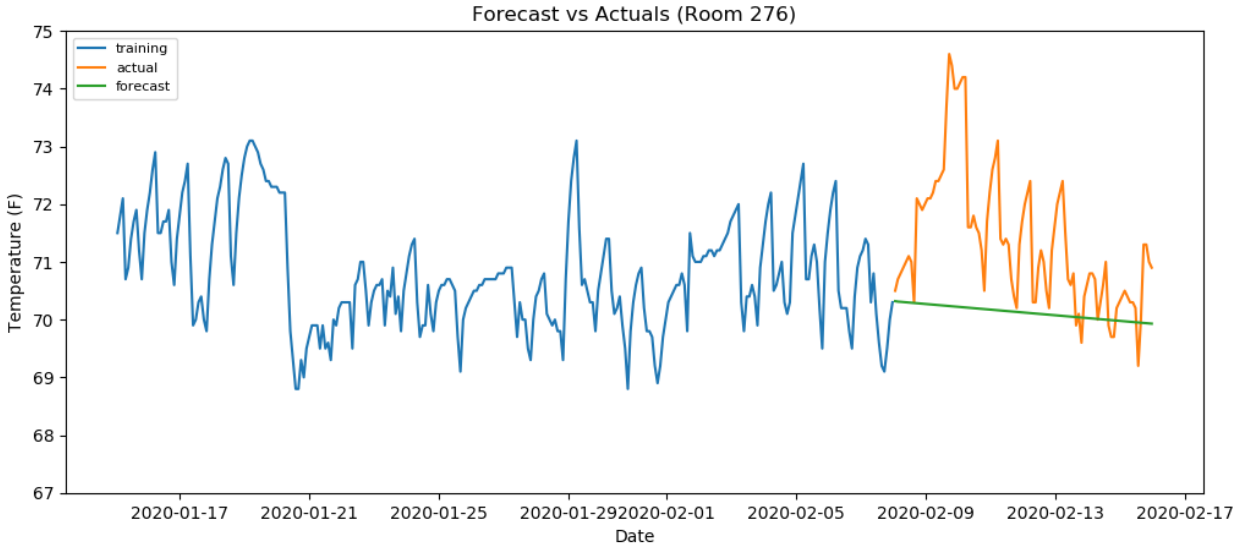
1) ARIMA



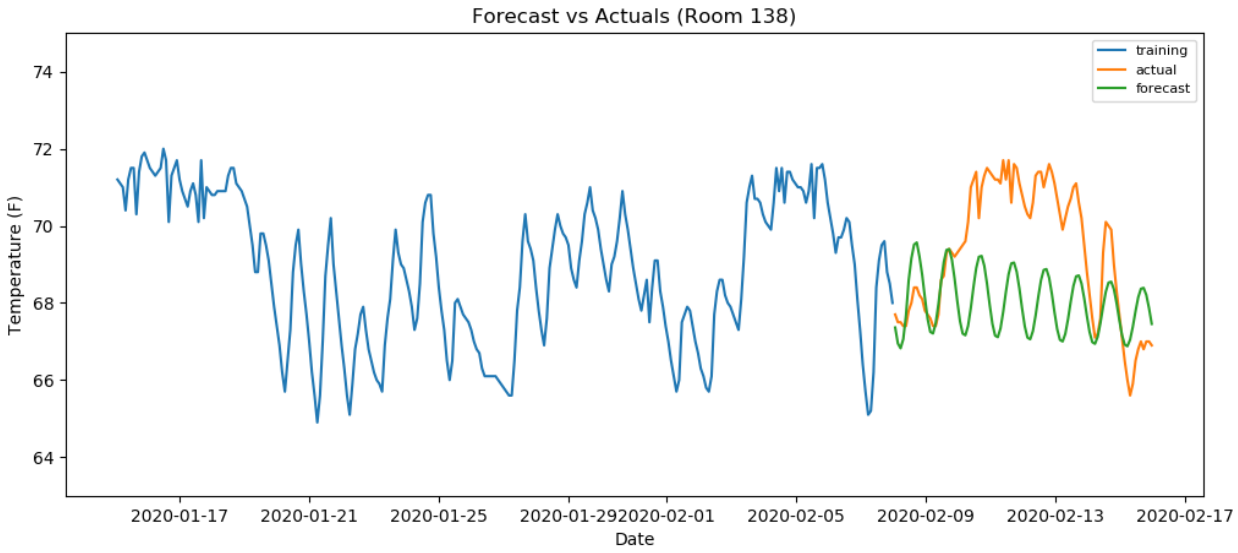

Forecast vs Actuals (Room 212)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-292020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

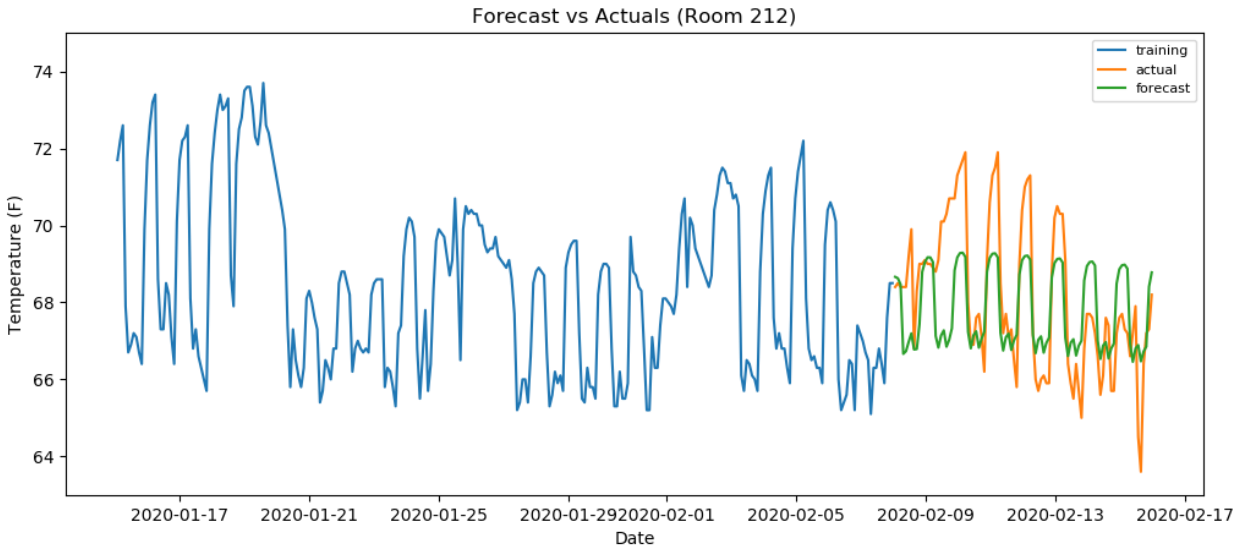

Forecast vs Actuals (Room 214)
training
actual
forecast
Temperature (F)
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-292020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

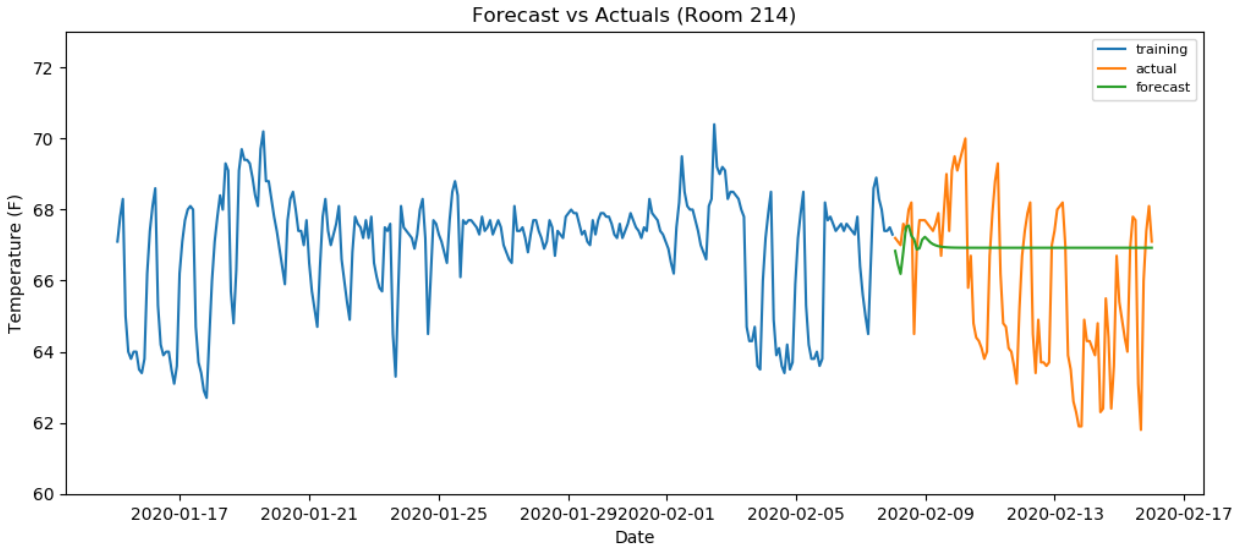

Forecast vs Actuals (Room 269)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-292020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

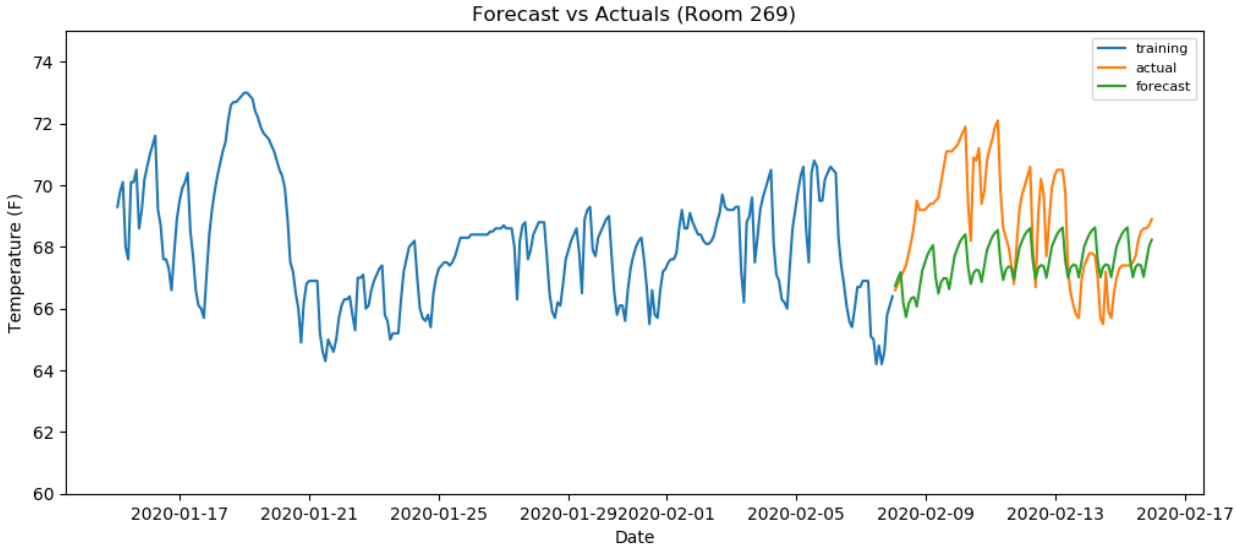

Forecast vs Actuals (Room 276)
training
actual
forecast
Temperature (F)
Date
75
74
73
72
71
70
69
68
67
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17

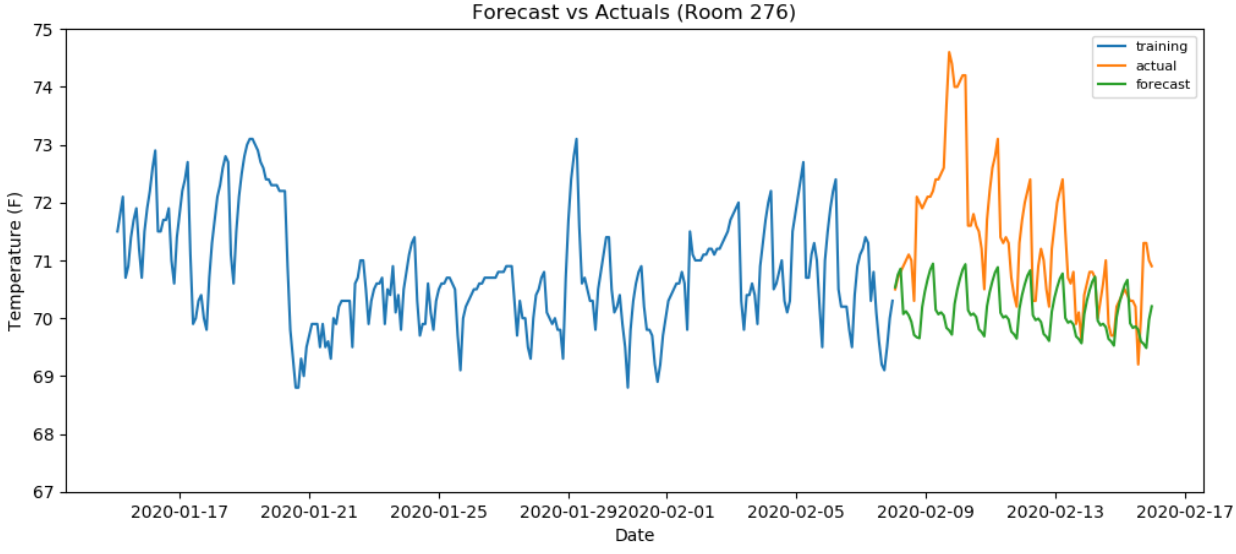
2) SARIMA


Forecast vs Actuals (Room 138)
training
actual
forecast
Temperature (F)
Date
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17

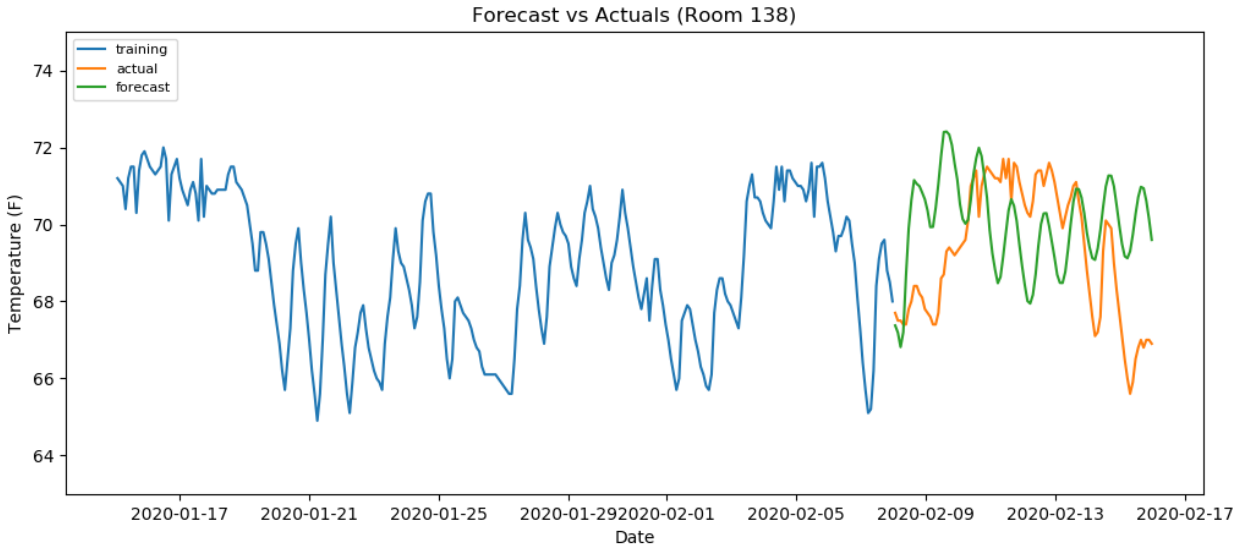

Forecast vs Actuals (Room 212)
training
actual
forecast
Temperature (F)
Date
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17

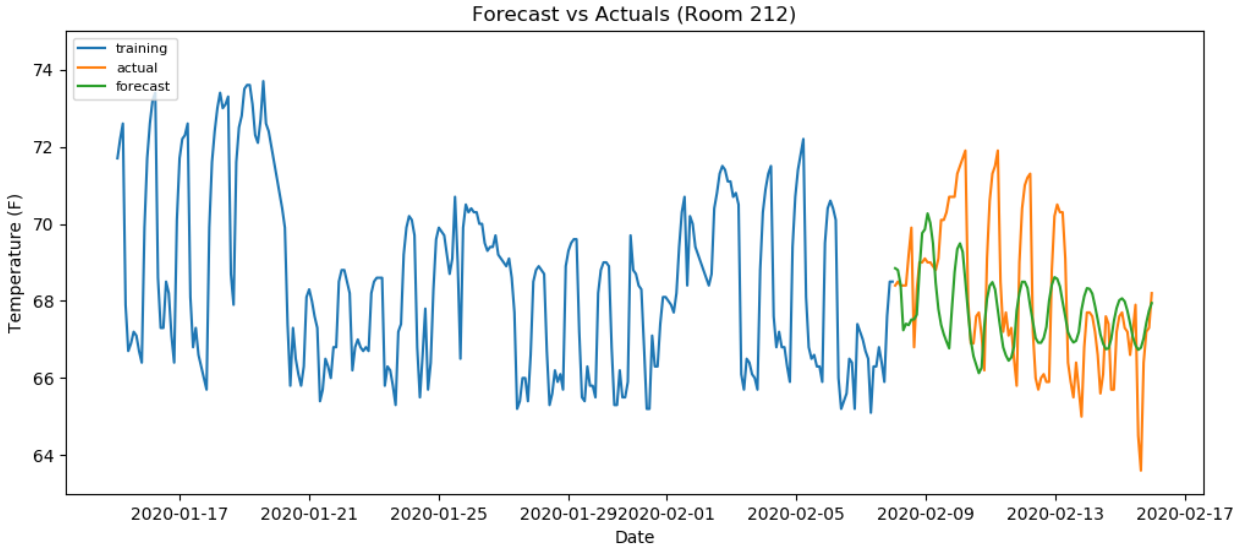

Forecast vs Actuals (Room 214)
training
actual
forecast
Temperature (F)
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

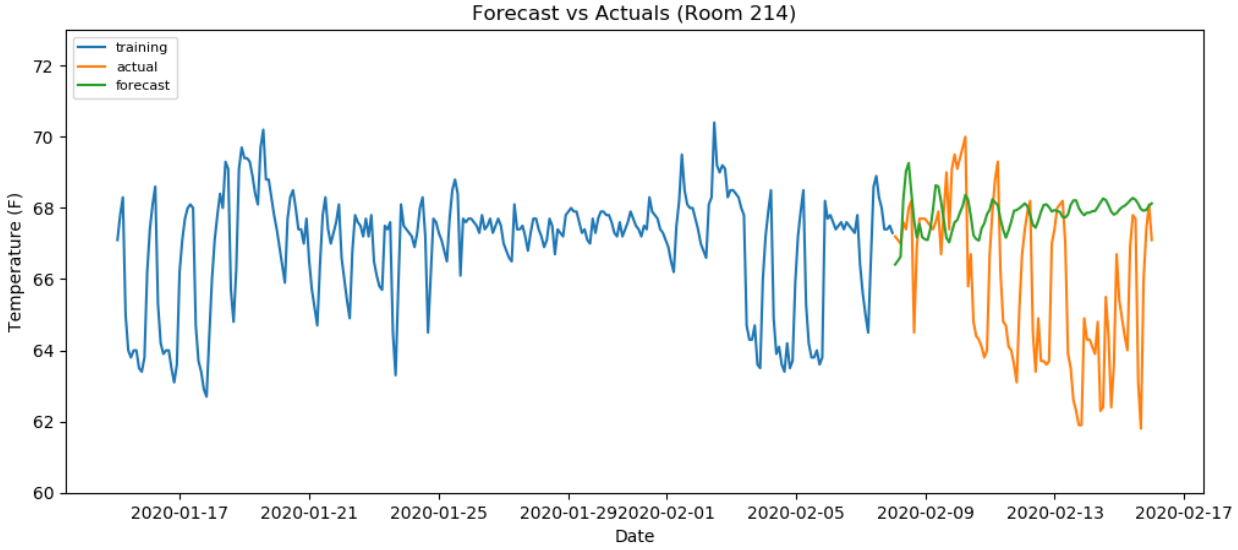

Forecast vs Actuals (Room 269)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

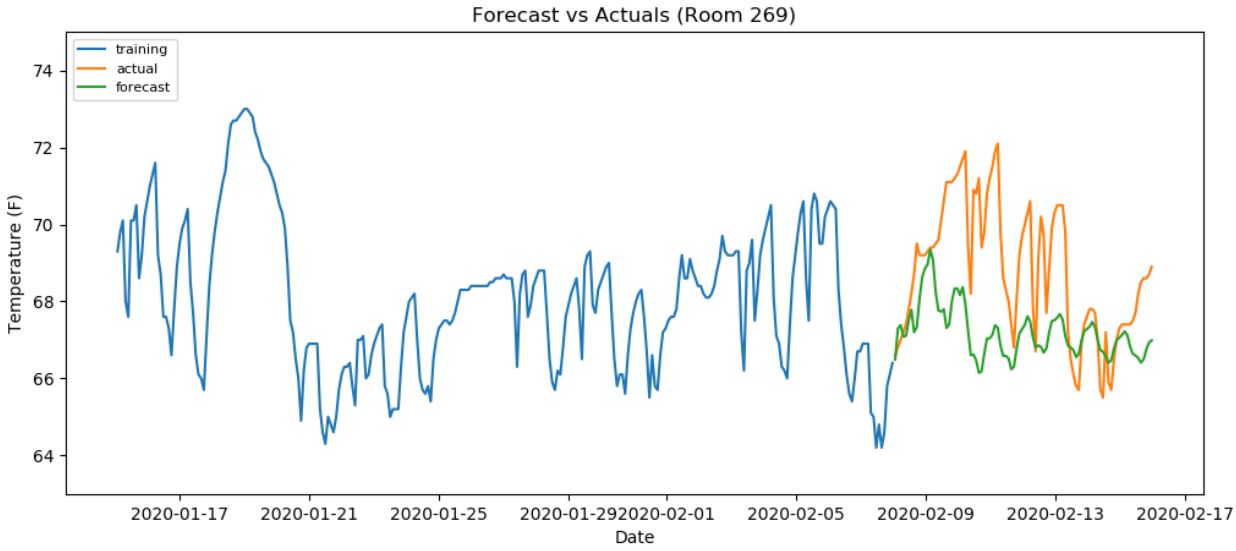

Forecast vs Actuals (Room 276)
training
actual
forecast
Temperature (F)
75
74
73
72
71
70
69
68
67
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

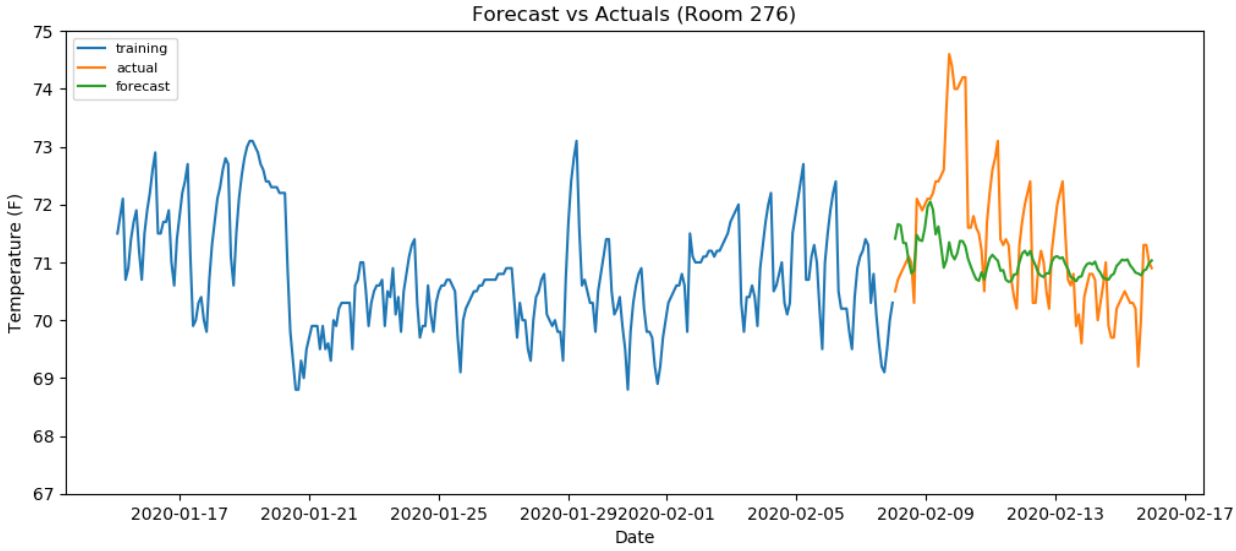
3) VAR


Forecast vs Actuals (Room 138)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

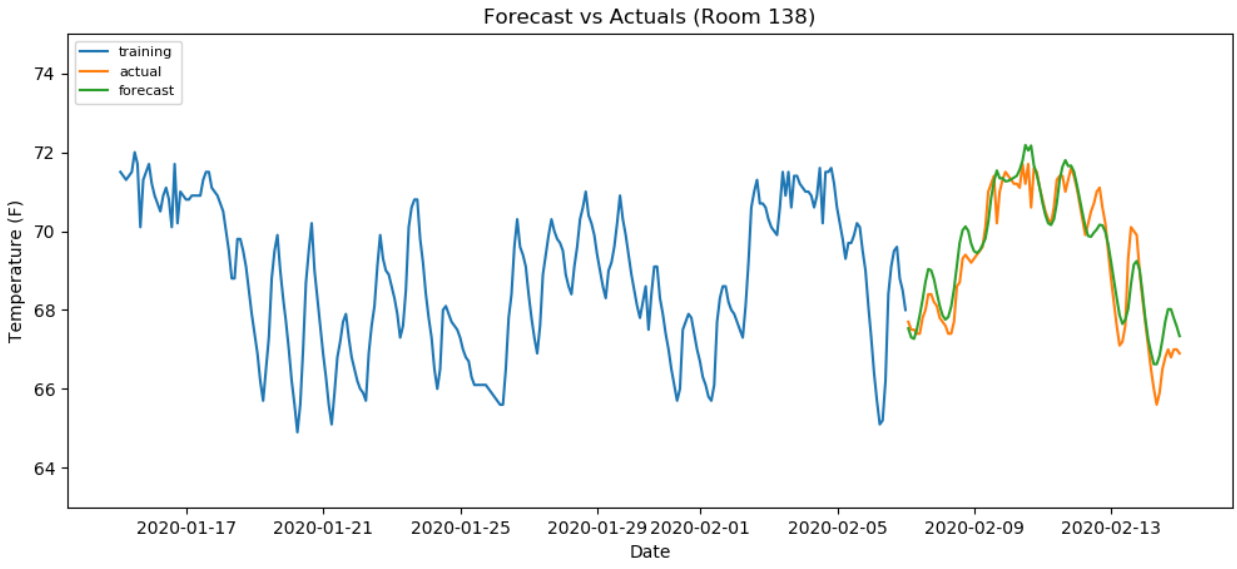

Forecast vs Actuals (Room 212)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

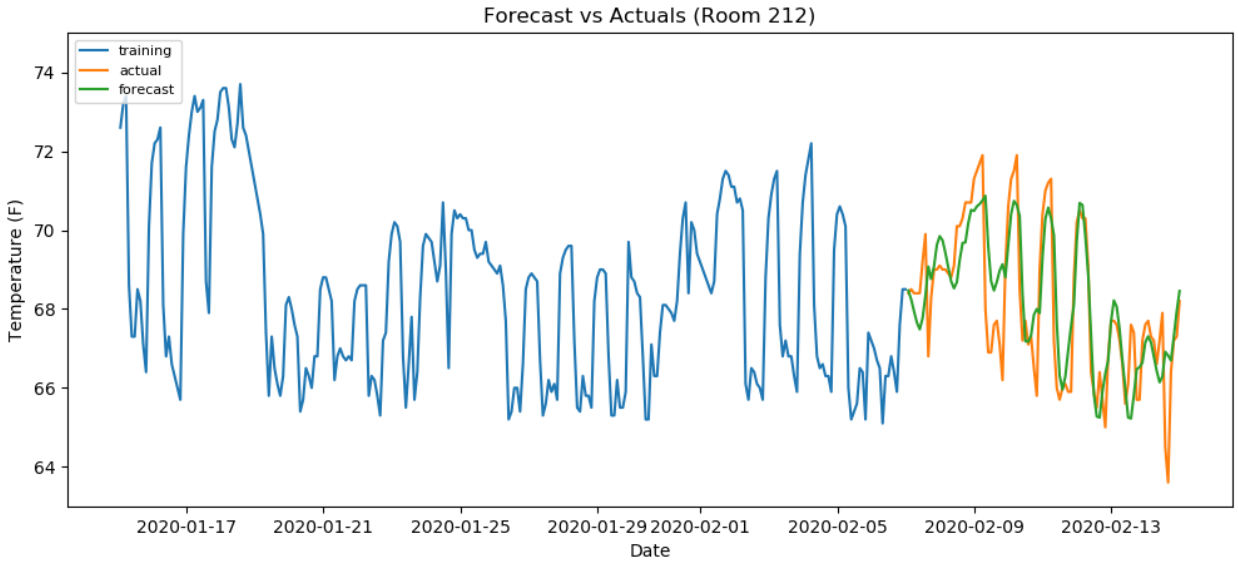

Forecast vs Actuals (Room 214)
training
actual
forecast
Temperature (F)
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

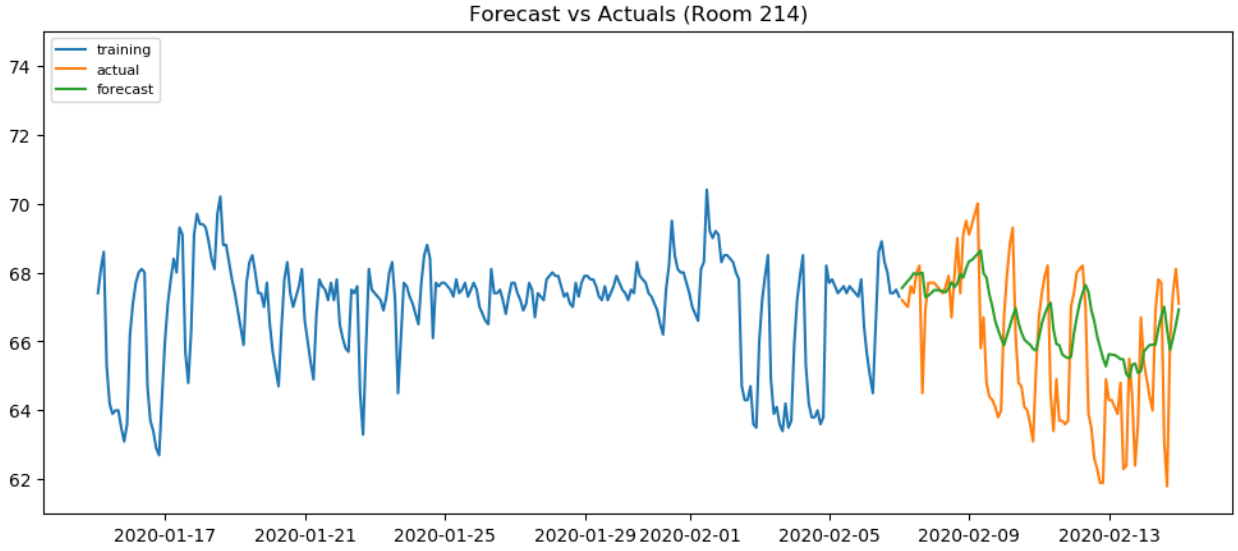

Forecast vs Actuals (Room 269)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

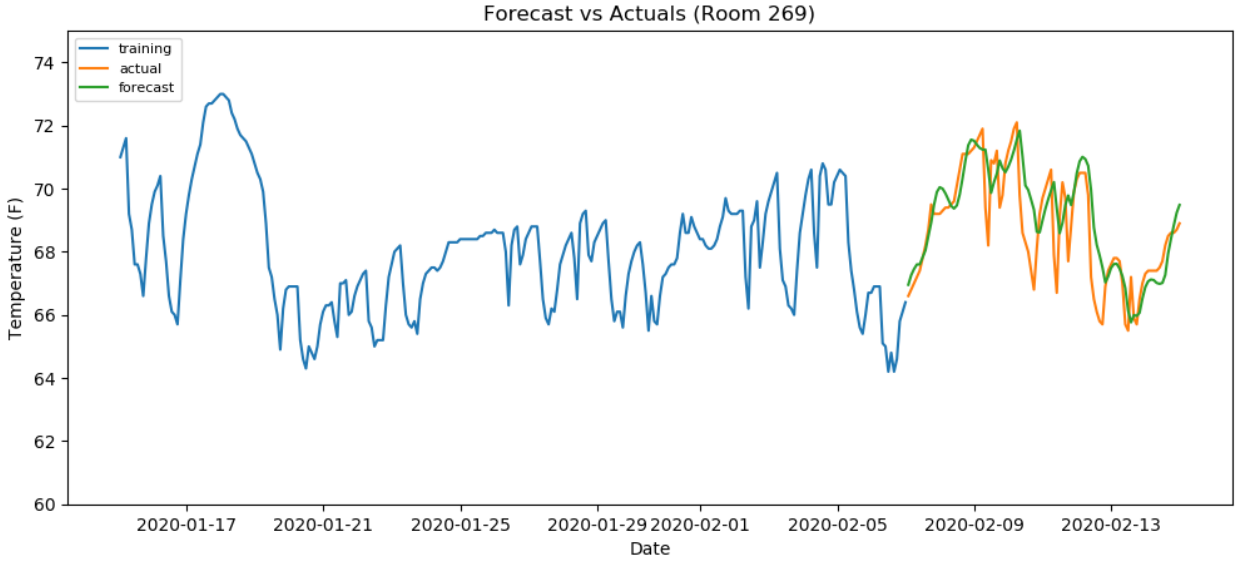

Forecast vs Actuals (Room 276)
training
actual
forecast
Temperature (F)
75
74
73
72
71
70
69
68
67
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
2020-02-17
Date

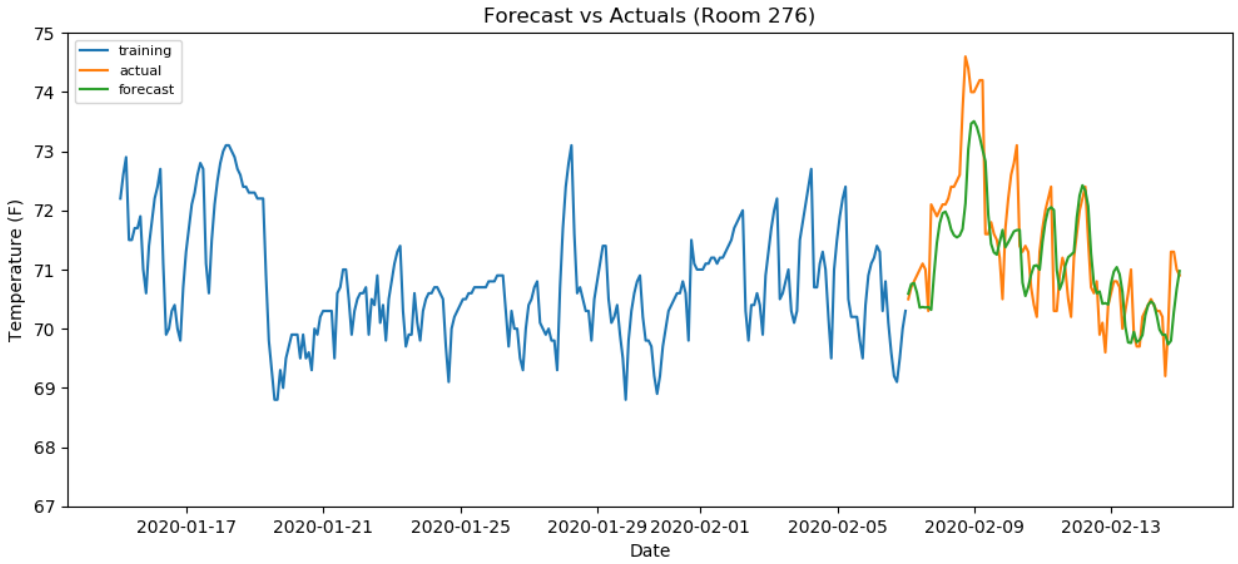
4) LSTM


Forecast vs Actuals (Room 138)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date

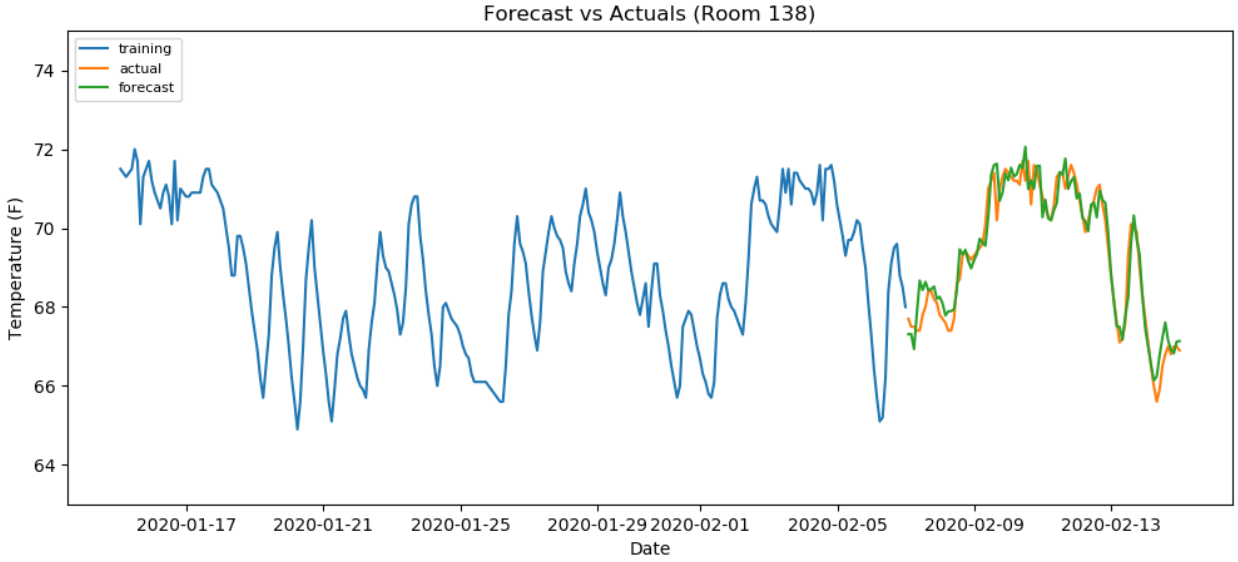

Forecast vs Actuals (Room 212)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date

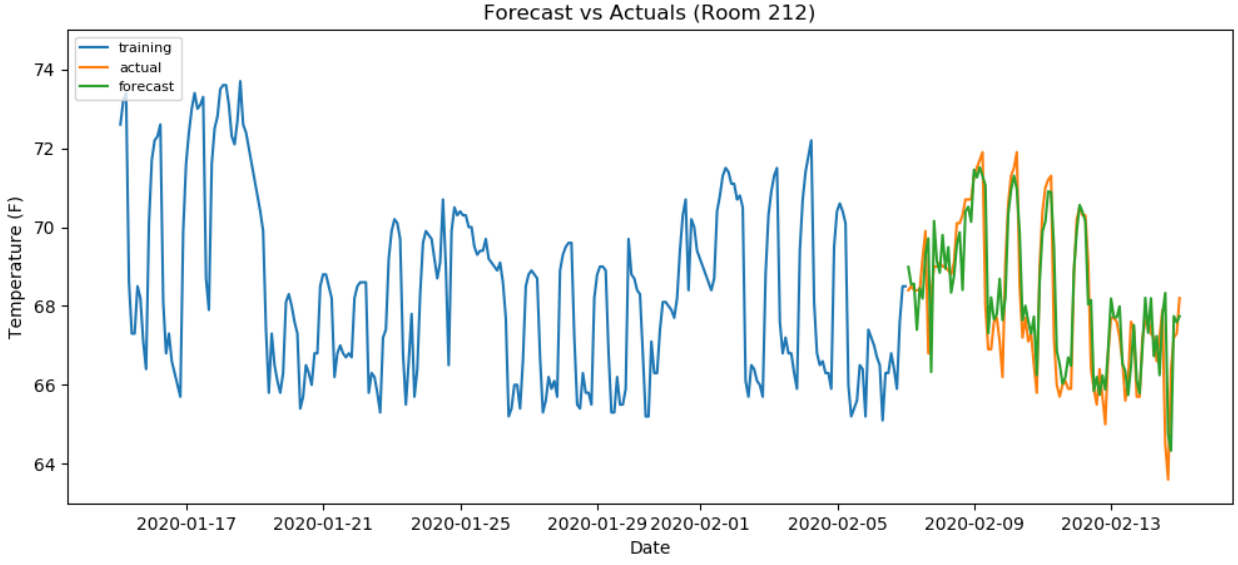

Forecast vs Actuals (Room 214)
training
actual
forecast
74
72
70
68
66
64
62
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13

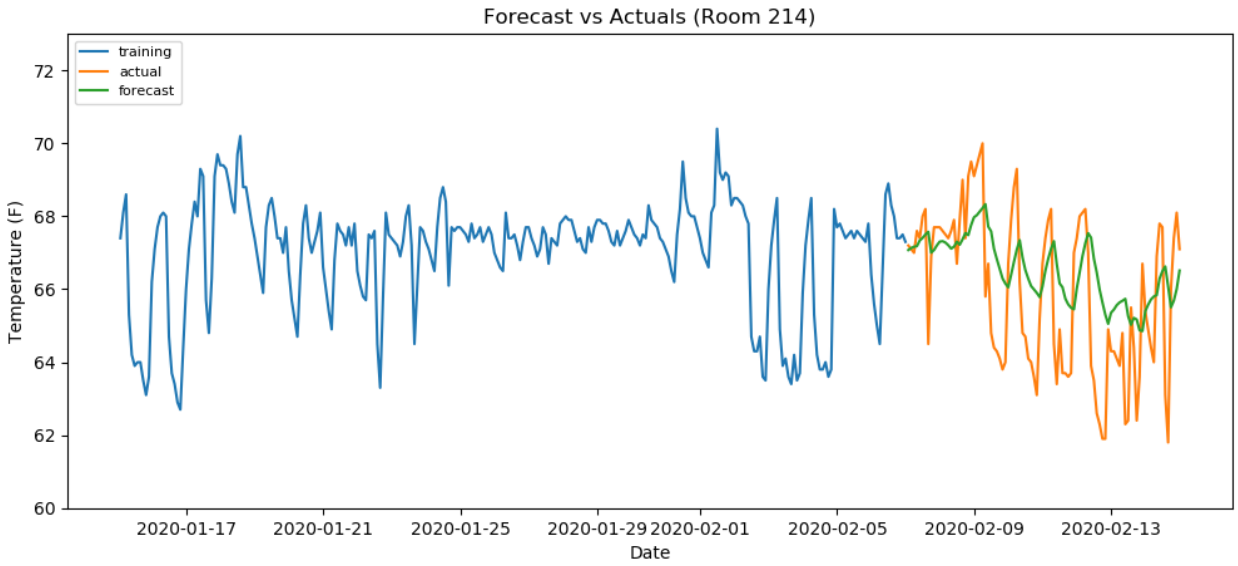

Forecast vs Actuals (Room 269)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date

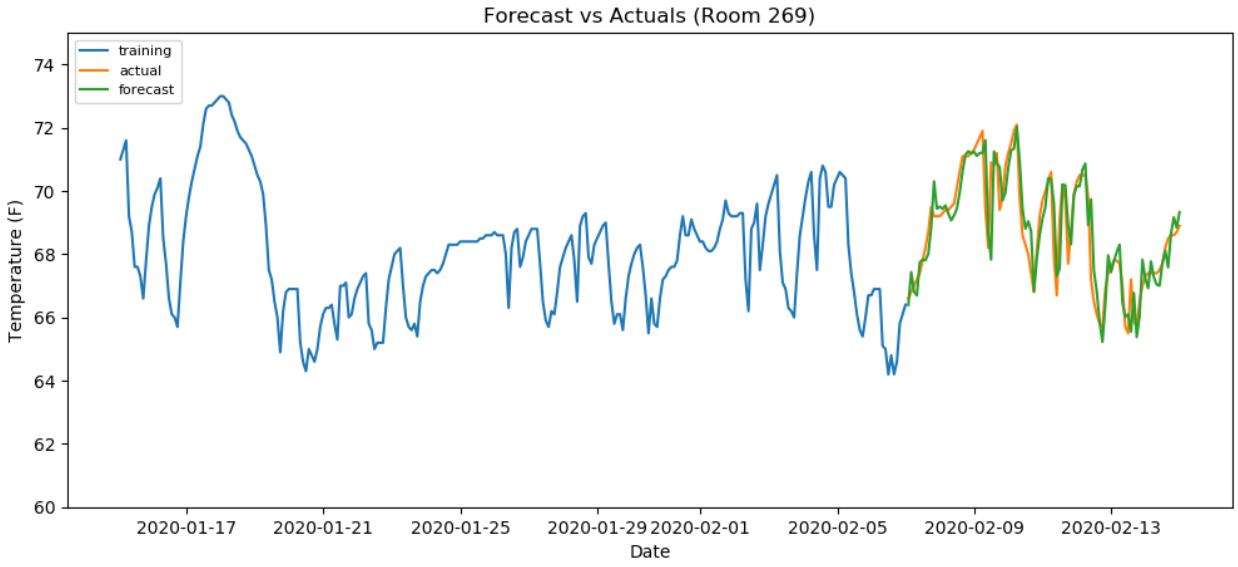


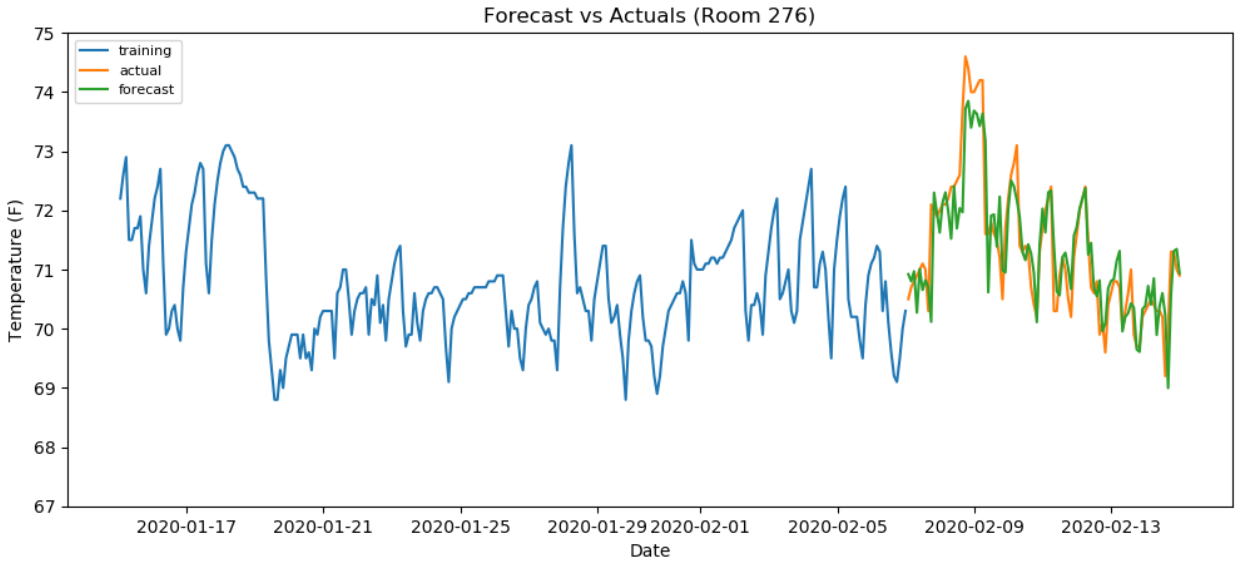
5) Multiple Linear Regression






Forecast vs Actuals (Room 214)
training
actual
forecast
Temperature (F)
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date


Forecast vs Actuals (Room 269)
training
actual
forecast
Temperature (F)
74
72
70
68
66
64
62
60
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date


Forecast vs Actuals (Room 276)
training
actual
forecast
Temperature (F)
75
74
73
72
71
70
69
68
67
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
2020-02-05
2020-02-09
2020-02-13
Date

B. *Appendix B*

1) ARIMA model python code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA
#Import dataset
df=pd.read_csv("132_Temp.csv")
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')
df.dropna(inplace=True)
#Define time series
ts=df.loc['1/15/2020':'2/15/2020']['Temperature']
# Create Training and Test
train = ts[:288]
test = ts[288:]
#Build function to test stationarity
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(6).mean()
    rolstd = timeseries.rolling(6).std()
    #Plot rolling statistics:
    plt.figure(figsize=(12,5), dpi=100)
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.ylabel('dT (F)')
    plt.show(block=False)
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
#Test for stationarity
test_stationarity(ts)
#Perform differencing
ts_diff=ts-ts.shift()
plt.plot(ts_diff)
ts_diff.dropna(inplace=True)
#Test stationarity again
test_stationarity(ts_diff)
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf
lag_acf = acf(ts_diff, nlags=6)
lag_pacf = pacf(ts_diff, nlags=6, method='ols')
#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
```

```
plt.axhline(y=-1.96/np.sqrt(len(ts_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
# ARIMA Model
model = ARIMA(train, order=(1, 1, 1))
fitted = model.fit(disp=-1)
# Forecast
fc, se, conf = fitted.forecast(96, alpha=0.05)  # 95% conf
# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)
# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
#plt.fill_between(lower_series.index, lower_series, upper_series,color='k', alpha=.15)
plt.title('Forecast vs Actuals (Room 132)')
plt.xlabel('Date')
plt.ylabel('Temperature (F)')
plt.legend(loc='upper left', fontsize=8)
plt.ylim(63,75)
plt.show()
print('RMSE: %.4f'% np.sqrt(sum((test.values-fc_series)**2)/len(fc_series)))
#Perform decomposition to check for seasonility
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.figure(figsize=(12,5), dpi=100)
plt.subplot(411)
plt.plot(ts, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```

2) SARIMA model python code

```
import pmdarima as pm
from pmdarima.arima import auto_arima
model = pm.auto_arima(train, start_p=1, start_q=1, seasonal=True, m=12)
# make your forecasts
forecasts = model.predict(96)  # predict N steps into the future
# Visualize the forecasts (blue=train, green=forecasts)
plt.figure(figsize=(12,5), dpi=100)
plt.plot(ts.index[:288],train.values, label='training')
plt.plot(ts.index[288:],test.values, label='actual')
plt.plot(ts.index[288:],forecasts, label='forecast')
plt.title('Forecast vs Actuals (Room 132)')
plt.legend(loc='best', fontsize=8)
plt.ylim(63,75)
plt.xlabel('Date')
```

```
plt.ylabel('Temperature (F)')
plt.show()
print('RMSE: %.4f'% np.sqrt(sum((test.values-forecasts)**2)/len(forecasts)))
```

3) VAR model python code

```
import pandas as pd
import numpy as np
from matplotlib import pyplot
import matplotlib.pyplot as plt
# Import Statsmodels
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic
#Import dataset
df=pd.read_csv("132_temp.csv")
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')
df.dropna(inplace=True)
#Define time series
ts=df.loc['1/15/2020':'2/15/2020',['Temperature','Humidity','Outside Temperature','Temperature on Previous day','Temperature
2 days before']]
#Granger's casuality test
from statsmodels.tsa.stattools import grangercausalitytests
maxlag=12
test = 'ssr_chi2test'
def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
    """Check Granger Causality of all possible combinations of the Time series.
    The rows are the response variable, columns are predictors. The values in the table
    are the P-Values. P-Values lesser than the significance level (0.05), implies
    the Null Hypothesis that the coefficients of the corresponding past values is
    zero, that is, the X does not cause Y can be rejected.
    data      : pandas dataframe containing the time series variables
    variables : list containing names of the time series variables.
    """
    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
    for c in df.columns:
        for r in df.index:
            test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
            min_p_value = np.min(p_values)
            df.loc[r, c] = min_p_value
    df.columns = [var + '_x' for var in variables]
    df.index = [var + '_y' for var in variables]
    return df
grangers_causation_matrix(ts,ts.columns)
#Cointegration test
from statsmodels.tsa.vector_ar.vecm import coint_johansen
def cointegration_test(df, alpha=0.05):
    """Perform Johanson's Cointegration Test and Report Summary"""
    out = coint_johansen(df,-1,5)
    d = {'0.90':0, '0.95':1, '0.99':2}
    traces = out.lr1
    cvts = out.cvt[:, d[str(1-alpha)]]
    def adjust(val, length= 6): return str(val).ljust(length)
    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*20)
    for col, trace, cvt in zip(df.columns, traces, cvts):
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt, 8), ' =>  ' , trace > cvt)
cointegration_test(ts)
#Divide into train and test sets
nobs = 96
```

```
ts_train, ts_test = ts[0:-nobs], ts[-nobs:]
#Build checking stationarity function
def adfuller_test(series, signif=0.05, name='', verbose=False):
    """Perform ADFuller to test for Stationarity of given series and print report"""
    r = adfuller(series, autolag='AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']
    def adjust(val, length= 6): return str(val).ljust(length)
    # Print Summary
    print(f'    Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'*47)
    print(f' Null Hypothesis: Data has unit root. Non-Stationary.')
    print(f' Significance Level    = {signif}')
    print(f' Test Statistic        = {output["test_statistic"]}')
    print(f' No. Lags Chosen       = {output["n_lags"]}')
    for key,val in r[4].items():
        print(f' Critical value {adjust(key)} = {round(val, 3)}')
    if p_value <= signif:
        print(f" => P-Value = {p_value}. Rejecting Null Hypothesis.")
        print(f" => Series is Stationary.")
    else:
        print(f" => P-Value = {p_value}. Weak evidence to reject the Null Hypothesis.")
        print(f" => Series is Non-Stationary.")
for name, column in ts_train.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')
#Perform differencing
ts_differenced = ts_train.diff().dropna()
for name, column in ts_differenced.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')
#Select order of VAR model
model = VAR(ts_differenced)
x = model.select_order(maxlags=20)
x.summary()
#Fit model
model_fitted = model.fit(14)
model_fitted.summary()
#Check for serial correlation of residuals
from statsmodels.stats.stattools import durbin_watson
out = durbin_watson(model_fitted.resid)
for col, val in zip(ts.columns, out):
    print(col, ':', round(val, 2))
lag_order = model_fitted.k_ar
print(lag_order)
forecast_input = ts_differenced.values[-lag_order:]
forecast_input
#Forecast
fc = model_fitted.forecast(y=forecast_input, steps=nobs)
ts_forecast = pd.DataFrame(fc, index=ts.index[-nobs:], columns=ts.columns + '_1d')
#Invert back to the original scale
def invert_transformation(df_train, df_forecast, second_diff=False):
    """Revert back the differencing to get the forecast to original scale."""
    df_fc = df_forecast.copy()
    columns = df_train.columns
    for col in columns:
        # Roll back 2nd Diff
        if second_diff:
            df_fc[str(col)+'_1d'] = (df_train[col].iloc[-1]-df_train[col].iloc[-2]) + df_fc[str(col)+'_2d'].cumsum()
        # Roll back 1st Diff
        df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] + df_fc[str(col)+'_1d'].cumsum()
    return df_fc
ts_results = invert_transformation(ts_train,ts_forecast, second_diff=False)
```

```
ts_results.loc[:,['Temperature_forecast', 'Humidity_forecast', 'Outside Temperature_forecast','Temperature on Previous day_forecast','Temperature 2 days before_forecast']]
#Plot results
plt.figure(figsize=(12,5), dpi=100)
plt.plot(ts_train['Temperature'], label='training')
plt.plot(ts_test['Temperature'], label='actual')
plt.plot(ts_results['Temperature_forecast'], label='forecast')
plt.title('Forecast vs Actuals (Room 132)')
plt.legend(loc='upper left', fontsize=8)
plt.xlabel('Date')
plt.ylabel('Temperature (F)')
plt.ylim(63,75)
plt.show()
#Check forecast accuracy
from statsmodels.tsa.stattools import acf
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)             # ME
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]   # corr
    mins = np.amin(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)             # minmax
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'corr':corr, 'minmax':minmax})
print('Forecast Accuracy of: Temperature')
accuracy_prod = forecast_accuracy(ts_results['Temperature_forecast'].values, ts_test['Temperature'])
for k, v in accuracy_prod.items():
    print(k, ': ', round(v,4))
```

4) LSTM & MLR python code

```
from math import sqrt
from numpy import concatenate
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import pyplot
import pandas as pd
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
#Import dataset
df=pd.read_csv("132_Temp.csv")
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')
df.dropna(inplace=True)
#Define time series
ts=df.loc['1/15/2020':'2/15/2020',['Temperature','Humidity','Outside Temperature','Temperature on Previous day','Temperature 2 days before']]
values=ts.values
#Apply scaling
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
```

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg
reframed = series_to_supervised(scaled, 12, 1)
# drop columns that is not needed to predict
reframed.drop(reframed.columns[[61,62,63,64]], axis=1, inplace=True)
# split into train and test sets
values = reframed.values
n_train = 276
train = values[:n_train, :]
test = values[n_train:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
print(train_X.shape,test_X.shape, train_y.shape,test_y.shape)
# specify the number of lag hours
n_hours = 12
n_features = 5
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print(train_X.shape,test_X.shape, train_y.shape,test_y.shape)
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=30, batch_size=32, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[1]*test_X.shape[2]))
# invert scaling for forecast
testy_forecast = np.zeros(shape=(len(test_y), 5))
testy_forecast[:,0] = yhat[:,0]
inv_yhat = scaler.inverse_transform(testy_forecast)[:,0]
# invert scaling for actual test
```

```
testy_actual = np.zeros(shape=(len(test_y), 5))
testy_actual[:,0] = test_y
inv_y = scaler.inverse_transform(testy_actual)[:,0]
# invert scaling for actual train
trainy_actual = np.zeros(shape=(len(train_y), 5))
trainy_actual[:,0] = train_y
inv_y_train = scaler.inverse_transform(trainy_actual)[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
#Plot results
plt.figure(figsize=(12,5), dpi=100)
plt.plot(ts.index[:276],inv_y_train, label='training')
plt.plot(ts.index[276:372],inv_y, label='actual')
plt.plot(ts.index[276:372],inv_yhat, label='forecast')
plt.title('Forecast vs Actuals (Room 132)')
plt.legend(loc='upper left', fontsize=8)
plt.ylim(63,75)
plt.xlabel('Date')
plt.ylabel('Temperature (F)')
plt.show()

# # Linear regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
train_X_lr, train_y_lr = train[:, :-1], train[:, -1]
test_X_lr, test_y_lr = test[:, :-1], test[:, -1]
regressor.fit(train_X_lr,train_y_lr)
coef=regressor.fit(train_X_lr,train_y_lr).coef_.reshape(5,12)
coef_df=pd.DataFrame(coef)
y_pred= regressor.predict(test_X_lr)
#Invert scaling for forecast
testy_forecast_lr = np.zeros(shape=(len(test_y_lr), 5))
testy_forecast_lr[:,0] = y_pred
inv_yhat_lr = scaler.inverse_transform(testy_forecast_lr)[:,0]
#Invert scaling for actual
testy_actual_lr = np.zeros(shape=(len(test_y_lr), 5))
testy_actual_lr[:,0] = test_y_lr
inv_y_lr = scaler.inverse_transform(testy_actual_lr)[:,0]
#Plot results
plt.figure(figsize=(12,5), dpi=100)
plt.plot(ts.index[:276],inv_y_train, label='training')
plt.plot(ts.index[276:372],inv_y_lr, label='actual')
plt.plot(ts.index[276:372],inv_yhat_lr, label='forecast')
#plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.15)
plt.title('Forecast vs Actuals (Room 132)')
plt.legend(loc='upper left', fontsize=8)
plt.ylim(63,75)
plt.xlabel('Date')
plt.ylabel('Temperature (F)')
plt.show()
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y_lr, inv_yhat_lr))
print('Test RMSE: %.3f' % rmse)
```