Estimating Room Occupancy: A Comparative Study of ARMA, Multivariate Regression, and LSTM Models

CS5526: Data Analytics II
Virginia Polytechnic Institute & State University
Jonathan Samuel

Abstract

This report presents a comprehensive analysis of the "Room Occupancy Estimation Data Set" from UCSD, which was collected over a four-day period using multiple non-intrusive environmental sensors. The dataset was pre-processed and divided into training and testing entries, with a focus on discrete room occupancy data ranging from 0 to 3 people. Various time series models, including the Holt-Winter Method, baseline models, Multiple Linear Regression, ARMA, and LSTM, were implemented to assess their suitability for predicting room occupancy. Among these models, the ARMA(1,0) and LSTM models demonstrated superior performance in terms of parameter significance, residual analysis, and predictive accuracy. The ARMA(1,0) model was supported by the GPAC, ACF, and PACF analyses, while the LSTM model exhibited a notably low mean squared error (MSE). This study concludes that the ARMA(1,0) and LSTM models offer the most accurate representation of the dataset, considering the available data and the discrete nature of room occupancy.

Table of Contents

Table of Figures

Table of Tables

Introduction

Time series analysis and modeling play a crucial role in understanding and predicting the behavior of sequential data over time. It allows us to capture underlying patterns, trends, and seasonal components, while accounting for noise and irregularities within the data. This report presents a comprehensive examination of the "Room Occupancy Estimation Data Set" from UCSD, which utilizes environmental sensors such as temperature, light, sound, CO_2, and PIR to estimate the number of occupants in a room. The goal is to develop an accurate and reliable model capable of predicting room occupancy based on the given dataset.

The report is structured as follows: first, I explore the dataset's characteristics and perform necessary pre-processing steps to ensure data quality and consistency. Next, I assess the stationarity of the dataset, as well as decompose the time series data into its trend and seasonal components. I then implement several time series forecasting baseline models, such as the Holt-Winter method, Average and Multiple Linear Regression, to establish a foundation for comparison.

Feature engineering techniques, including Single Value Decomposition (SVD) and Backward Stepwise Regression, are employed to improve the dataset's model-readiness prior to multiple linear regression training. I then dive into more advanced time series models, such as ARMA, and leverage the Generalized Partial Autocorrelation Function (GPAC) to determine the optimal order. Parameter estimation using the Levenberg Marquardt algorithm (LMA) and diagnostic analysis are performed to assess the quality and performance of the ARMA model.

Finally, I explore the Long Short-Term Memory (LSTM) model, a deep learning approach designed to capture long-range dependencies within the time series data. I evaluate the performance of the LSTM model and compare it to the previously implemented models. In conclusion, I discuss the most suitable models for the given dataset and provide recommendations for potential real-world applications.

Dataset

The dataset, titled "Room Occupancy Estimation Data Set," has been sourced from UCSD and is designed for estimating the precise number of occupants in a room utilizing multiple non-intrusive environmental sensors such as temperature, light, sound, CO_2, and PIR. The data was systematically collected over a four-day period, with room occupancy varying between 0 and 3 people. The ground truth for the occupancy count was recorded manually. Figure 1 presents the fields in the dataset.

Comprising approximately 10,000 entries and 19 fields, the dataset includes the time and dependent field (time and room_occupancy_count, respectively). Pre-processing involves focusing on December 22, 23, and 24 due to the gap between these dates and the remaining data. Samples in the dataset are measured at an approximate 30-second interval (±1 second). To normalize the frequency, the time was replaced with a fixed 30-second interval. Following pre-processing, the dataset contains approximately 5,305 entries, which are divided into 4,244 training entries and 1,061 testing entries. Figures 2 and 3 provide basic statistics for the numerical fields within the dataset, while Figure 4 displays the complete room_occupancy_count versus time plot.

Contrasting with other datasets covered in this course that focus on continuous data and a large number of entries, this dataset features discrete data (0,1,2,3 – representing the number of people in a room) and a limited number of entries. The ACF plot in Figure 5 reveals a slight decrease in the dependent variable with respect to increased lags. The PACF indicates the significance of lag 1, which drops immediately after lag 1, suggesting a potential AR order of 1. Figure 6 highlights a high correlation among all temperature and light probes, with CO_2 also showing a correlation with temperature probes. Sound and PIR do not exhibit a strong correlation with other fields.

```
#   Column               Non-Null Count  Dtype
---  ------               --------------  -----
0   time                 5305 non-null   datetime64[ns]
1   S1_Temp              5305 non-null   float64
2   S2_Temp              5305 non-null   float64
3   S3_Temp              5305 non-null   float64
4   S4_Temp              5305 non-null   float64
5   S1_Light             5305 non-null   int64
6   S2_Light             5305 non-null   int64
7   S3_Light             5305 non-null   int64
8   S4_Light             5305 non-null   int64
9   S1_Sound             5305 non-null   float64
10  S2_Sound             5305 non-null   float64
11  S3_Sound             5305 non-null   float64
12  S4_Sound             5305 non-null   float64
13  S5_CO2               5305 non-null   int64
14  S5_CO2_Slope         5305 non-null   float64
15  S6_PIR               5305 non-null   int64
16  S7_PIR               5305 non-null   int64
17  Room_Occupancy_Count 5305 non-null   int64
```

```
time                    2017-12-22 10:49:41
S1_Temp                               24.94
S2_Temp                               24.75
S3_Temp                               24.56
S4_Temp                               25.38
S1_Light                                121
S2_Light                                 34
S3_Light                                 53
S4_Light                                 40
S1_Sound                               0.08
S2_Sound                               0.19
S3_Sound                               0.06
S4_Sound                               0.06
S5_CO2                                  390
S5_CO2_Slope                       0.769231
S6_PIR                                    0
S7_PIR                                    0
Room_Occupancy_Count                      1
```

Figure 1 Fields datatype, non-null count and example of values using the first entry

	S1_Temp	S2_Temp	S3_Temp	S4_Temp	S1_Light	S2_Light	S3_Light	S4_Light	S1_Sound	S2_Sound	S3_Sound
count	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000	5305.000000
mean	25.569955	25.717084	25.130224	25.855016	42.912912	41.822809	50.087276	15.815269	0.235361	0.161957	0.233676
std	0.412754	0.728116	0.493852	0.359521	62.419314	83.962010	71.551968	21.233240	0.408521	0.335159	0.551491
min	24.940000	24.750000	24.440000	25.130000	0.000000	0.000000	0.000000	0.000000	0.060000	0.040000	0.050000
25%	25.190000	25.190000	24.690000	25.560000	0.000000	0.000000	0.000000	0.000000	0.070000	0.050000	0.060000
50%	25.440000	25.440000	25.000000	25.750000	0.000000	0.000000	6.000000	4.000000	0.070000	0.050000	0.060000
75%	25.940000	25.940000	25.560000	26.250000	117.000000	24.000000	71.000000	26.000000	0.120000	0.080000	0.090000
max	26.380000	29.000000	26.190000	26.560000	165.000000	258.000000	280.000000	74.000000	3.880000	3.440000	3.670000

Figure 2 Basic statistics of the first 11 numerical fields

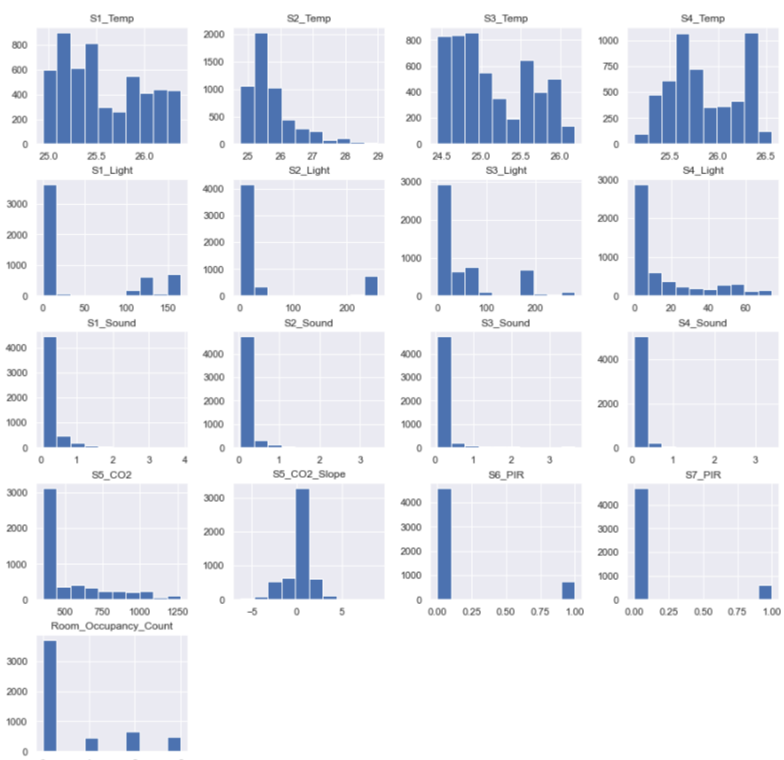


Figure 3 Distribution of each numerical field

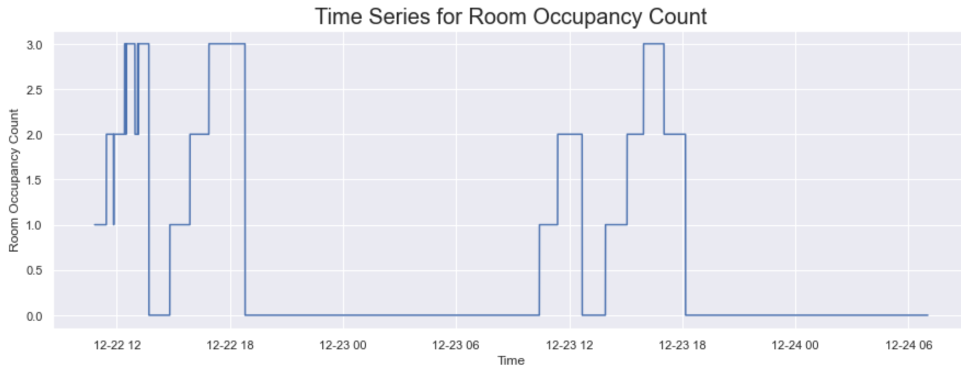


Figure 4 Plot of the time series dependent variable vs. time

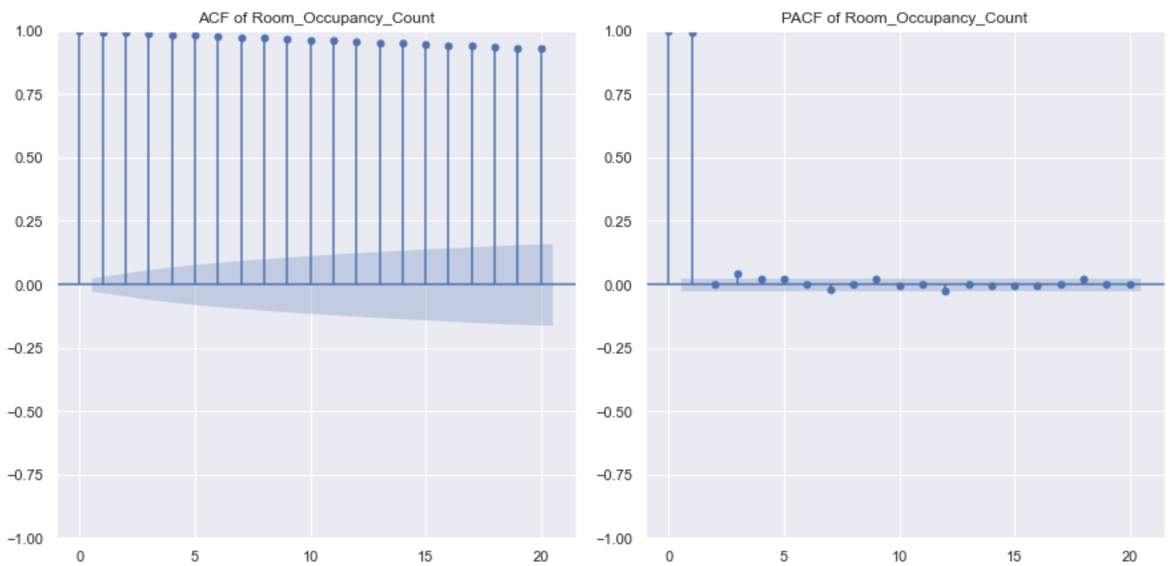


Figure 5 ACF and PACF of the dependent variable, Room_Occupancy_Count

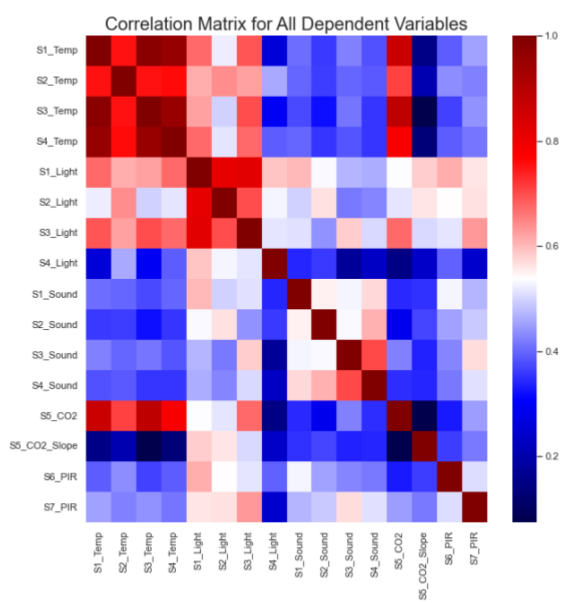


Figure 6 Correlation Matrix

Stationarity
The Augmented Dickey-Fuller (ADF) tests indicate that the data is stationary. However, the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test suggests non-stationarity in the data. It is worth noting that a similar issue was encountered in a previous assignment, and it was concluded that the ADF test is a more reliable measure for stationarity, as it is commonly utilized in various sources. Figure 8 presents the results of the rolling mean and variance, both of which appear to be constant over time, further supporting the notion of stationarity.

```
ADF Test for Dependent Variable
p-value: 0.043495
Critical Values:
        1%: -3.432
        5%: -2.862
        10%: -2.567
Series is stationary


KPSS Test for Dependent Variable
Results of KPSS Test:
KPSS Statistic: 1.0699858109253737
p-value: 0.01
num lags: 40
Critical Values:
        10% : 0.347
        5% : 0.463
        2.5% : 0.574
        1% : 0.739
Series is not stationary
```

Figure 7 Results of the ADF and KPSS Test

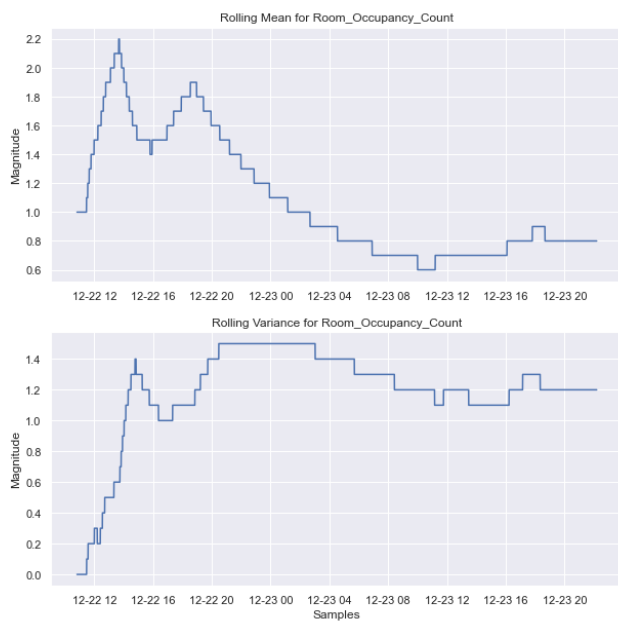


Figure 8 Room Occupancy Count Rolling Mean (Top) and Variance (Bottom)

Time Series Decomposition
Utilizing the Seasonal and Trend decomposition using Loess (STL) method to decompose the time series dataset, a distinct negative trend emerges as the room occupancy count progresses over time. The seasonal component aligns with the original dataset, indicating a strong seasonal influence. Notably, both the trend and seasonal components exhibit approximately 100% strength. Figure 10 represents the original, seasonally adjusted, and detrended datasets in graphical form.

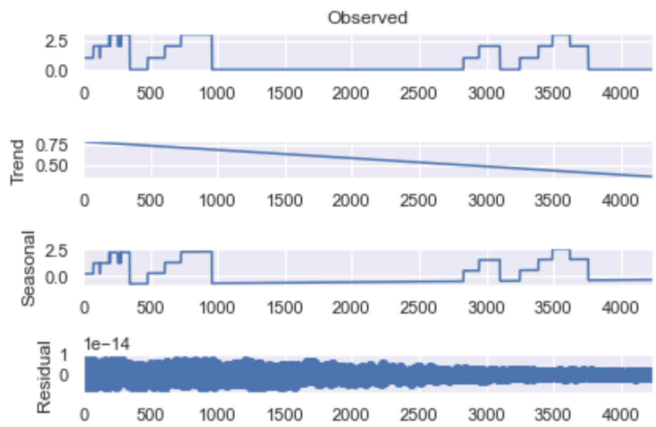


Figure 9 Decomposed Time Series Data using STL

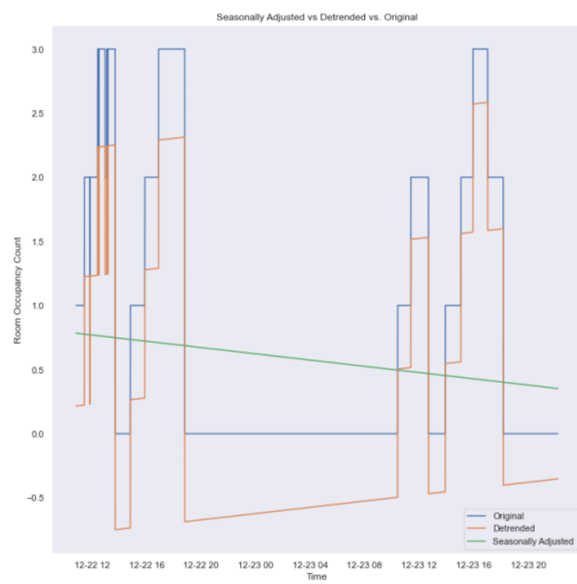


Figure 10 Seasonally Adjusted, Detrended and Original Time Series Data

Holt-Winter Method

Figure 11 presents the Holt-Winters method with additive trend and seasonal components. The predictions and forecasts for the training and testing datasets, respectively, closely align with the ground truth, suggesting that this method could be a strong contender for the final model to forecast the given datasets.

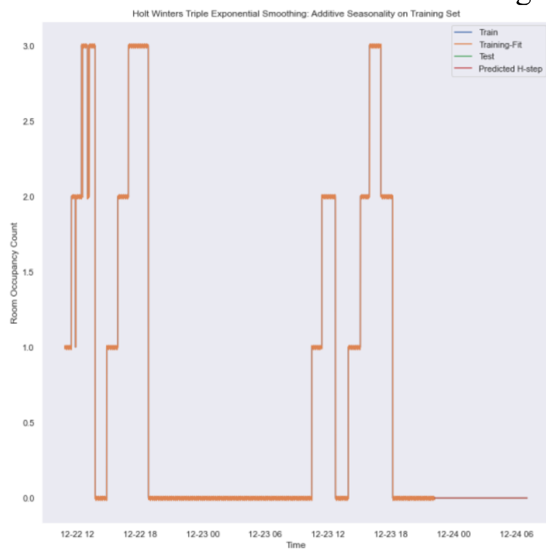
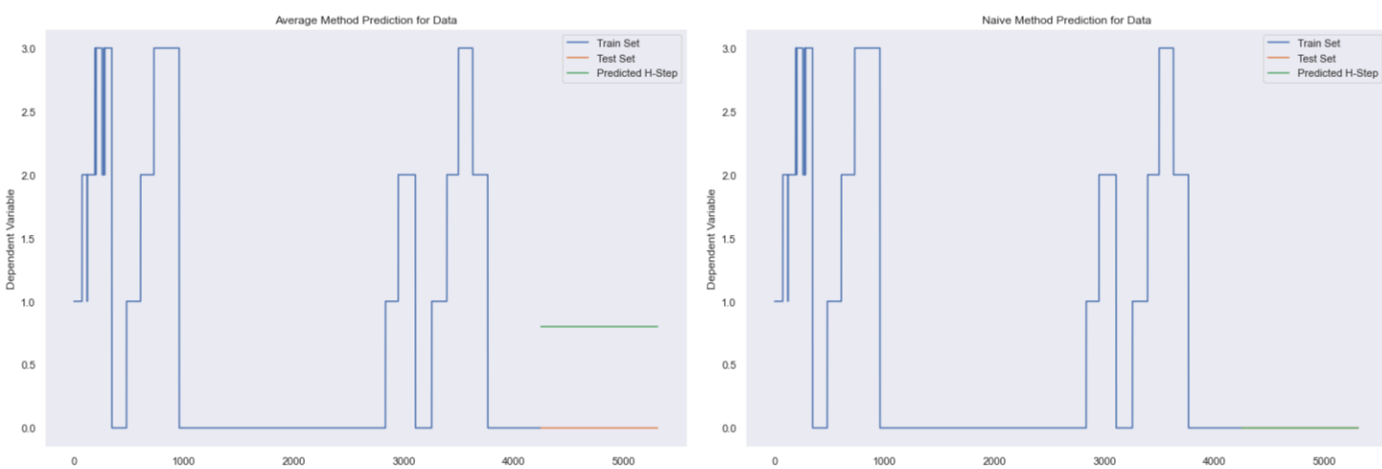


Figure 11 Holt-Winter Triple Exponential Smoothing with Additive Seasonality and Trend

Baseline Models

Figure 12 illustrates baseline forecasting using average, naïve, and drift methods. Intriguingly, the naïve method serves as an excellent predictor since the test set solely contains zeros for room occupancy count. While effective for the immediate future, this method will falter when additional occupants enter the room. Similarly, the average method is satisfactory but limited to a one-step prediction. The drift method, on the other hand, forecasts negative occupants. Figure 13 displays the ACF of the residuals for the same methods. Table 1 provides basic statistics concerning the residuals and forecasts, including Q values, MSE, and variance. It is unclear why the drift method exhibits comparable results to the naïve method, as its forecasting performance is inadequate, as demonstrated in Figure 12. Nevertheless, the results for the naïve and average methods are reasonable for the reasons previously discussed. Figure 14 shows the Simple Exponential Smoothing method applied for different alphas on the time series dataset. All with the exception of $\alpha = 0$, have models that forecast well. All of these models will be used as a baseline when assessing a final model for the given dataset.



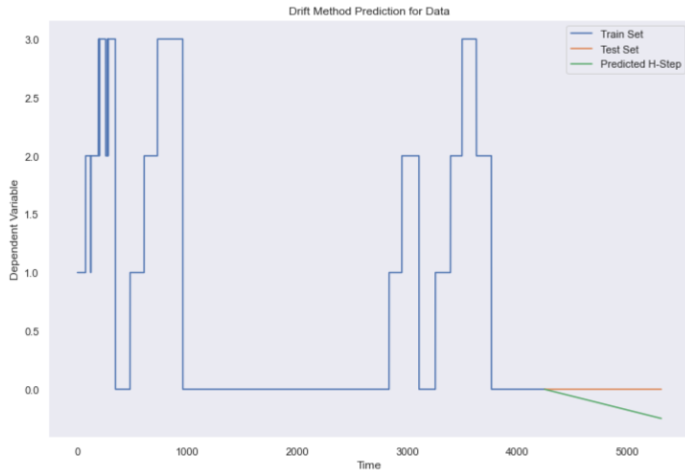


Figure 12 1-Step Forecasting for Average, Naive and Drift Methods

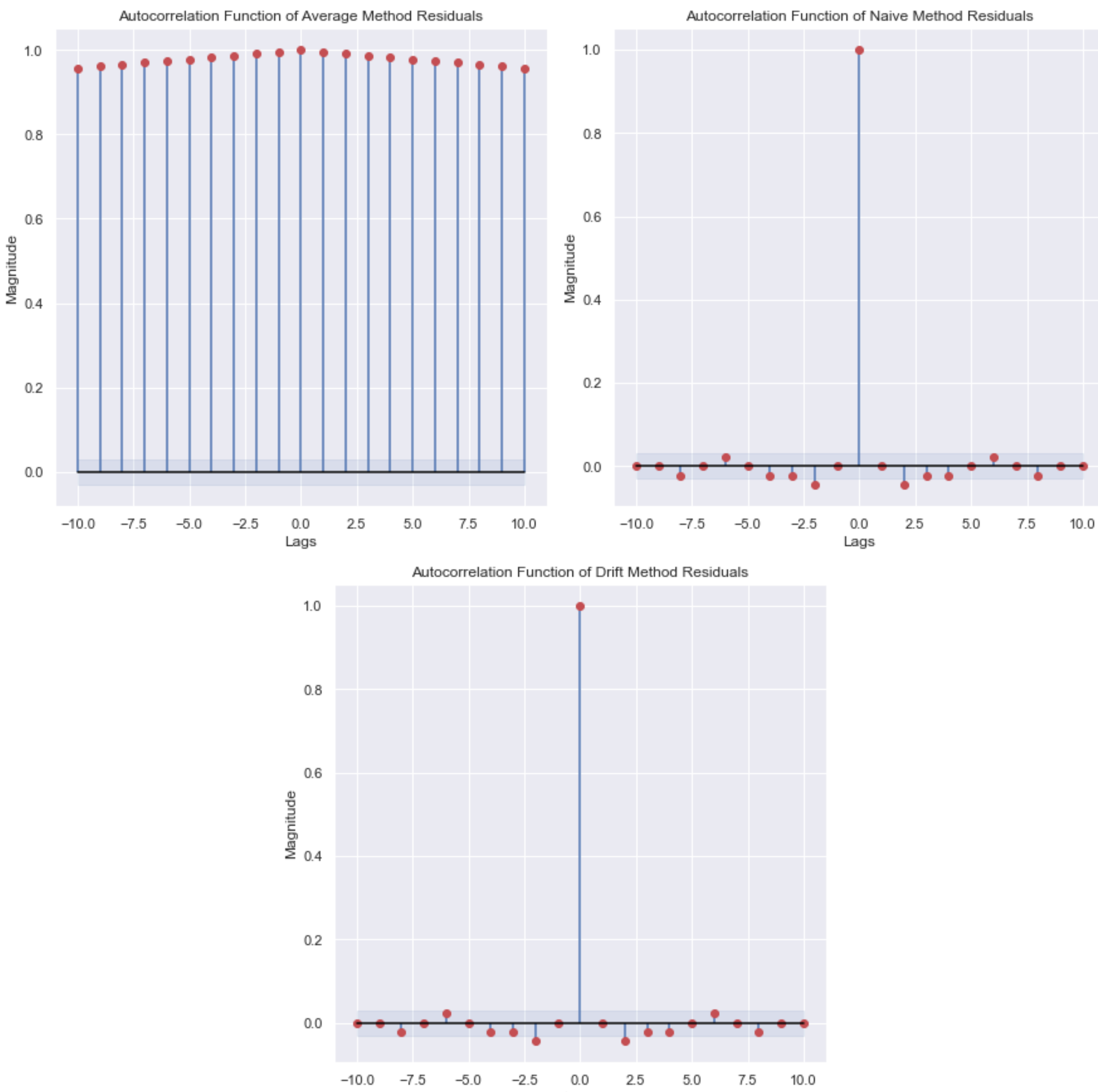


Figure 13 ACF of Residuals for Average, Naive and Drift Methods

	Q via Box-Pierce	MSE - Residual	MSE - Forecast	Variance - Residual
Average	20617.16	1.18	0.6	0.0
Naive	12.57	0.01	0.0	0.0
Drift	12.57	0.01	0.0	0.0

Table 1 Diagnostic Analysis for Average, Naive and Drift Method

Figure 14 1-Step Prediction for Simple Exponential Smoothing (SES) Method w/ Different Alphas

Feature Engineering

For feature engineering, I performed a single value decomposition (SCD) and backwards stepwise regression. SVD is a linear algebra technique that decomposes a matrix into three constituent matrices: one orthogonal matrix representing the row space, one diagonal matrix containing singular values, and another orthogonal matrix representing the column space. SVD is employed to identify collinearity among features by examining the singular values; small or near-zero singular values suggest the presence of collinearity. In my case, I removed features that had a singular value lower than or equal to 0.5. Figure 15 shows a snapshot of the process indicating that only *S7_PIR* was considered to have high collinearity with other features.

Backward stepwise regression is a feature selection method in which all predictor variables are initially included in the model, and then iteratively removed one at a time based on a predefined criterion. For my use case, I used the BIC, AIC and R^2-adjusted values as my criteria to down select. This process continues until the optimal subset of features is obtained. Backward stepwise regression helps identify collinearity by excluding redundant or highly correlated features, ultimately improving model performance and interpretability. Figure 15 shows that *S1_Temp, S1_Sound, S4_Temp* and *S7_PIR* are considered redundant amongst all three criterions with backwards stepwise regression. Therefore, for the multiple linear regression in the next section, I chose to remove those features (*S1_Temp, S1_Sound, S4_Temp* and *S7_PIR)*.

```
SVD Results
Via SVD with a threshold of <= 0.5 the following fields exhibit colinearity: S7_PIR

Backwards Stepwise Regression Results

Selected features via Backwards Stepwise Regression w/ BIC as Evaluation: S2_Temp, S3_Temp, S1_Light, S2_Light, S3_Li
ght, S4_Light, S2_Sound, S3_Sound, S4_Sound, S5_CO2, S5_CO2_Slope, S6_PIR

Selected features via Backwards Stepwise Regression w/ AIC as Evaluation: S2_Temp, S3_Temp, S4_Temp, S1_Light, S2_Lig
ht, S3_Light, S4_Light, S2_Sound, S3_Sound, S4_Sound, S5_CO2, S5_CO2_Slope, S6_PIR, S7_PIR

Selected features via Backwards Stepwise Regression w/ R2Adjusted as Evaluation: S2_Temp, S3_Temp, S4_Temp, S1_Light,
S2_Light, S3_Light, S4_Light, S2_Sound, S3_Sound, S4_Sound, S5_CO2, S5_CO2_Slope, S6_PIR, S7_PIR

Overlap amongst all: S2_Temp, S3_Temp, S1_Light, S2_Light, S3_Light, S4_Light, S2_Sound, S3_Sound, S4_Sound, S5_CO2,
S5_CO2_Slope, S6_PIR

Dropped Features: S1_Temp, S1_Sound, S4_Temp, S7_PIR
```

Figure 15 Snapshot of the SVD and Backward Stepwise Regression Results

Multiple Linear Regression

Figure 16 presents the h-step prediction for the given time series data using a multivariate linear regression based on the selected features from the previous section. As depicted, the forecasted results closely resemble the test set. Table 2 provides statistics related to the multivariate linear regression model. The adjusted R2 is relatively high at approximately 0.99, indicating a strong fit to the dataset. Both the AIC and BIC are significantly low, suggesting a well-performing model. All features exhibit a p-value of approximately 0.0, indicating their significance. The F-test also demonstrates significance with a p-value of approximately 0.0. A Q value of 0 implies that the false discovery rate (FDR) associated with a specific test is low, making the result more likely to be a true positive rather than a false positive. The root mean squared error (RMSE) is roughly 0.011, indicating a low error between the prediction and ground truth. Similarly, the residual mean and variance are approximately 0.0, conveying information akin to the RMSE. The ACF for the first 10 lags in the model are as follows: [1, 0.81, 0.74, 0.71, 0.68, 0.64, 0.63, 0.62, 0.59, 0.57, 0.56]. All in all, the multivariate linear regression with the selected features provides a good model for the given dataset based on these statistics. However, given the nature of the dataset, this model would fail in a real-world scenario if subject to a linear relationship.

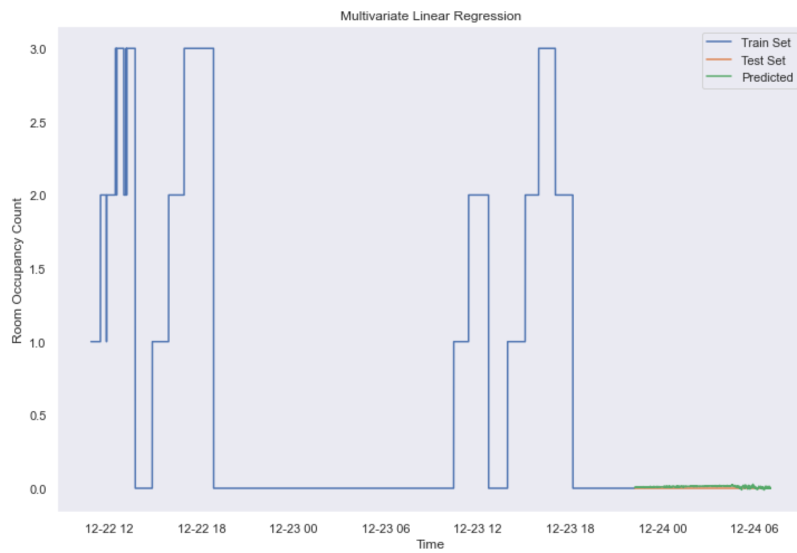


Figure 16 H-Step Prediction for Multivariate Linear Regression

OLS Regression Results

Dep. Variable:	Room_Occupancy_Count	R-squared:	0.992
Model:	OLS	Adj. R-squared:	0.992
Method:	Least Squares	F-statistic:	4.437e+04
Date:	Tue, 02 May 2023	Prob (F-statistic):	0.00
Time:	18:16:19	Log-Likelihood:	3894.0
No. Observations:	4244	AIC:	-7762.
Df Residuals:	4231	BIC:	-7679.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2871	0.181	1.587	0.112	-0.067	0.642
x1	0.0287	0.003	8.381	0.000	0.022	0.035
x2	-0.0418	0.008	-5.270	0.000	-0.057	-0.026
x3	0.0087	5.54e-05	157.322	0.000	0.009	0.009
x4	0.0032	3.35e-05	95.823	0.000	0.003	0.003
x5	0.0054	4.52e-05	118.630	0.000	0.005	0.005
x6	-0.0120	0.000	-116.882	0.000	-0.012	-0.012
x7	0.0275	0.006	4.735	0.000	0.016	0.039
x8	-0.0216	0.004	-5.630	0.000	-0.029	-0.014
x9	0.0570	0.012	4.580	0.000	0.033	0.081
x10	8.072e-05	1.51e-05	5.331	0.000	5.1e-05	0.000
x11	0.0137	0.001	9.537	0.000	0.011	0.017
x12	0.0194	0.005	3.825	0.000	0.009	0.029

Omnibus:	2979.315	Durbin-Watson:	0.719
Prob(Omnibus):	0.000	Jarque-Bera (JB):	9612168.711
Skew:	1.695	Prob(JB):	0.00
Kurtosis:	236.122	Cond. No.	7.75e+04

Table 2 Summary of Multivariate Linear Regression Statistics

ARMA

Order Determination

Using the ACF and PACF from figure 5 and the Generalized Partial Autocorrelation Function (GPAC) table presented in figure 17, I can conclude that the ARMA process should have AR=1 and MA=0 due to the repeating 1's in column 1 and approximately repeating values in row 0. Also, as stated before, the PACF has an immediate drop after lag 1 indicating AR process should be 1. As a second selection I also chose AR=5 and MA=0 because there is a slight repeating of 0.2 in column 5 and approximately 0 in row 0. Both models are used for parameter estimation in the next section.

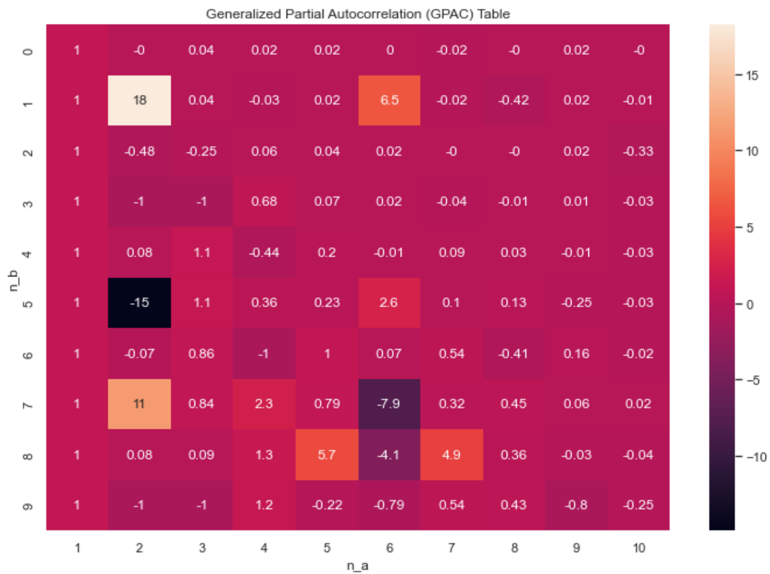


Figure 17 GPAC for Time Series Data

Parameter Estimation

The Levenberg-Marquardt algorithm (LMA) was employed, along with the parameter estimation provided by StatsModel's ARIMA module, to estimate the parameters for the ARIMA models discussed in the previous section. It is important to note that these models are actually ARMA, as they do not include any differencing parameter. Utilizing a custom-built LMA process, the ARMA(1,0,0) model was determined to have an AR parameter of -0.9969. As illustrated in Table 3, this is comparable to StatsModel's parameter estimate of -1 (with the negative applied to the AR in StatsModel). Additionally, the AIC and BIC values are similar to those found in the multivariate regression, and all parameters are deemed significant.

For the ARMA(5,0,0) model, the LMA process estimated -0.9972, 0.0431, -0.0226, 0.0019, and -0.0224, which are highly similar to StatsModel's -1.0, 0.04, -0.02, 0.0, and -0.02. Although the AIC and BIC values are comparable, several parameters lack significance. As a result, the ARMA(1,0,0) process will be used for the provided dataset moving forward. ARIMA or SARIMA models were not considered, as the ARMA model already yielded excellent results and as demonstrated in the subsequent section, has supporting statistics.

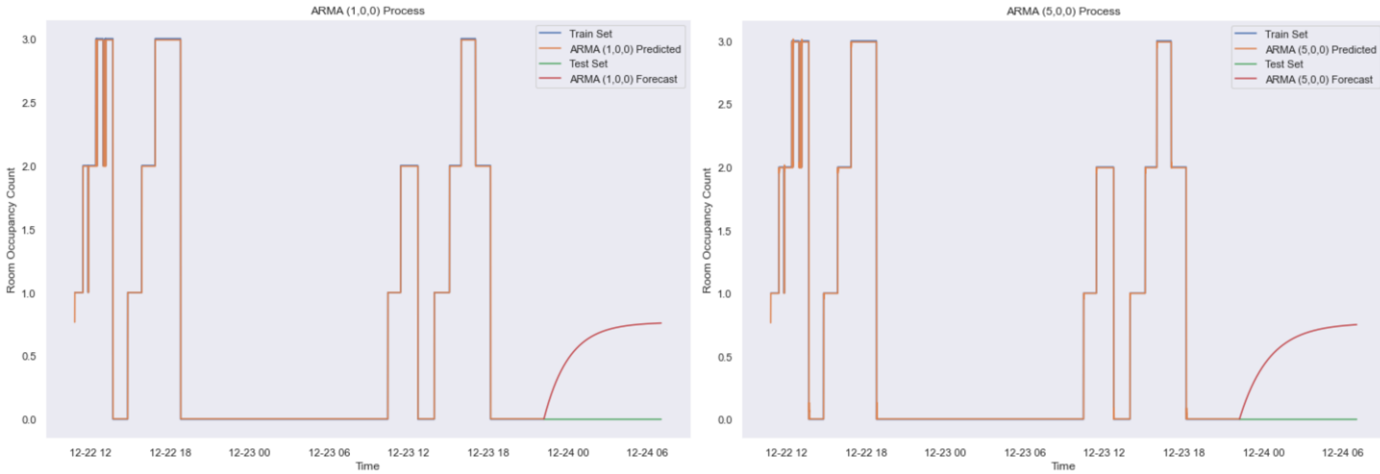


Figure 18 H-Step prediction for (Left) ARIMA(1,0,0) and (Right) ARIMA (5,0,0)

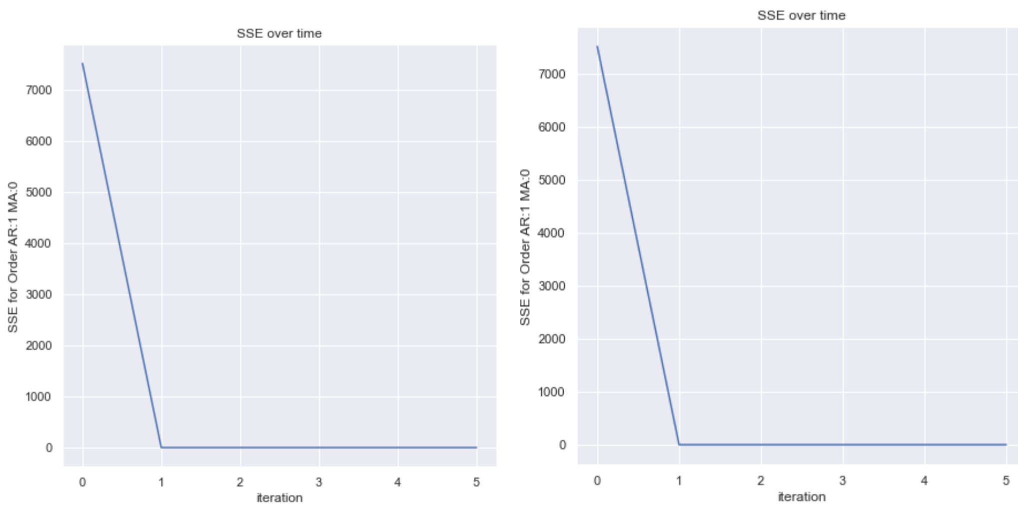


Figure 19 Sum Square Error via LMA for (Left) ARIMA(1,0,0) and (Right) ARIMA (5,0,0)

```
Statsmodel AR Parameters: [1.0]
Statsmodel MA Parameters: []
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 4244
Model:                 ARIMA(1, 0, 0)   Log Likelihood                3628.160
Date:                Wed, 03 May 2023   AIC                          -7250.320
Time:                        12:06:07   BIC                          -7231.260
Sample:                             0   HQIC                         -7243.584
                               - 4244
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7653      0.696      1.100      0.271      -0.598       2.129
ar.L1          0.9956      0.003    380.822      0.000       0.990       1.001
sigma2         0.0106   3.06e-05    346.280      0.000       0.011       0.011
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):          34339498.35
Prob(Q):                              0.88   Prob(JB):                         0.00
Heteroskedasticity (H):               0.45   Skew:                           -13.01
Prob(H) (two-sided):                  0.00   Kurtosis:                       442.90
===================================================================================
```

```
Statsmodel AR Parameters: [1.0, -0.04, 0.02, -0.0, 0.02]
Statsmodel MA Parameters: []
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 4244
Model:                 ARIMA(5, 0, 0)   Log Likelihood                3633.896
Date:                Wed, 03 May 2023   AIC                          -7253.791
Time:                        12:02:07   BIC                          -7209.319
Sample:                             0   HQIC                         -7238.074
                               - 4244
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7653      0.762      1.005      0.315      -0.727       2.258
ar.L1          0.9966      0.311      3.201      0.001       0.386       1.607
ar.L2         -0.0440      0.312     -0.141      0.888      -0.655       0.567
ar.L3          0.0231      0.012      1.849      0.064      -0.001       0.048
ar.L4         -0.0020      0.016     -0.130      0.897      -0.033       0.029
ar.L5          0.0224      0.011      2.094      0.036       0.001       0.043
sigma2         0.0106   3.29e-05    320.668      0.000       0.010       0.011
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):          34621449.22
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.46   Skew:                           -13.13
Prob(H) (two-sided):                  0.00   Kurtosis:                       444.70
===================================================================================
```

Table 3 Summary Statistics for (Left) ARIMA(1,0,0) and (Right) ARIMA (5,0,0)

Diagnostic Analysis

Table 4 shows basic statistics of the ARMA(1,0,0) process described in the prior section. As stated, the features are considered significant demonstrated by the p-value in table 3. Because this is strictly an AR process, there are no zero/pole cancellations. The residuals, with it's ACF represented in figure 20, indicate that they carry no additional information and are complete white noise.

```
Confidence intervals:
 [[-0.59834028  2.12897125]
 [ 0.99045538  1.00070321]
 [ 0.01051948  0.01063924]]

Zero/pole cancellation:
 [0.99557929]
No root cancellation, only AR process

Chi-square test:
Test statistic: 1
P-value: 1

Residuals:
Residual Variance: 0.0106
Residual Mean: -0.0002

Foreacst:
Foreacst Variance: 0.0361
Foreacst Mean: -0.6043
```

Table 4 Statistics for ARMA(1,0,0) Process

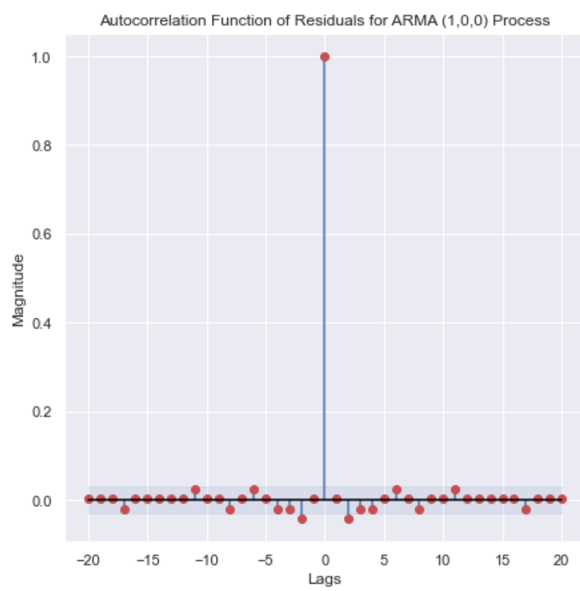


Figure 20 ACF of Residuals for ARIMA(1,0,0)

LSTM

For the Long Short-Term Memory (LSTM) model, a single dense layer with 32 units was implemented, utilizing a rectified linear unit (ReLU) for activation. The model was trained using a mean squared error (MSE) loss function and parameter optimization through the Adam optimizer. Training was conducted with a mini-batch size of 32 and a total of 50 passes. The final MSE for the model is 1.708E-4. This model demonstrates superior performance compared to several previously mentioned models. Figure 21 illustrates the h-step prediction using the given datasets.

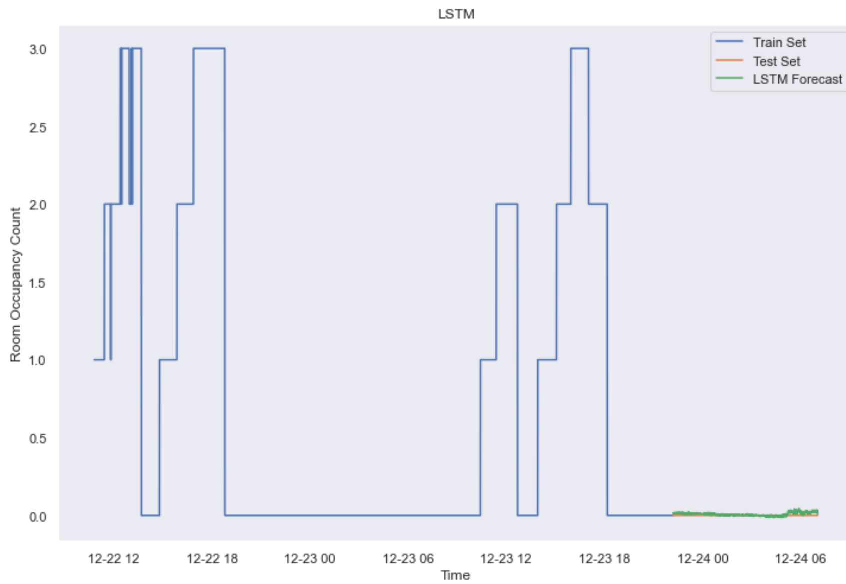


Figure 21 h-Step Prediction for LTSM

Summary & Conclusion

In conclusion, the most suitable models for the dataset are the ARMA(1,0) with AR coefficients [1] and the LSTM model. The GPAC, ACF, and PACF analyses support an ARMA model with AR=1 and MA=0. The significance of the parameters for the ARMA(1,0) compared to the ARMA(5,0) suggests that additional AR parameters are unnecessary, as the AIC and BIC values are approximately equal. The ACF of the residuals exhibits white noise, indicating that all relevant information is captured by the ARMA(1,0) model.

Regarding the LSTM model, the notably low MSE signifies that the model's predicted values are, on average, relatively close to the actual values. The LSTM model also accounts for multivariable interactions as they influence the dependent variable, whereas the ARMA process focuses solely on the sequential progression of the dependent variable. This dataset features characteristics not commonly encountered throughout the course, such as limited data and discrete outputs. It resembles potential real-world scenarios where data is scarce and resulting outputs are non-continuous. Given more data, a clearer and more representative model could be developed. However, considering the available data, the ARMA(1,0) and LSTM models offer the most accurate representation of the dataset.

References

[1] Reza, J. (2023) CS5526 Data Analytics II [Course Lectures]. Virginia Polytechnic Institute & State University, Blacksburg, VA, United States

Appendix

Code developed and used to generate the supporting results in this report: *cs5526_finalReport.py*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.model_selection import train_test_split
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import statsmodels.api as sm
from scipy.stats import chi2
import statistics
from scipy.signal import dlsim
from statsmodels.tsa.arima.model import ARIMA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.callbacks import EarlyStopping

sns.set(style="darkgrid")

################################################################################
# Methods
################################################################################

########################################
# Via Lab 1
def ADF_Cal(x, conf=0.05):
    result = adfuller(x)
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    print(f'Series is {"not " if result[1] >= conf else ""}stationary')

def kpss_test(timeseries, conf=0.05):
    print ('Results of KPSS Test:')
```

```
    statistic, p_value, n_lags, critical_values = kpss(timeseries, regression='c', nlags="auto")
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critical Values:')
    for key, value in critical_values.items():
        print(f'\t{key} : {value}')
    print(f'Series is {"not " if p_value < conf else ""}stationary')

def getRollingMeanVar(df, rnd=2):
    newDf = df.copy()
    for col in newDf.select_dtypes(include=np.number).columns:
        for row in range(len(newDf)):
            newDf.at[row, col+'_rollingMean']=newDf.iloc[0:row+1][col].mean()
            newDf.at[row, col+'_rollingVar']=newDf.iloc[0:row+1][col].var()
    newDf.loc[:, newDf.filter(regex='rolling').columns] = newDf.filter(regex='rolling').fillna(0.0)
    newDf.loc[:, newDf.filter(regex='rolling').columns] =  newDf.filter(regex='rolling').round(rnd)
    return newDf

def getRollingMeanVarPlot(df, x_label="",label=""):
    if label != x_label:
        fig, (ax1, ax2) = plt.subplots(2, 1, sharey=False, figsize=(10, 10))
        if x_label=="":
            ax1.plot(range(len(df)), df[label+'_rollingMean'])
            ax1.set_title(f'Rolling Mean for {label}')
            ax1.set_ylabel("Magnitude")
            ax2.plot(range(len(df)), df[label+'_rollingVar'])
            ax2.set_title(f'Rolling Variance for {label}')
            ax2.set_ylabel("Magnitude")
            plt.xlabel("Samples")
            plt.show()
        else:
            ax1.plot(df[x_label], df[label+'_rollingMean'])
            ax1.set_title(f'Rolling Mean for {label}')
            ax1.set_ylabel("Magnitude")
            ax2.plot(df[x_label], df[label+'_rollingVar'])
            ax2.set_title(f'Rolling Variance for {label}')
            ax2.set_ylabel("Magnitude")
            plt.xlabel("Samples")
            plt.show()
```

```python
############################################
# Via Lab 2

def getACFSingle(data, timeLag=0):
    """
    :data - numpy array
    :timeLag - t2-t1
    :return - Autocorrelation value [-1,1]
    """
    mean = np.mean(data)
    return np.sum((data[timeLag:]-mean)*(data[:len(data[timeLag:])]-mean))/np.sum(np.square(data - mean))

def getACFAll(data, timeLagMax=10, title="White Noise", plot = False, soloPlot=True):
    """
    :data - numpy array
    :timeLagMax - maximum time lag wish to calcualte. Will calculate all time lag from 0 to timeLagMax
    :return data - Autocorrelation value for timeLag in [0:timeLagMax]
    """
    returnData = []
    for timeLag in range(timeLagMax+1):
        returnData.append(getACFSingle(data,timeLag))
    returnDataX = list(range(-timeLagMax,0))+list(range(timeLagMax+1))
    returnDataY = returnData[::-1]+returnData[1:]
    if plot:
        if soloPlot:
            fig = plt.figure(figsize=(7,7))
        ci = 1.96/np.sqrt(len(data))
        plt.stem(returnDataX,returnDataY, markerfmt="or", basefmt="black",linefmt="C0-")
        plt.fill_between(returnDataX, len(returnDataX)*[-ci], len(returnDataX)*[ci], color='b', alpha=.1)
        plt.xlabel("Lags")
        plt.ylabel("Magnitude")
        plt.title("Autocorrelation Function of "+title)
    return returnDataX, returnDataY


############################################
# Via HW 2
def trainAvgMethod(train, test):
    predTrain = [0]
    for idx in range(len(train)):
        predTrain = predTrain + [round(sum(train[:idx+1])/(idx+1),1)]
```

```
    predTest = predTrain.pop(-1)
    return predTrain, len(test)*[predTest]

def trainNaiveMethod(train,test):
    predTrain = [0] + train[:-1]
    predTest = len(test)* [train[-1]]
    return predTrain, predTest

def trainDriftMethod(train,test):
    predTrain = [0,0]
    predTest = []
    for idx in range(2,len(train)):
        m = (train[idx-1]-train[0])/(idx-1)
        predTrain = predTrain + [train[idx-1]+m]
    m = (train[-1]-train[0])/len(train)
    for idx in range(len(test)):
        predTest = predTest + [train[-1]+(idx+1)*m]
    return predTrain, predTest

def getError(act, pred, offset=0):
    error = [round(a-b,1) for a, b in zip(act, pred)][offset:]
    return error, [round(x**2,1) for x in error]

def getMSE(errorSq,offset = 1):
    return round(sum(errorSq[offset:])/len(errorSq[offset:]),2)

def getMethodDF(train, test, offsetDF=1, method="Average"):
    if method=="Average":
        predTrain, predTest = trainAvgMethod(train, test)
    elif method=="Naive":
        predTrain, predTest = trainNaiveMethod(train, test)
    elif method=="Drift":
        predTrain, predTest = trainDriftMethod(train, test)

    errorTrain, errorTrainSq = getError(train, predTrain, offset=offsetDF)
    errorTest, errorTestSq = getError(test, predTest, offset=0)
    predTrain[:offsetDF]=offsetDF*[None]
    trainDF = pd.DataFrame({'actual': train, 'predicted': predTrain, "error":offsetDF*[None]+errorTrain,
"errorSq":offsetDF*[None]+errorTrainSq}, index=range(1,len(train)+1))
    testDF= pd.DataFrame({'actual': test, 'predicted': predTest, "error":errorTest, "errorSq":errorTestSq},
index=range(len(train)+1,len(train)+len(test)+1))
```

```
    return trainDF, testDF

def getGraph(dfTrain, dfTest, name="Average"):
    fig = plt.figure(figsize=(12,8))
    ## To connect train and test
        #plt.plot(list(dfTrain.index)+[len(dfTrain)+1], list(dfTrain.actual)+[dfTest.actual.iloc[0]], marker="o",label="Train Set")
    plt.plot(list(dfTrain.index), list(dfTrain.actual),label="Train Set")
    plt.plot(dfTest.index, dfTest.actual,label="Test Set")
    plt.plot(dfTest.index,dfTest.predicted,label="Predicted H-Step")
    plt.xlabel("Time")
    plt.ylabel("Dependent Variable")
    plt.title(name+" Method Prediction for Data")
    plt.grid()
    plt.legend()
    plt.show()

########################################
# Via Lab 5

def getPhiJKK(acf, j, k):
    assert(k<=len(acf))
    assert(j<=len(acf))
    assert((j+k)<=len(acf))
    acf = np.concatenate((acf[::-1],acf[1:]))
    mid = len(acf)//2
    pacfNum = []
    pacfDen = []
    for row in range(k):
        pacfNum.append(np.append(acf[mid+j+row:mid+j+row-k+1:-1],acf[mid+j+row+1]))
        pacfDen.append(np.append(acf[mid+j+row:mid+j+row-k+1:-1],acf[mid+j-k+row+1]))
    pacfNum = np.vstack(pacfNum)
    pacfDen = np.vstack(pacfDen)
#     return pacfNum, pacfDen
    ret = np.linalg.det(pacfNum)/np.linalg.det(pacfDen)
    return round(ret,2) if ret!=np.nan else np.nan


def getGPAC(acf, maxJ=7, maxK=7, plot=True):
    gpac = []
    for j in range(maxJ):
        row = []
```

```
        for k in range(1,maxK+1):
            row.append(getPhiJKK(acf,j,k))
        gpac.append(row)
    gpac = pd.DataFrame(gpac, columns=list(range(1,maxK+1)))
    if plot:
        fig = plt.figure(figsize=(12,8))
        sns.heatmap(gpac, annot=True)
        plt.title("Generalized Partial Autocorrelation (GPAC) Table")
        plt.xlabel("n_a")
        plt.ylabel("n_b")
        plt.show()
    return gpac

##########################################
# Via HW 5

def ACF_PACF_Plot(y,lags):
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure(figsize=(12,8))
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.subplot(212)
    plot_pacf(y, ax=plt.gca(), lags=lags)
    fig.tight_layout(pad=3)
    plt.show()

def arma_lma(y, ar_order, ma_order, delta=1e-6, max_iter=100,
             tol=1e-3, lambda_init=1e-2, lambda_factor=10, lambda_max = 1e9):
    ### HELPER FUNCTIONS
    def getDLSIMParams(params):
        """
        Note that this is to solve for eT rather than yT hence the ordering of the
        ar and ma coefficients are reversed
        """
        a = np.hstack((1, params[:ar_order])) ## ar
        b = np.hstack((1, params[ar_order:])) ## ma
        if ma_order < ar_order:
            b = np.append(b, np.array((ar_order-ma_order)*[0]))
        if ma_order > ar_order:
```

```python
        a = np.append(a, np.array((ma_order-ar_order)*[0]))
    return (a,b,1)

residual_fun = lambda params: dlsim(getDLSIMParams(params), y.squeeze())[1]
SSE_fun = lambda e: (e.T @ e).squeeze()

## INITIALIZATION
var = 0
cov = 0
params = np.zeros(ar_order+ma_order)
res = y.copy()
SSE = SSE_fun(res).item()
lambda_val = lambda_init
X = np.zeros((len(y),len(params)))
SSE_graph = [SSE]
error = ""
# PERFORM Levenberg-Marquardt
for i in range(max_iter):

    # CALCULATE X
    for j in range(len(params)):
        params_pert = params.copy()
        params_pert[j] += delta
        X[:, j] = ((res - residual_fun(params_pert)) / delta).squeeze()

    # CALCULATE A, g, DeltaTheta (dp)
    A = X.T @ X
    g = X.T @ res
    A += lambda_val * np.identity(len(A))
    dp = np.linalg.inv(A) @ g


    # UPDATE PARAMETERS
    params = params + dp.squeeze()
    res = residual_fun(params)
    SSE_new = SSE_fun(params)
    SSE_graph.append(SSE_new)

    # CONVERGENCE CHECK
    if SSE_new < SSE:
        if np.linalg.norm(dp) < tol:
```

```
            var = SSE_new/(X.shape[1]-X.shape[0])
            cov = (var*np.linalg.inv(A)).squeeze()
            break
        lambda_val /= lambda_factor
    else:
        lambda_val *= lambda_factor
        if lambda_val > lambda_max:
            print("ERROR BROKE LAMBDA_MAX")
            error = "ERROR BROKE LAMBDA_MAX"
            break
    SSE = SSE_new
  if i == max_iter:
    print("ERROR PASS ITERATION")
    error = "ERROR PASS ITERATION"
  return params, var, cov, SSE_graph, error

########################################
# New

def backward_stepwise_regression(X, y, criteria='aic'):
  p = X.shape[1]
  selected_features = np.arange(p)
  model = sm.OLS(y, X).fit()
  if criteria == 'aic':
    best_criteria_value = model.aic
  elif criteria == 'bic':
    best_criteria_value = model.bic
  else:
    best_criteria_value = -model.rsquared_adj
  for _ in range(p):
    best_feature_to_remove = None
    for feature_to_remove in selected_features:
      remaining_features = np.setdiff1d(selected_features, feature_to_remove)
      remaining_X = X[:, remaining_features]
      remaining_model = sm.OLS(y, remaining_X).fit()
      if criteria == 'aic':
        criterion_value = remaining_model.aic
      elif criteria == 'bic':
        criterion_value = remaining_model.bic
      else:
        criterion_value = -remaining_model.rsquared_adj
```

```
            if criterion_value < best_criteria_value:
                best_criteria_value = criterion_value
                best_feature_to_remove = feature_to_remove
        selected_features = np.setdiff1d(selected_features, best_feature_to_remove)
        selected_X = X[:, selected_features]
    return selected_features



################################################################################
# Results
################################################################################

## Data Processing
df = pd.read_csv("Occupancy_Estimation.csv", delimiter=',', skipinitialspace = True)
df.columns = df.columns.str.replace(' ', '')
print(df.shape)
df['time'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], format='%Y/%m/%d %H:%M:%S')
df.drop(columns=['Date','Time'], inplace = True)
df = df[df['time'].dt.month == 12]
df = df[df['time'].dt.day.isin([22,23,24])]
df = df[['time']+list(df.columns)[:-1]]
df['time'] = pd.date_range("2017-12-22 10:49:41", freq="30S", periods=len(df))

print("The shape of the December ONLY data is (row, column):", str(df.shape))

df.describe()
df.info()

df.drop(columns=['time'],axis=1).hist(figsize=(15,15))
plt.show()

plt.figure(figsize=(15,5))
plt.plot(df.time.values, df.Room_Occupancy_Count.values)
plt.xlabel('Time')
plt.ylabel('Room Occupancy Count')
plt.title('Time Series for Room Occupancy Count', fontsize=20)
plt.show()

## ACF and PACF
curr_fig, curr_ax = plt.subplots(figsize=(7, 7))
```

```python
plot_acf(df.Room_Occupancy_Count.values, lags=20, ax=curr_ax, title="ACF of Room_Occupancy_Count")
plt.show()

getACFAll(df.Room_Occupancy_Count, timeLagMax=20, title="of Room_Occupancy_Count", plot = True, soloPlot=True)
plt.show()

curr_fig, curr_ax = plt.subplots(figsize=(7, 7))
plot_pacf(df.Room_Occupancy_Count.values, lags=20, ax=curr_ax, title="PACF of Room_Occupancy_Count")
plt.show()

## Correlation Map
plt.figure(figsize=(10,10))
sns.heatmap(df.drop(['Room_Occupancy_Count'], axis=1).corr(method='pearson'), annot=False, cmap="seismic")
plt.title(f"Correlation Matrix for All Dependent Variables",  fontsize=20)
plt.show()

## Split data
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.2, random_state=17, shuffle=False)
dfY_train = pd.DataFrame({"time":X_train['time'], "Room_Occupancy_Count":y_train})
dfY_test = pd.DataFrame({"time":X_test['time'], "Room_Occupancy_Count":y_test})

print(f"Total Training Entires: {X_train.values.shape[0]}")
print(f"Total Test Entires: {X_test.values.shape[0]}")

## Stationality

## ADF and KPSS
print("ADF Test for Dependent Variable")
ADF_Cal(y_train, conf=0.05)
print("\n")
print("KPSS Test for Dependent Variable")
kpss_test(y_train, conf=0.05)

## Rolling Average and Variance of Independent
dfRolling = getRollingMeanVar(X_train, rnd=1)
for label in X_train.columns:
    getRollingMeanVarPlot(dfRolling, x_label="time", label=label)

## Rolling Average and Variance of Dependent
getRollingMeanVarPlot(getRollingMeanVar(dfY_train, rnd=1), x_label="time", label="Room_Occupancy_Count")
```

```python
X_train = X_train.set_index("time")
X_test = X_test.set_index("time")
dfY_train = dfY_train.set_index("time")
dfY_test = dfY_test.set_index("time")

## Time Series Decomposition

## Plot decomposed components
dfY_STL = STL(dfY_train['Room_Occupancy_Count'].values,  period=2*60*24).fit()
dfY_STL.plot().show()

## Plot detrended and seasonally adjusted
fig = plt.figure(figsize=(10,10))
plt.plot(dfY_train.index, dfY_train.Room_Occupancy_Count,label="Original")
plt.plot(dfY_train.index, (dfY_STL.resid + dfY_STL.seasonal),label=f"Detrended")
plt.plot(dfY_train.index, (dfY_STL.resid + dfY_STL.trend),label=f"Seasonally Adjusted")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title(f"Seasonally Adjusted vs Detrended vs. Original")
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()

## State trend and seasonal strength
Ft = np.maximum(0 ,1 - dfY_STL.resid.var()/(dfY_STL.resid+ dfY_STL.trend).var())
print(f'The strength of trend for this data set is ~{Ft*100:.2f}%')

Fs= np.maximum(0 ,1 - dfY_STL.resid.var()/(dfY_STL.resid+dfY_STL.seasonal).var())
print(f'The strength of seasonality for this data set is ~{Fs*100:.2f}%')

## Holt-Winters Model -- Additive Method
HWES3_ADD =
ExponentialSmoothing(dfY_train["Room_Occupancy_Count"].values,trend='add',seasonal='add',seasonal_periods=24).fit()
dfY_train['HWES3_ADD'] = HWES3_ADD.fittedvalues
dfY_test['HWES3_ADD'] = HWES3_ADD.forecast(len(dfY_test))

fig = plt.figure(figsize=(10,10))
plt.plot(dfY_train.index, dfY_train.Room_Occupancy_Count,label="Train")
plt.plot(dfY_train.index, dfY_train.HWES3_ADD,label=f"Training-Fit")
plt.plot(dfY_test.index, dfY_test.Room_Occupancy_Count,label="Test")
```

```
plt.plot(dfY_test.index, dfY_test.HWES3_ADD,label=f"Predicted H-step")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title(f"Holt Winters Triple Exponential Smoothing: Additive Seasonality on Training Set")
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()

## Base Models - Average, Naive, Drift
method =["Average", "Naive", "Drift"]
results = []
for i in method:
    print("\n============================================")
    print(f"{i} Method")
    dfTrain, dfTest = getMethodDF(list(dfY_train.Room_Occupancy_Count.values), list(dfY_test.Room_Occupancy_Count.values),
offsetDF=1, method=i)
    Q = sm.stats.acorr_ljungbox(dfTrain.error[2:], lags=[5], boxpierce=True, return_df=True)
    mseDF = pd.DataFrame({"MSE":[getMSE(list(dfTrain.errorSq),offset=2),getMSE(list(dfTest.errorSq),offset=0)],
                        "Variance":[round(statistics.variance(list(dfTrain.error[2:])),1),round(statistics.variance(list(dfTest.error)),1)],
                        "Q via Box-Pierce":[round(Q.bp_stat.iloc[0],2),None]},index=["Residual", "Forecast"])
    getGraph(dfTrain,dfTest, name=i)
    getACFAll(list(dfTrain.error[1:]), timeLagMax=10, title=f"{i} Method Residuals", plot = True)
    plt.show()
    print(mseDF)
    results.append(mseDF)

results = pd.DataFrame({"Q via Box-Pierce":[x["Q via Box-Pierce"].iloc[0] for x in results],
            "MSE - Residual":[x.MSE.iloc[0] for x in results],
            "MSE - Forecast":[x.MSE.iloc[1] for x in results],
            "Variance - Residual":[x.Variance.iloc[1] for x in results]},
           index=method)

## Base Models - Exponential Smoothing w/ different alphas
c = plt.cm.get_cmap("hsv", 14)
plt.figure(figsize=(12, 8))
plt.plot(dfY_train.index, dfY_train.Room_Occupancy_Count, color=c(10), label="Train")
plt.plot(dfY_test.index, dfY_test.Room_Occupancy_Count, color=c(9), label="Test")
alpha= [0.0,0.25,0.75,0.99]
for idx in range(len(alpha)):
```

```
    sse = SimpleExpSmoothing(dfY_train.Room_Occupancy_Count.values, initialization_method="known",
initial_level=dfY_train.Room_Occupancy_Count[0]).fit(
        smoothing_level=alpha[idx], optimized=False)
    plt.plot(dfY_train.index, sse.fittedvalues, color=c(idx))
    plt.plot(dfY_test.index, sse.forecast(len(dfY_test)), color=c(idx),label=r"$\alpha=$"+str(alpha[idx]))
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("Comparing Different Alpha for SES Method")
plt.grid()
plt.legend()
plt.show()

## Feature Selection / Elimination

## SVD
X_train_np = np.c_[np.ones(len(X_train)),X_train.iloc[:,1:].to_numpy()]
s, d, v = np.linalg.svd(X_train_np)
colinearityViaSVD = np.where(d<=0.5)[0]
print("SVD Results")
print(f"Via SVD with a threshold of <= 0.5 the following fields exhibit colinearity: {', '.join(list(X_train.columns[colinearityViaSVD]))}")

## BackwardsStepwise Regression
selectedFeatures_BIC = backward_stepwise_regression(X_train.values, y_train.values, criteria='bic')
selectedFeatures_AIC = backward_stepwise_regression(X_train.values, y_train.values, criteria='aic')
selectedFeatures_R2Adj = backward_stepwise_regression(X_train.values, y_train.values, criteria='rsquared_adj')
print("\nBackwards Stepwise Regression Results")
print(f"\nSelected features via Backwards Stepwise Regression w/ BIC as Evaluation: {',
'.join(list(X_train.columns[selectedFeatures_BIC]))}")
print(f"\nSelected features via Backwards Stepwise Regression w/ AIC as Evaluation: {',
'.join(list(X_train.columns[selectedFeatures_AIC]))}")
print(f"\nSelected features via Backwards Stepwise Regression w/ R2Adjusted as Evaluation: {',
'.join(list(X_train.columns[selectedFeatures_R2Adj]))}")
selectedFeatures =  list(set(selectedFeatures_BIC)&set(selectedFeatures_AIC)&set(selectedFeatures_R2Adj))
selectedFeatures_Dropped = list(set(range(X_train.values.shape[1])) - set(selectedFeatures))
print(f"\nOverlap amongst all: {', '.join(list(X_train.columns[selectedFeatures]))}")
print(f"\nDropped Features: {', '.join(list(X_train.columns[selectedFeatures_Dropped]))}")

## Multiple Linear Regression
linearReg = sm.OLS(y_train,sm.add_constant(X_train.iloc[:,selectedFeatures].values)).fit()

fig = plt.figure(figsize=(12,8))
```

```python
plt.plot(X_train.index, y_train, label="Train Set")
plt.plot(X_test.index, y_test, label="Test Set")
plt.plot(X_test.index, linearReg.predict(sm.add_constant(X_test.iloc[:,selectedFeatures].values)),label="Predicted")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("Multivariate Linear Regression")
plt.grid()
plt.legend()
plt.show()


## Summary Report
print(linearReg.summary())

## One-step ahead prediction on test set
X_test_ = sm.add_constant(X_test.iloc[:,selectedFeatures].values)
y_pred = linearReg.predict(sm.add_constant(X_test_))

# Calculate evaluation metrics
n = len(y_test)
k = X_test_.shape[1] - 1
resid = y_test.values - y_pred
rmse = np.sqrt(np.sum((resid)** 2) / (n - k - 1))
aic, bic, rsquared, rsquared_adj= linearReg.aic, linearReg.bic, linearReg.rsquared, linearReg.rsquared_adj

print("\nEvaluation Metrics:")
print(f"RMSE: {rmse:.4f}")
print(f"R-squared: {rsquared:.4f}")
print(f"Adjusted R-squared: {rsquared_adj:.4f}")
print(f"AIC: {aic:.4f}")
print(f"BIC: {bic:.4f}")


## Hypothesis tests
f_stat = linearReg.fvalue
f_pval = linearReg.f_pvalue
t_stat = linearReg.tvalues
t_pval = linearReg.pvalues
print("\nHypothesis Tests:")
print(f"F-statistic: {f_stat:.4f}")
print(f"F p-value: {f_pval:.4f}")
```

```
for i in range(1, k+1):
    print(f"t({i}): {t_stat[i]:.4f}, p-value: {t_pval[i]:.4f}")


# Calculate ACF of residuals and Q-value
acf = sm.tsa.stattools.acf(resid, nlags=10, qstat=True, fft=True)
q_value = acf[2][-1]
print("\nACF of Residuals:")
print(acf[0])
print("\nQ-value:")
print(f"Q-value: {q_value:.4f}")

# Calculate variance and mean of residuals
resid_var = np.var(resid)
resid_mean = np.mean(resid)
print("\nResiduals:")
print(f"Residual Variance: {resid_var:.4f}")
print(f"Residual Mean: {resid_mean:.4f}")

## ARMA, ARIMA, SARIMA -- GPAC
yTrain_acf = sm.tsa.acf(y_train.values, nlags=20)
# use one or the other for ACF
_, a = getACFAll(y_train.values, timeLagMax=20, title="Dependent Variable", plot = True, soloPlot=True)
# pacf
ACF_PACF_Plot(y_train.values,20)
getGPAC(yTrain_acf, maxJ=10, maxK=10, plot=True)

## ARMA Model Parameters -- LMA

## ARMA - AR:1, Diff:0, MA:0

AR = 1
Diff = 0
MA =0

params, _, _, SSE_graph, _ = arma_lma(y_train.values.reshape(-1, 1), AR, MA, delta=1e-6, max_iter=100,
        tol=1e-3, lambda_init=1e-2, lambda_factor=10, lambda_max = 1e9)

print(f"Predicted AR Order: {[round(i,4) for i in params[:AR]]}")
print(f"Predicted MA Order: {[round(i,4) for i in params[AR:]]}")
```

```
fig = plt.figure(figsize=(7,7))
plt.plot(list(range(len(SSE_graph))), SSE_graph)
plt.title("SSE over time")
plt.xlabel("iteration")
plt.ylabel("SSE for Order AR:1 MA:0")
plt.show()

ARMA = ARIMA(y_train.values, order=(AR, Diff, MA)).fit()
print(f"Statsmodel AR Parameters: {[round(i,2) for i in ARMA.arparams]}")
print(f"Statsmodel MA Parameters: {[round(i,2) for i in ARMA.maparams]}")
print(ARMA.summary())

fig = plt.figure(figsize=(12,8))
plt.plot(X_train.index, y_train, label="Train Set")
plt.plot(X_train.index, ARMA.predict(start=0, end=len(y_train)-1),label="ARMA (1,0,0) Predicted")
plt.plot(X_test.index, y_test, label="Test Set")
plt.plot(X_test.index, ARMA.forecast(steps=len(y_test)) ,label="ARMA (1,0,0) Forecast")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("ARMA (1,0,0) Process")
plt.grid()
plt.legend()
plt.show()

## ARMA - AR:5, Diff:0, MA:0

AR = 5
Diff = 0
MA =0

params, _, _, SSE_graph, _ = arma_lma(y_train.values.reshape(-1, 1), AR, MA, delta=1e-6, max_iter=100,
        tol=1e-3, lambda_init=1e-2, lambda_factor=10, lambda_max = 1e9)

print(f"Predicted AR Order: {[round(i,4) for i in params[:AR]]}")
print(f"Predicted MA Order: {[round(i,4) for i in params[AR:]]}")

fig = plt.figure(figsize=(7,7))
plt.plot(list(range(len(SSE_graph))), SSE_graph)
plt.title("SSE over time")
plt.xlabel("iteration")
plt.ylabel("SSE for Order AR:1 MA:0")
```

```
plt.show()

ARMA = ARIMA(y_train.values, order=(AR, Diff, MA)).fit()
print(f"Statsmodel AR Parameters: {[round(i,2) for i in ARMA.arparams]}")
print(f"Statsmodel MA Parameters: {[round(i,2) for i in ARMA.maparams]}")
print(ARMA.summary())

fig = plt.figure(figsize=(12,8))
plt.plot(X_train.index, y_train, label="Train Set")
plt.plot(X_train.index, ARMA.predict(start=0, end=len(y_train)-1),label="ARMA (5,0,0) Predicted")
plt.plot(X_test.index, y_test, label="Test Set")
plt.plot(X_test.index, ARMA.forecast(steps=len(y_test)) ,label="ARMA (5,0,0) Forecast")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("ARMA (5,0,0) Process")
plt.grid()
plt.legend()
plt.show()

## Diagnostic Analysis

ARMA = ARIMA(y_train.values, order=(1, 0, 0)).fit()
y_pred = ARMA.predict(start=0, end=len(y_train)-1)

# Confidence Interval
print("\nConfidence intervals:\n", ARMA.conf_int())

# Zero/Pole cancellation
print("\nZero/pole cancellation:\n", np.roots(np.r_[1, -ARMA.arparams]))
print("No root cancellation, only AR process")

# Calculate and display chi-square test
resid = y_train.values - y_pred
chi2, p_value = sm.stats.acorr_ljungbox(resid, lags=[len(ARMA.arparams)])
print("\nChi-square test:")
print("Test statistic:", chi2[0])
print("P-value:", p_value[0])

# Calculate variance and mean of residuals
resid_var = np.var(resid)
resid_mean = np.mean(resid)
```

```
print("\nResiduals:")
print(f"Residual Variance: {resid_var:.4f}")
print(f"Residual Mean: {resid_mean:.4f}")

# Calculate variance and mean of forecast
forecast = y_test.values - ARMA.forecast(steps=len(y_test))
forecast_var = np.var(forecast)
forecast_mean = np.mean(forecast)
print("\nForeacst:")
print(f"Foreacst Variance: {forecast_var:.4f}")
print(f"Foreacst Mean: {forecast_mean:.4f}")

## Show residuals are WN
_, _ = getACFAll(resid, timeLagMax=20, title="Residuals for ARMA (1,0,0) Process", plot = True, soloPlot=True)

fig = plt.figure(figsize=(12,8))
plt.plot(X_train.index, y_train, label="Train Set")
plt.plot(X_train.index, ARMA.predict(start=0, end=len(y_train)-1),label="ARMA (1,0,0) Predicted")
plt.plot(X_test.index, y_test, label="Test Set")
plt.plot(X_test.index, ARMA.forecast(steps=len(y_test)) ,label="ARMA (1,0,0) Forecast")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("ARMA (1,0,0) Process")
plt.grid()
plt.legend()
plt.show()

## Deep Learning Model -- LSTM

X_train_LSTM = np.reshape(X_train.values, (X_train.values.shape[0], 1, X_train.values.shape[1]))
X_test_LSTM = np.reshape(X_test.values, (X_test.values.shape[0], 1, X_test.values.shape[1]))


# define the model architecture
model = Sequential()
model.add(LSTM(units=32, activation='relu', input_shape=(X_train_LSTM.shape[1], X_train_LSTM.shape[2])))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mse')

# train the model with early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5)
```

```
model.fit(X_train_LSTM, y_train.values, epochs=50, batch_size=32, validation_data=(X_test_LSTM, y_test.values),
callbacks=[early_stop])

# make predictions on the test set
y_pred = model.predict(X_test_LSTM)

# calculate performance metrics (e.g. mean squared error)
mse = np.mean((y_test.values - y_pred)**2)
print('MSE:', mse)

# Residual and Forecast Variance and Meann
resid = y_train.values - model.predict(X_train_LSTM).squeeze()
resid_var = np.var(resid)
resid_mean = np.mean(resid)
print("\nResiduals:")
print(f"Residual Variance: {resid_var:.4f}")
print(f"Residual Mean: {resid_mean:.4f}")

forecast = y_test.values - model.predict(X_test_LSTM).squeeze()
forecast_var = np.var(forecast)
forecast_mean = np.mean(forecast)
print("\nForeacst:")
print(f"Foreacst Variance: {forecast_var:.4f}")
print(f"Foreacst Mean: {forecast_mean:.4f}")

## Show residuals are WN
# _, _ = getACFAll(resid, timeLagMax=20, title="Residuals for LSTM", plot = True, soloPlot=True)

## Plot train, test and predicted
fig = plt.figure(figsize=(12,8))
plt.plot(X_train.index, y_train, label="Train Set")
plt.plot(X_test.index, y_test, label="Test Set")
plt.plot(X_test.index, y_pred ,label="LSTM Forecast")
plt.xlabel("Time")
plt.ylabel("Room Occupancy Count")
plt.title("LSTM")
plt.grid()
plt.legend()
plt.show()
```