The Appliances Energy Consumption : Time-Series Forecasting

Joshitha Mandali
Towson University
jmanda2@students.towson.edu

***Abstract*—Over the years, the demand for energy has been continuously increasing in every sector. Such increased dependency on electrical appliances and electronic necessitates for future demands a forecast. Energy consumption forecasting plays an important role in planning the future in terms of implementing any future generating power plants as well as the prediction will help in increasing and maintaining the existing power stations. In this paper, an attempt is made on predicting the energy consumed by appliances at home using different classical time-series forecasting models and various features and metrics like RMSE, MAE have been calculated for the forecasting models to predict the best fit model. Initially, I wanted to predict the energy consumption by Auto Regression (AR), Moved Approach (MA) and Auto Regression Integrated moved approach (ARIMA) models but on researching about the best fit models for time-series forecasting, I have encountered few other models that changed my interest on it. Finally, in this paper I have predict the energy consumed by appliances using LSTM (Long Short-term Memory), Seasonal Autoregressive Integrated Moving Average with exogenous factors (SARIMAX) models and performed metrics Root Mean Squared Error (RMSE) and Mean Squared Error (MSE) to see the best fit model.**

Keywords— Energy, time-series, forecasting, Long Short-term Memory, Seasonal Autoregressive Integrated Moving Average with exogenous factors, Root Mean Squared Error, Mean Absolute Error

I. INTRODUCTION

In today's world, much of the world is focused on reducing energy consumption; our increase in consumption is neither economically nor environmentally sustainable. Additionally, there seems to be an increasing consensus that environmental economical sustainably are inextricably linked [1]. As the cost of energy rises, it has become essential to find technological solutions which can help reduce and optimize the energy use. Generally, residential homes contribute about 34% of total consumption in the USA and their consumption is estimated to increase to 39% by 2030 [2]. One way to reduce the energy consumption is to monitor how much power their appliances are using through a mechanism. And time- series data can be analyzed to extract meaningful statistics and other characteristics. The chosen dataset that has collected the energy consumption from different appliances for about four months.

In this paper, with the dataset available I wanted to analyze the energy consumed by appliances in the dataset and predict the future energy consumption by building different classic time-series forecasting models and provide results which model is the best fit. My initial choice was to explore Auto Regression (AR), Moved Approach (MA) and Auto Regression Integrated moved approach (ARIMA) model. But without having the knowledge of the dataset, I have decided on working with the AR, MA and ARIMA models

II. BACKGROUND AND MOTIVATION

Time series forecasting is an essential area in machine learning which is often neglected.

It is important because there are so many prediction problems that involve a time component. Such problems are neglected because it is this time component that makes time series problems more difficult to handle.

Due to my interest in time-series forecasting, I have opted to focus on predicting energy consumption using a more performant and efficient algorithm using Auto Regressive, Moving average and Auto Regressive Integrated Moving Average models. My primary goal is to predict the future energy consumption using different classical time-series forecasting and find out which is best fitting model for it.

I wanted to utilize the Auto Regressive (AR), Moving average (MA) and both combined models i.e., Auto regressive integrated moving average (ARIMA) methods and Python for this project. But reference [2] has given me insight on how to work with multivariate time series data, and with other references AR, MA and ARIMA are not best suited models for predicting timeseries forecasting.

III. LITERATURE REVIEW

To investigate what was expected from a data science project, especially one that deals with Timeseries Forecasting, I looked at [3]. As this was the original intention of my project, this gave me the boundaries on which to base my project.

I would need to decide on a topic, so 'Time-Series Forecasting' sounded fascinating. I have then decided on a dataset, as is discussed in the Description of the Dataset. Later, I have decided on the analyzing the classical models to do timeseries forecasting AR, MA and ARIMA. As I did not have much insight into any of these topics, I have investigated them individually. Reference [4] gave me insight into what Time-Series Forecasting and what are best suited models are.

To investigate timeseries forecasting models, I have started with [7] and [8], which gave me some insight into what the classical model of forecasting is. Once I started diving into the topic deeper and got to know that the best suited forecasting models for multivariate database are

LSTM (Long Short-term Memory), Seasonal Autoregressive Integrated Moving Average with exogenous factors (SARIMAX) and Vector autoregression (VAR). LSTM [11], Multistep timeseries forecasting with LSTM [12]. Apart from LSTM, another model that made me focus on was SARIMAX [13] have decided on using TensorFlow Federated as the platform, because it was heavily documented ([6] and [8]) and had some Image Classification tutorials ([13] and [14]).

This literature review formed the foundation of what this research paper ended up becoming. The timeseries forecasting knowledge gained was transposed to provide insight into the LSTM and SARIMAX models work that I have ended up researching, modeling and implementing.

IV. DESCRIPTION OF THE DATASET

The dataset has been selected from UCI Machine Learning Repository [4]. This dataset has been collected for about four and half month for every 10 minutes. Dataset consists of measurements of temperature and humidity sensors. The dataset chosen is mainly for data-driven predictive models. This particular data used in this paper includes measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station and recorded energy readings. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network. Each wireless node transmitted the temperature and humidity conditions around 3.3 min. Then, the wireless data was averaged for 10 minutes periods. The energy data was logged every 10 minutes with m-bus energy meters. Weather from the nearest airport weather station was downloaded from a public data set from Reliable Prognosis (and merged together with the experimental data sets using the date and time column)

The choice of Appliances Energy Consumption was actually my first choice. In terms of the short time period of the semester, this would have been the better choice for me, who had only started working with data mining this semester. Although, I have encountered many hurdles finding out the features of this dataset. I did not expect that by the end of the semester I would have been able to present anything beyond stating that the dataset was over my heads. In the spirit of being able to derive a useful understanding of Timeseries Forecasting, I have put my complete time and effort into this project. As an individual project, I feel that this dataset was close enough to the current knowledge that we I could progress and the means to processing the data would require enough reach that I would grow as student.

Below are the attribute information and their description

1. date: year-month-day hour: minute: second
2. Appliances: energy use in Wh (watt-hour)
3. lights: energy use of light fixtures in the house in Wh (watt-hour)
4. T1: Temperature in kitchen area, in Celsius
5. RH_1: Humidity in kitchen area, in %
6. T2: Temperature in living room area, in Celsius
7. RH_2: Humidity in living room area, in %
8. T3: Temperature in laundry room area
9. RH_3: Humidity in laundry room area, in %
10. T4: Temperature in office room, in Celsius
11. RH_4: Humidity in office room, in %
12. T5: Temperature in bathroom, in Celsius
13. RH_5: Humidity in bathroom, in %
14. T6: Temperature outside the building (north side), in Celsius
15. RH_6: Humidity outside the building (north side), in %
16. T7: Temperature in ironing room, in Celsius
17. RH_7: Humidity in ironing room, in %
18. T8: Temperature in teenager room 2, in Celsius
19. RH_8: Humidity in teenager room 2, in %
20. T9: Temperature in parents' room, in Celsius
21. RH_9: Humidity in parents' room, in %
22. To: Temperature outside (from Chievres weather station), in Celsius
23. Pressure: (from Chievres weather station), in mm Hg
24. RH_out: Humidity outside (from Chievres weather station), in %
25. Wind speed: (from Chievres weather station), in m/s
26. Visibility: (from Chievres weather station), in km
27. Tdewpoint: (from Chievres weather station), Ã‚Â°C
28. rv1: Random variable 1, nondimensional
29. rv2: Random variable 2, nondimensional

Where indicated, hourly data (then interpolated) from the nearest airport weather station (Chievres Airport, Belgium) was downloaded from a public data set from Reliable Prognosis, rp5.ru. Permission was obtained from Reliable Prognosis for the distribution of the 4.5 months of weather data.

V. EXPLORATORY DATA ANALYSIS

A. Data Cleaning

I have utilized Jupyter Notebook and python code to clean the dataset acquired. In total the dataset comprises of 19735 rows and 29 columns out of which 28 attributes are integer type and 1 attribute is of string. I have checked for null values, but the dataset did not contain any null values to be dropped.

Loading the dataset, finding the displaying the five rows of data

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        from scipy import stats
        import statsmodels.api as sm
        import warnings
        from itertools import product
        from datetime import datetime
        warnings.filterwarnings('ignore')
        plt.style.use('seaborn-poster')

In [19]: data = pd.read_csv("/Users/Joshitha/Downloads/energydata_complete.csv")

In [21]: print('The number of rows in dataset is - ' , data.shape[0])
         print('The number of columns in dataset is - ' , data.shape[1])

         The number of rows in dataset is -  19735
         The number of columns in dataset is -  29
```

```
data.isnull().sum().sort_values(ascending = True)
```

```
date            0
Tdewpoint       0
Visibility      0
Windspeed       0
RH_out          0
Press_mm_hg     0
T_out           0
RH_9            0
T9              0
RH_8            0
T8              0
RH_7            0
T7              0
rv1             0
RH_6            0
RH_5            0
T5              0
RH_4            0
T4              0
RH_3            0
T3              0
RH_2            0
T2              0
RH_1            0
T1              0
lights          0
Appliances      0
T6              0
rv2             0
dtype: int64
```

B. Feature Selection

In order to better understand the data and analyze it properly, To get the aggregate energy consumption, the related column values have been combined together. For instance, columns “T1”, “T2”, “T3”, “T4”, “T5”, “T6”, “T7”, “T8” and “T9” are combined together into one column temperature, whereas weather related columns like “T_out”, “Tdewpoint”, “RH_out”, “Press_mm_hg”, “Windspeed” and “Visibility” are combined into one column weather.

Also, column time comprises of column “date” and column target comprises of column “Appliances”

```
col_time=["date"]
col_temp = ["T1","T2","T3","T4","T5","T6","T7","T8","T9"]
col_hum = ["RH_1","RH_2","RH_3","RH_4","RH_5","RH_6","RH_7","RH_8","RH_9"]
col_weather = ["T_out", "Tdewpoint","RH_out","Press_mm_hg",
               "Windspeed","Visibility"]
col_light = ["lights"]
col_randoms = ["rv1", "rv2"]
col_target = ["Appliances"]
```

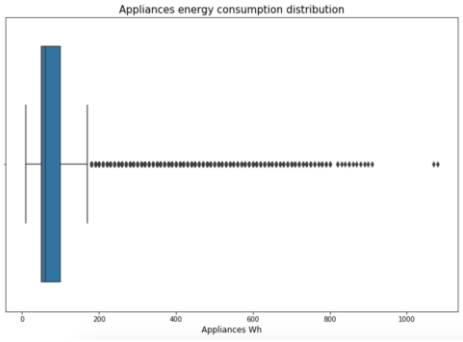
After dropping the unwanted columns, just using the related columns, the columns values have come up to 22 from 29.

```
values=data1.values
values.shape
```

```
(19735, 22)
```

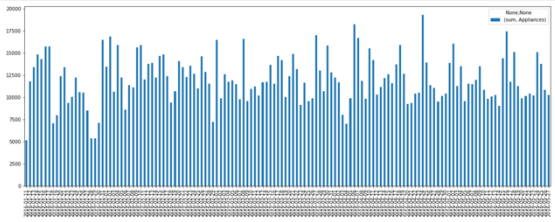
C. Data Visualization

The data visualization will help me understand the data in a better way, which will make me analyze the dataset and predict the models. Below is the appliances energy consumption plotted.



In the boxplot, the median is represented with a thick black line inside the blue rectangle. It has a value of around 60 to 70 Wh. The lower whisker has a value of 10 Wh and the upper whisker has a value of 170 Wh. It also shows that the data above the median is more dispersed and that there are several outliers (marked with the round circles above the upper whisker).

```
In [16]: dates=data["exact_date"].unique()
         arranged_day = pd.Categorical(data["exact_date"], categories=dates,ordered=True)
         date_series = pd.Series(arranged_day)
         table = pd.pivot_table(data,values="Appliances",index=date_series, aggfunc=[np.sum],fill_value=0)
         table.plot(kind="bar",figsize=(20, 7))
         plt.show()
```

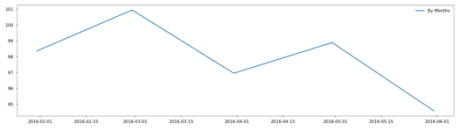


Below is the data plotted to see how the appliances consume energy by days, by months and by weeks to see the distribution

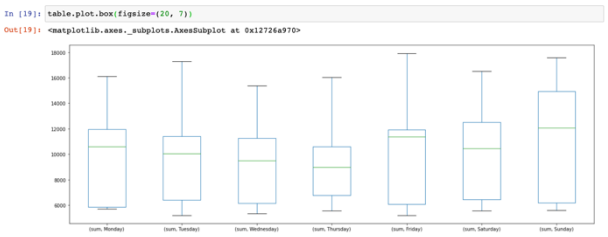
```
In [70]: fig = plt.figure(figsize=[65, 17])
         plt.suptitle('Appliances enegy consumption', fontsize=22)

         plt.subplot(221)
         plt.plot(df.Appliances, '-', label='By Days')
         plt.legend()
Out[70]: <matplotlib.legend.Legend at 0x156206bb0>
```

Visualizing energy usage by appliances by months

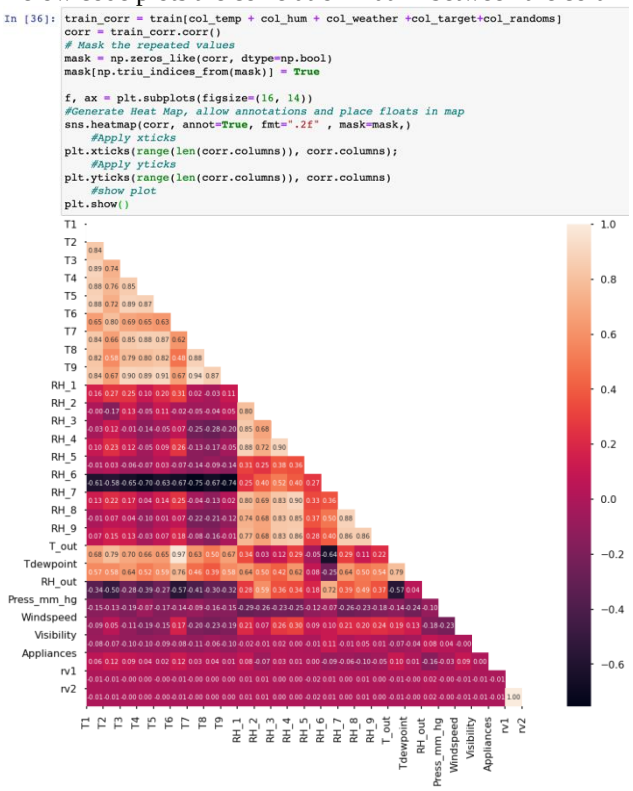
```
In [72]: fig = plt.figure(figsize=[65, 17])
         plt.suptitle('Appliances enegy consumption', fontsize=22)

         plt.subplot(222)
         plt.plot(df_month.Appliances, '-', label='By Months')
         plt.legend()
Out[72]: <matplotlib.legend.Legend at 0x1523e8c10>
```



Visualizing energy usage for a week. As we can see from below plot figure, the energy consumed on Saturday and Sunday is more



Below code plots the correlation matrix between the columns



The correlation plot helped me a lot in telling the features which gives positive correlation and when used for training the model might give good results. And features having negative correlation will not have much effect on the prediction

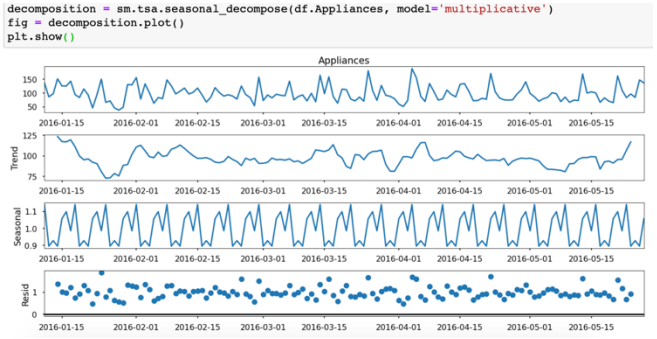
D. Time Series Decomposition

One of the most common analysis for time series is decomposing it into multiple parts. The parts we can divide a time series into are level, trend, seasonality and noise, all series contain level and noise, but seasonality and trend are not always present (there will be more analysis for this two parts).

These four parts can combine either additively or multiplicatively into the time series. Multiplicative models are nonlinear, trend is curved, and seasonality is not constant. Change is not constant over time. Decomposing is used to analyze the time series. Identify each one of the different parts of the time series and its behavior, each of the components may affect your models in different ways.

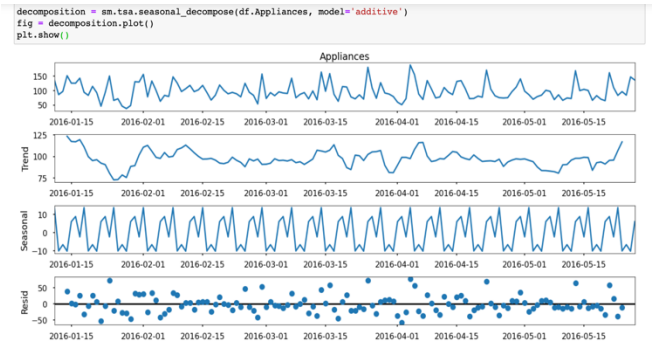
Most time series are a combination of an additive model and a multiplicate model, is hard to identify real world time series into one single model

y(t) = Level * Trend * Seasonality * Noise



Whereas Additives models are lineal. Trend is linear and seasonality has constant frequency and amplitude. Change is constant over time

y(t) = Level + Trend + Seasonality + Noise



So, the Trend-cycle — increases or decreases within the data in the long-term or not of a fixed frequency. For instance, if energy demand goes up over time as there are more people and businesses.

Seasonal — pattern occurs when a time series is affected by seasonal factors. For instance, we have seen the weekend pattern in our visualization data

Residuals — the remainder after removing the two aforementioned components. As every time series is inherently a stochastic process, there will always be random noises in the data. For example: There might be few broken light bulbs and the energy level thus fluctuates.

VI. METHODOLOGY AND ALGORITHM

A. Long Short Term Memory

Long short-term memory (LSTM) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the

connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

The expression long short-term refers to the fact that LSTM is a model for the short-term memory which can last for a long period of time. An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events. LSTMs were developed to deal with the exploding and vanishing gradient problem when training traditional RNNs [16]

```
def series_to_supervised(dataset, n_in=1, n_out=1, dropnan=True):
    num_vars = 1 if type(dataset) is list else dataset.shape[1]
    dataframe = DataFrame(dataset)
    cols, names = list(), list()

    # input sequence (t-n, ....t-1)
    for i in range(n_in, 0, -1):
        cols.append(dataframe.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(num_vars)]
    # forecast sequence (t, t+1 .... t+n)
    for i in range(0, n_out):
        cols.append(dataframe.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(num_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(num_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
```

```
feature=["RH_out","RH_8","RH_1","T3","RH_3","T2","Press_mm_hg","RH_2","RH_7","T8","RH_6","RH_4"
         "T4","T7","Tdewpoint","Windspeed","T1","T5"]
data1 = data[col_target + col_time + feature]
```

```
data1["date"]=pd.to_datetime(data1["date"])
data1 = data1.set_index(['date'], drop=True)
data1.head()
```

Scaling the training set and reframing the training set for efficient modelling

```
In [53]: scaler = MinMaxScaler(feature_range=(0,1))
         scaled = scaler.fit_transform(values)

In [54]: reframed = series_to_supervised(scaled, 1, 1)

In [55]: reframed.head()
```

Out[55]:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var8(t-1)	var9(t-1)	var10(t-1)	...	var13(t)	var14(t)
1	0.046729	0.894737	0.661412	0.566187	0.215188	0.746066	0.225345	0.097674	0.684038	0.653428	...	0.782437	0.381691
2	0.046729	0.894737	0.660155	0.541326	0.215188	0.748871	0.225345	0.100000	0.682140	0.651064	...	0.778062	0.380037
3	0.037383	0.894737	0.655586	0.530502	0.215188	0.755569	0.225345	0.102326	0.679445	0.646572	...	0.770949	0.380037
4	0.037383	0.894737	0.650788	0.524080	0.215188	0.758685	0.225345	0.104651	0.678414	0.641489	...	0.762697	0.380037
5	0.046729	0.894737	0.650788	0.531419	0.215188	0.758685	0.225345	0.106977	0.676727	0.639362	...	0.771233	0.379135

5 rows × 44 columns

Since LSTMs store long term memory state, I have created a data structure with 50 timesteps and 1 output. So, for each element of training set, we have 50 previous training set elements

```
reframed.drop(reframed.columns[[22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,
```

```
values = reframed.values
X = values[:,:21]
Y = values[:,21]
X.shape
```

```
(19734, 21)
```

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size=0.3)

# reshape input to be 3D [samples, timesteps, features]
X_Train = X_Train.reshape((X_Train.shape[0], 1, X_Train.shape[1]))
X_Test = X_Test.reshape((X_Test.shape[0], 1, X_Test.shape[1]))
```

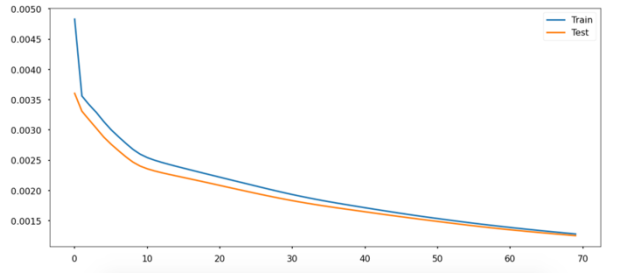
The LSTM architecture with the output layer and fitting to the training set

```
model = Sequential()
model.add(LSTM(50, input_shape=(X_Train.shape[1], X_Train.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')

# fit
history = model.fit(X_Train, Y_Train, epochs=70, batch_size=10, validation_data=(X_Test, Y_Test
```

```
Epoch 1/70
1382/1382 - 8s - loss: 0.0048 - val_loss: 0.0036
Epoch 2/70
1382/1382 - 4s - loss: 0.0036 - val_loss: 0.0033
Epoch 3/70
1382/1382 - 3s - loss: 0.0034 - val_loss: 0.0032
Epoch 4/70
1382/1382 - 3s - loss: 0.0033 - val_loss: 0.0030
Epoch 5/70
1382/1382 - 3s - loss: 0.0031 - val_loss: 0.0029
Epoch 6/70
1382/1382 - 3s - loss: 0.0030 - val_loss: 0.0028
Epoch 7/70
1382/1382 - 3s - loss: 0.0029 - val_loss: 0.0027
Epoch 8/70
1382/1382 - 3s - loss: 0.0028 - val_loss: 0.0026
Epoch 9/70
1382/1382 - 4s - loss: 0.0027 - val_loss: 0.0025
Epoch 10/70
1382/1382 - 3s - loss: 0.0014 - val_loss: 0.0013
Epoch 63/70
1382/1382 - 3s - loss: 0.0014 - val_loss: 0.0013
Epoch 64/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 65/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 66/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 67/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 68/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 69/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
Epoch 70/70
1382/1382 - 3s - loss: 0.0013 - val_loss: 0.0013
```

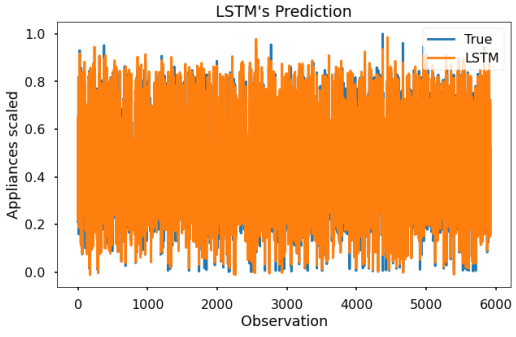
Preparing the test data and predicting the energy consumption

```
pyplot.plot(history.history['loss'], label='Train')
pyplot.plot(history.history['val_loss'], label='Test')
pyplot.legend()
pyplot.show()
```



Below is visualizing the results for LSTM

```
lstm_y_pred_test = model.predict(X_Test)
plt.figure(figsize=(10, 6))
plt.plot(Y_Test, label='True')
plt.plot(y_pred_test_lstm, label='LSTM')
plt.title("LSTM's Prediction")
plt.xlabel('Observation')
plt.ylabel('Appliances scaled')
plt.legend()
plt.show();
```



B. Seasonal Autoregressive Integrated Moving-Average with exogenous factors (SARIMAX)

Seasonal Auto-Regressive Integrated Moving Average with exogenous factors, or SARIMAX, is an extension of the ARIMA class of models. Intuitively, ARIMA models compose 2 parts: the autoregressive term (AR) and the moving-average term (MA). The former views the value at one time just as a weighted sum of past values. The latter model that same value also as a weighted sum but of past residuals (confer. time series decomposition). There is also an integrated term (I) to difference the time series (we will discuss this further below). This is such a rich topic with a lot of math involved [18]

Trend Elements: There are three trend elements that require configuration. They are the same as the ARIMA model, specifically: p: Trend autoregression order; d: Trend difference order; q: Trend moving average order.
Seasonal Elements: There are four seasonal elements that are not part of ARIMA that must be configured; they are:
P: Seasonal autoregressive order; D: Seasonal difference order; Q: Seasonal moving average order; m: The number of time steps for a single seasonal period. For example, an S of 12 for monthly data suggests a yearly seasonal cycle.

$$\text{SARIMA } (p,d,q) \times (P,D,Q,S)$$

Also, Lags are nothing but delaying in time steps within a series. For instance, consider a time index t, the lag 1 time index with respect to t is simply t-1, lag 2 is t-2, and so on. Whereas Stationarity is time series is one that has its mean, variance and autocorrelation structure unchanging overtime. In other words, it does not have any cycle/trend or seasonality. The ARMA model's family is actually built on this concept.

Model Identification:

I shall start by making sure that the data is stationary. Looking at the plots that I have made initially in section, it is evident that the data is not stationary with such a clear trend and seasonality.

```
]: print("Dickey–Fuller test: p=%f" % sm.tsa.stattools.adfuller(
```

```
Dickey–Fuller test: p=0.000000
```

From the above figure, the p-value is 0.00. Thus. I can say that the time-series is stationary, and I can reject the null hypothesis. With this I can know the parameters for the integrated terms in SARIMAX: for both seasonal and non-seasonal ones

Model Estimation:

This algorithm mainly conducts an exhaustive search over all the parameters combinations. Fig shows the best parameter combinations output for Seasonal ARIMAX

```
In [76]: import itertools
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

I ended up choosing the top four best models of the parameter's combinations based on loss function as per standard Seasonal ARIMAX modeling process as shown in Fig below shows

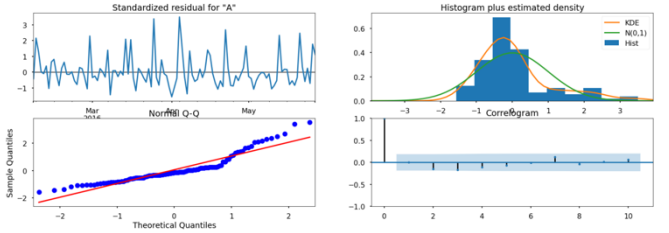
```
In [77]: for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(df.Appliances,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1658.7532763969257
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1428.672615405985
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1313.1647659322534
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1106.9819680790347
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1318.1457992688975
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1246.812389545211
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:1152.4882411221834
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1108.948270366783
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1541.3587631153246
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1359.5348550785525
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:1300.6379936358646
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1097.6124476711925
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1315.2894811596118
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1237.2447152217849
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1154.3111338739009
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1099.5876716945093
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1393.4310643175713
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1273.7778863925737
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1370.6220166317241
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1168.7101728241978
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:1284.3681017807576
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:1275.4413811391619
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:1217.0805405553356
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:1170.6258435193877
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1314.1722229601296
```

Model Validation:

In order to determine the goodness of fit of the model, one can examine its residuals using the standard assumption: they should be normally distributed around 0, or in other words, white noise. Random signal having equal intensity at different frequencies giving it a constant power spectral density is called White noise.

```
: results.plot_diagnostics(figsize=(25, 8))
plt.show()
```



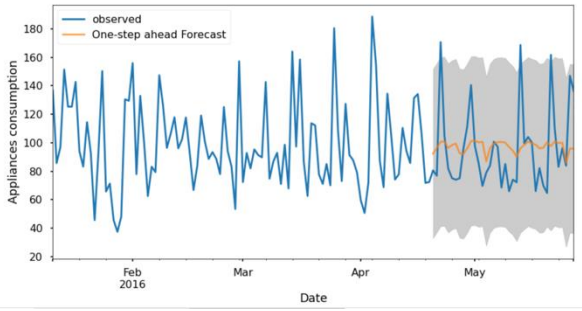
In the above plots, the residuals seem to be normally distributed around 0, which is the condition that I need with slightly heavy tails.

Now let's use the model to predict on the test set judge it

```
In [85]: mod = sm.tsa.statespace.SARIMAX(df.Appliances,
                                         order=(1, 1, 1),
                                         seasonal_order=(1, 1, 1, 12),
                                         enforce_stationarity=False,
                                         enforce_invertibility=False)

         results = mod.fit()

         print(results.summary().tables[1])
```

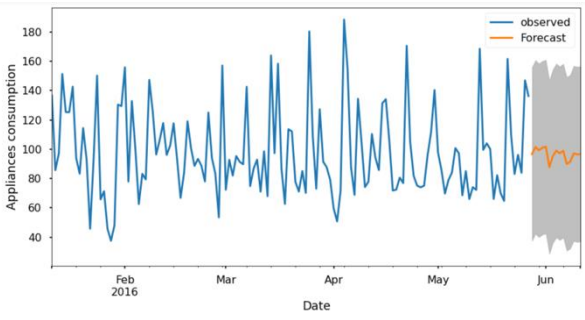
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0611	0.113	0.542	0.588	-0.160	0.282
ma.L1	-1.0000	388.338	-0.003	0.998	-762.129	760.129
ar.S.L12	-0.0017	0.020	-0.088	0.930	-0.040	0.037
ma.S.L12	-1.0000	388.377	-0.003	0.998	-762.206	760.206
sigma2	847.3239	0.315	2690.347	0.000	846.707	847.941

```
In [79]: results.plot_diagnostics(figsize=(15, 12))
         plt.show()
```



From the above figure, the overall in-sample prediction seems to fit the time series pretty well. There seems to be energy consumed on weekends is higher comparatively to that of energy consumed on weekdays. I am going to perform the forecasts on test data

```
In [83]: pred_uc = results.get_forecast(steps=15)
         pred_ci = pred_uc.conf_int()

         ax = df.Appliances.plot(label='observed', figsize=(14, 7))
         pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
         ax.fill_between(pred_ci.index,
                         pred_ci.iloc[:, 0],
                         pred_ci.iloc[:, 1], color='k', alpha=.25)
         ax.set_xlabel('Date')
         ax.set_ylabel('Appliances consumption')

         plt.legend()
         plt.show()
```



VII. Evaluate Forecasts

E. Root Mean Squared Error (RMSE)

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results [17]

Root Mean Square Error for LSTM model is 37.867

```
In :  yhat = model.predict(X_Test)
      X_Test = X_Test.reshape((X_Test.shape[0], 21))
      inv_yhat = np.concatenate((yhat, X_Test[:, -21:]), axis=1)
      inv_yhat = scaler.inverse_transform(inv_yhat)
      inv_yhat = inv_yhat[:,0]
      Y_Test = Y_Test.reshape((len(Y_Test), 1))
      inv_y = np.concatenate((Y_Test, X_Test[:, -21:]), axis=1)
      inv_y = scaler.inverse_transform(inv_y)
      inv_y = inv_y[:,0]
      rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
      print('Test RMSE: %.3f' % rmse)

      Test RMSE: 37.867
```

Root Mean Square Error for SARIMAX model is 28.34

```
In [82]: print('The Root Mean Squared Error of the ARIMA forecasts is {}'.format(round(np.sqrt(mse), 2)))

         The Root Mean Squared Error of the ARIMA forecasts is 28.34
```

F. Mean Squared Error (MSE)

Mean square error (MSE) is the average of the square of the errors. The larger the number the larger the error. **Error** in this case means the difference between the observed values and the predicted ones.

```
In [62]: lstm_test_mse = model.evaluate(X_Test, Y_Test, batch_size=1)
         print('Test MSE: %f'%lstm_test_mse)

         5921/5921 [==============================] - 9s 1ms/step - loss: 0.0013
         Test MSE: 0.001252
```

```
In [81]: y_forecasted = pred.predicted_mean
         y_truth = df.Appliances['2016-04-20':]

         mse = ((y_forecasted - y_truth) ** 2).mean()
         print('The Mean Squared Error of the ARIMA forecasts is {}'.format(round(mse, 2)))

         The Mean Squared Error of the ARIMA forecasts is 803.0
```

Model	Root Mean Square Error (RMSE)	Mean Squared Error (MSE)
LSTM	37.867	0.0012
SARIMAX	28.34	803.0

Table1: RMSE and MSE results

VIII. Analysis of Results

According to the table1 results, model LSTM depicted higher root mean squared error which 37.867 than SARIMAX which is 28.34.

In general, lower values of RMSE indicate better fit. But when seeing the Mean Squared Error value of the LSTM is nearly to zero which is like a perfect fit model, whereas MSE for SARIMAX is too high i.e., 803.0. Considering the values and reference [18] provided a direction for me to consider which as the best model in such situations. As simply put for MSE, the lower the value the better and 0 means the model is perfect fit. So, LSTM model shows best results from my prediction

IX. CONCLUSION

Time series forecasting is an essential area in machine learning which is often neglected. In this paper, on the Appliances Energy Consumption dataset, I have implemented a Long Short-term Memory and Seasonal Auto-Regressive Integrated Moving Average with exogenous model. I have further performed an analysis of the results of all the implemented algorithms, comparing the performance based on certain metrics. From the results acquired, I have seen that the Long Short-term Memory model showed the best results in terms of Mean Squared Root Error.

X. REFERENCES

[1] " Sustainable Development" Linking economy, society and environment by Tracey Strange and Anne Bayley https://www.oecd-ilibrary.org/docserver/9789264055742-en.pdf?expires=1618361634&id=id&accname=guest&checksum=F069AD64B1DDB26E0AB95E5CC3CC61AF

[2] Ehrhardt-Martinez, K. et al. Advanced metering initiatives and residential feedback programs: a meta-review for household electricity-saving opportunities (American Council for an Energy-Efficient Economy Washington, 2010).

[3] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[4] UCI Machine Learning Repository http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

[5] How to Choose the Right Forecasting Technique by John C. Chambers, Mullick and Donald D. Smith https://hbr.org/1971/07/how-to-choose-the-right-forecasting-technique

[6] 5 Statistical Methods For Forecasting Quantitative Time Series May 31, 2016, https://www.bistasolutions.com/resources/blogs/5-statistical-methods-for-forecasting-quantitative-time-series/

[7] 11 Classical Time Series Forecasting Methods in Python (Cheat Sheet) by Jason Brownlee on August 6, 2018 https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

[8] Time Series for beginners with ARIMA https://www.kaggle.com/freespirit08/time-series-for-beginners-with-arima

[9] An overview of time series forecasting models by Davide Burba https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb

[10] "I want to do multivariate time series forecasting with python. Could anyone suggest me which machine learning model I have to use?" Kashif Sultan, May 2017, https://www.researchgate.net/post/I_want_to_do_multivariate_time_series_forecasting_with_python_Could_anyone_suggest_me_which_machine_learning_model_I_have_to_use

[11] "Time Series Forecasting with the Long Short-Term Memory Network in Python" Jason Brownlee, April 2017, https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

[12] "Multistep Time Series Forecasting with LSTMs in Python" Jason Brownlee, May 2017, https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/

[13] "End-to-End Time Series Analysis and Forecasting: a Trio of SARIMAX, LSTM and Prophet (Part 1)" Bruce Nguyen, February 2021. https://towardsdatascience.com/end-to-end-time-series-analysis-and-forecasting-a-trio-of-sarimax-lstm-and-prophet-part-1-306367e57db8

[14] "Time-series-forecasting-with-python" By jiwidi https://github.com/jiwidi/time-series-forecasting-with-python/blob/master/02-Forecasting_models.ipynb

[15] "A Multivariate Time Series Guide to Forecasting and Modeling (with Python codes)" Aishwarya Singh, September 2018, https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/

[16] Long short-term memory From Wikipedia, free encyclopedia https://en.wikipedia.org/wiki/Long_short-term_memory

[17] https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/

[18] https://www.bmc.com/blogs/mean-squared-error-r2-and-variance-in-regression-analysis/#:~:text=There%20is%20no%20correct%20value,one%20prediction%20model%20over%20another.&text=100%25%20means%20perfect%20correlation.