A project report on

ELECTRICITY DEMAND FORECASTING AND ANALYSIS USING TIME-SERIES ALGORITHMS

Submitted in partial fulfilment for the award of the degree of

B.Tech. (Information Technology)

by

SOUBHIK SINHA (19BIT0303)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY & ENGINEERING

April, 2023

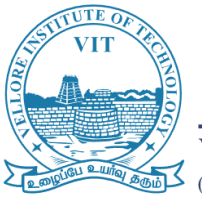
ELECTRICITY DEMAND FORECASTING AND ANALYSIS USING TIME-SERIES ALGORITHMS

Submitted in partial fulfilment for the award of the degree of

B.Tech. (Information Technology)

by

SOUBHIK SINHA (19BIT0303)



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY & ENGINEERING

April, 2023

DECLARATION

I hereby declare that the project report entitled “ELECTRICITY DEMAND FORECASTING AND ANALYSIS USING TIME-SERIES ALGORITHMS” submitted by me, for the award of the degree of B.Tech. (Information Technology) is a record of bonafide work carried out by me under the supervision of Dr. Bimal Kumar Ray.

I further declare that the word reported in this report has not been submitted and will not be submitted, either in part or full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 03/04/2023

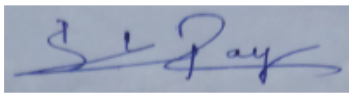
(SOUBHIK SINHA)

Signature of the Candidate

CERTIFICATE

This is to certify that the project report entitled "ELECTRICITY DEMAND FORECASTING AND ANALYSIS USING TIME-SERIES ALGORITHMS" submitted by SOUBHIK SINHA (19BIT0303), School of Information Technology & Engineering, Vellore Institute of Technology, Vellore for the award of the degree B.Tech. (Information Technology) is a record bonafide work carried out by him under my supervision.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project report fulfils the requirements and regulations of VELLORE INSTITUTE OF TECHNOLOGY, VELLORE and in my opinion meets the necessary standards for submission.



(Dr. BIMAL KUMAR RAY)

Signature of the Guide **Signature of the HoD**

Internal Examiner **External Examiner**

ABSTRACT

The ability to estimate demand is crucial for the power supply sector. To fulfil future demand, power production plants must be scheduled far in advance because electricity cannot be stored. Time series techniques on the historical demand series have been employed for demand prediction up till now in cases when online information about the external circumstances is not accessible. This paper introduces a project, which proposes the approach of using the appropriate Time-Series algorithms namely, LSTM (Long Short-Term Memory), ARIMA (Auto-Regressive Integrated Moving Average) and STM (Short Term Memory - RNN) Neural Network, evaluated on the basis of error rate (RMSE – Root Mean Squared Error) and R^2-score, as the evaluation metrics, on the dataset acquired from websites regulated by the Ministry of Power, Govt. of India. Later I show the prediction values of power production for 10 blocks of 15 minutes interval each for the next day. Afterwards I compare the prediction results to declare the winner among the 3 algorithms chosen and also compared the results obtained by the average of the predicted values with the original data-points (an extremely rudimentary step towards hybridization).

ACKNOWLEDGMENT

It is my pleasure to express with deep sense of gratitude to Dr. Bimal Kumar Ray, Professor (Higher Academic Grade), School of Information Technology & Engineering (SITE), Vellore Institute of Technology, Vellore, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavour. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Database Management Systems and Java Programming.

I would like to express my gratitude to Dr. G. Viswanathan, Chancellor, Vellore Institute Of Technology, Vellore, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Mr. G. V. Selvam, Vice – Presidents, Vellore Institute Of Technology, Vellore, Dr. Rambabu Kodali, Vice – Chancellor, Dr. Partha Sarathi Mallick, Pro-Vice Chancellor and Dr. S. Sumathy, Dean, School of Information Technology & Engineering (SITE), for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Usha Devi G., HoD, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 03/04/2023 **SOUBHIK SINHA**

ATTENDANCE

(No. of Interactions)

S. no.	Personnel	Motive & Activity Done	Remarks
1	Guide	0th Review: Submitted project topic and provided approximate idea & relevant explanation on how the project work shall be carried out.	Project title approved. 0th Review document signed and submitted.
2	Guide	Guide Interaction #1 : Gave a structured explanation of the project work. Presented 10 research papers relevant to the project title and their motive, working and flaws.	Guide suggestions: 1. Acquire knowledge on Time-Series Algorithms. 2. Include ARIMA apart from LSTM and RNN. 3. Concentrate on mathematical & theoretical working of models. 4. Find out appropriate source(s) of data (possibly from an uncommon source). 5. Create a new architecture with the inculcation of Short-Term Memory Neural Network.
3	Review Panellists	1st Review: Explained in detail how the project will be commenced and what could be the	Panellists' approval for 2nd Review confirmed.

		result. Presented the abstract, problem definition, lit. survey, objective, scope, architecture, and references. The Panellists asked questions regarding the project's usage in real world.	
4	Guide	Guide Interaction #2: Shared the progress of project implementation. Got suggestions on how to create 'design' diagram and what to be included in providing knowledge based on model working (theory and mathematical concepts).	Guide's approval for 2nd Review confirmed.
5	Review Panellists	2nd Review: Presented & explained 70% implementation.	Panellists' approval for 3rd Review.

TABLE OF CONTENTS

LIST OF FIGURES ……. *iii*

LIST OF TABLES ……... *iv*

LIST OF ACRONYMS …… *v*

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION ……..…. I

1.2 OBJECTIVES …… II

1.3 SCOPE OF THE PROJECT ……. III

CHAPTER 2

LITERATURE SURVEY

2.1 SUMMARY OF THE EXISITING WORKS ……. IV

2.2 CHALLENGES PRESENT IN EXISTING SYSTEM …….. XII

CHAPTER 3

REQUIREMENTS

3.1 HARDWARE REQUIREMENTS …… XIV

3.2 SOFTWARE REQUIREMENTS ……. XIV

CHAPTER 4

ANALYSIS & DESIGN

4.1 PROPOSED METHODOLOGY ……..... XV
4.2 SYSTEM ARCHITECTURE ……..… XVIII
4.3 COMPLETE DESIGN ……..... XIX
4.4 MODULE DESCRIPTION ……... XX

CHAPTER 5

IMPLEMENTATION & TESTING

5.1 DATA SET ……..…. XXXII
5.2 CODE & OUTPUT ……..…. XXXV
5.3 TESTING & PERFORMANCE METRICS ……...……. LI

CHAPTER 6

RESULTS ……. LXVI

CHAPTER 7

CONCLUSION & FUTURE WORK ……..... LXXIV

REFERENCES …… LXXVI

LIST OF FIGURES

1. Basic Neural Network Layers ……..... XXII

2. Basic Recurrent Neural Network (RNN) representation ……..…. XXII

3. Detailed layer representation of RNN …….. XXIII

4. Previous memory layer representation of RNN with vanishing gradient ……...……. XXIV

5. Working of RNN with numerically represented strings …… XXV

6. Working of RNN ……... XXV

7. **tanh** activation function ……...… XXVI

8. LSTM model representation (single layer) ……..… XXVII

9. **Sigmoid** activation function ……...……..... XXVII

10. Working of LSTM ……... XXVIII

11. Structure of 3-dimensional data ……..... XXXVIII

LIST OF TABLES

1. Literature Survey – Summary of Existing Work ……... IV

LIST OF ACRONYMS

MW	Mega-Watt
PSO	Particle Swarm Optimization
SVR	Support Vector Regression
ARIMA	Auto-Regressive Integrated Moving Average
ANN	Artificial Neural Network
AFD	Adaptive Fourier Decomposition
FFT	Fast Fourier Transform
HW	Holt-Winters
FOASVR	Fruit-fly Optimization Algorithm, optimized Support Vector Regression
GASVR	Genetic Algorithm, optimized SVR
MTP	Multiple Temporal Aggregation
MAPA	Multiple Aggregation Prediction Algorithm
GPU	Graphical Processing Unit
RAM	Random Access Memory
BP	Back-Propagation
DBN	Deep Belief Network
SVM	Support Vector Machine
ML	Machine Learning
MAPE	Mean Absolute Percentage Error
K-NN	K-Nearest Neighbour
NNDR	Neural Network Dynamic Regression
DRSARIMA	Dynamic Regression in Seasonal ARIMA
GUI	Graphical User Interface
DG	Distributed Generation
LSTM	Long Short-Term Memory (Neural Network)
STM	Short Term Memory
RNN	Recurrent Neural Network
DDR	Double Data Rate
SDRAM	Synchronous Dynamic Random-Access Memory

Chapter 1

Introduction

1.1 PROBLEM DEFINITION

Because of the nation's exponential industrial expansion and the resulting constant demand for power, several power plants have been built all around the country. Today's electricity demand, however, is driven by factors more than just expanding industry. Power demands are rising more quickly than anticipated. India is currently one of the world's top consumers of electricity; the country can thank its industrialisation and rising economy for this. But hold on! We simply specified how much energy the power plants will need to generate overall to meet our daily consumption. But it won't take place. Each power plant has a limited capacity for producing energy that it cannot exceed. Additionally, building a power plant in such a short amount of time is virtually unfeasible. If we break down our daily energy requirements, we may see a variety of patterns and swings. It is challenging for power plants to meet the requests of the purchasers (state governments) because of the wide variety of electricity demands and consumption. As already indicated, the electricity production fluctuates. The maximum output of a power plant, let us say one with a capacity of 2540 MW, can be changed depending on the situation. But on what condition? The government must get reports from the power plants every 15 minutes daily, detailing how much electricity they are producing. As a result, the state government plans to ask for the power demand for the next day. In this manner, data may be produced, but there is no analysis or forecast for the electricity that will be produced in the future. Let us assume that powerplant officials put ‘gut-instinct’ which is derived from their experience. But at times, sudden fluctuation of demand (i.e., a sudden need of electricity in a particular area OR a complete shutdown of power) can eventually cause the equipments to breakdown and decreasing efficiency.

1.2 OBJECTIVES

The aim of this project is to predict the values of power production for the initial 10 blocks (15 mins. Interval each) for the next day. First, prediction will be made on the values of last few blocks of the current day, giving out error rates and fitting values, w.r.t. test data. Later the models of LSTM, STM(RNN) and ARIMA will be taken as input parameter in a function to predict the electricity production values for the next day. Afterwards, comparison shall be made to conclude the best appropriate model for the issue – that can be efficacious for the power industry and whether the average values among the prediction values proves to be more efficient or not w.r.t. error rate, when compared to the original dataset values.

1.3 SCOPE OF THE PROJECT

The commencement of the project shall be done by keeping in mind that the result at the end should be industrially useful. For the very same reason, I am using a dataset, which is acquired from a government regulated website, which holds information about energy production and consumption – on daily basis. Then appropriate data shall be filtered out with necessary attributes. Data transformation and splitting can be carried out for creating training and testing data. Afterwards, model creation can take place and start the training procedure. As the prediction results must be de-transformed, we should take care by keeping in mind the implementation of inverse transform function. Afterwards, we can check for the error rate and positive or negative fitting of the predictions on the test data. Finally, the trained models will be exported and taken as an input parameter for a function which will provide the output as the final answer to our project. But the aforementioned was all about LSTM and STM(RNN). For ARIMA, the procedure deviates after data acquisition. We shall check for stationarity and seasonality and remove any trend that persists. Later, I create persistence model, AR, MA and the ARIMA models. The model performance can be evaluated based on error rates and fitting level. At last, comparison will be made to figure the best algorithm – suitable for the industries as well as how average prediction values will be useful to lower th error rate. This study will enhance the efficiency of the power plants in terms of cost as well as power production - which will later lead to lowering the equipment failure frequency and demand for coal (as coal prices are growing exponentially - nowadays they are being imported!).

Chapter 2

Literature Survey

2.1 SUMMARY OF THE EXISTING WORKS

S.no.	Title	Merits	Demerits
[1]	A hybrid data mining driven algorithm for long term electric peak load and energy demand forecasting	1. A major advantage of the proposed hybrid method is the combination of analytic time series method and the data mining approach which can handle the nonlinearity and seasonal trends in input samples (i.e., input observations). 2. The proposed hybrid approach prioritizes each algorithm based on its error over the input observations. The proposed hybrid method gives robust and accurate forecasting results. 3. It was shown that the proposed PSO-SVR and hybrid methods give more accurate results rather than ARIMA and ANN methods. 4. The PSO-SVR method is capable to forecast the load and energy demand with small errors without any sensitivity to seasonal trends in the original time series.	1. The ARIMA methods may results in undesired overestimation or underestimation in load and energy demand forecasting results. 2. For non-stationary time series (e.g., load and energy time series) identification of ARIMA orders and parameters is not an easy task. 3. Long term yearly peak load and energy demand forecasting is carried out in this paper.

		The reliable forecasts are obtained provided that the SVR parameters are set optimally using an optimization method such as the PSO algorithm.	
[2]	A novel hybrid forecasting scheme for electricity demand time series	1. AFD method can produce its basis adaptively based on processed signals to realize fast energy convergence. 2. The metaheuristics algorithm was applied to tune the parameters. And the hybrid method generally presents superior forecast performance. 3. The metaheuristic optimization algorithm is considered for combination to improve the instability.	1. The major disadvantage of the AFD is the inherent compromise that exists between frequency and time resolution. The length of AFD used can be critical in ensuring that subtle changes in frequency over time. 2. Metaheuristics Algorithms can be very effective on a given instance of a problem and, at the same time, show long running times on another without finding a satisfactory solution. 3. FFT cannot extract enough frequencies without enough samples
[3]	Holt–Winters smoothing enhanced by fruit fly optimization algorithm to forecast monthly electricity consumption	1. The proposed model can substantially improve the prediction accuracy of monthly electricity consumption even when few training samples are available. 2. The computation time of the proposed model is the shortest among the evaluated hybrid benchmark	1. The first disadvantage is that HW can take more time to generate forecasts. This is because the smoother curve requires more time to calculate and predict future values. The second disadvantage is that the HW method is

		algorithms. 3. The proposed hybrid model represents approximately twice the accuracy of the GASVR and FOASVR models.	less reliable than other forecasting methods. 2. FOA has weak ability to solve complex, high-dimensional and nonlinear optimization problems.
[4]	Cross-temporal aggregation: Improving the forecast accuracy of hierarchical electricity consumption	1. MTA has been shown to mitigate the model selection problem for low-frequency time series. 2. Propose a modification of the Multiple Aggregation Prediction Algorithm, a special implementation of MTA, for high-frequency time series to better handle the undesirable effect of seasonality shrinkage that MTA implies and combine it with conventional cross-sectional hierarchical forecasting. 3. The proposed MTA approach, combined with the optimal reconciliation method, demonstrates superior accuracy, aggregation consistency, and reliable automatic forecasting. 4. MTA significantly improves forecasting performance in terms of accuracy and bias. 5. Cross-sectional aggregation further enhances forecasting performance	1. Disadvantages of Cross-Sectional Study: - Cannot be used to analyse behaviour over a period to time - Does not help determine cause and effect - The timing of the snapshot is not guaranteed to be representative - Findings can be flawed or skewed if there is a conflict of interest with the funding source - May face some challenges putting together the sampling pool based on the variables of the population being studied. 2. MTA & MAPA: the exact demand from a specific region can't be ascertained, because it is aggregated.

		by combining appropriately, the base forecasts produced. 6. Applying MTA to seasonally adjusted data leads to better forecasts than applying MTA to the original series.	
[5]	Accuracy of different machine learning algorithms and added-value of predicting aggregated-level energy performance of commercial buildings	1. Boosted-tree, random forest, and ANN provided the best outcomes for prediction at hourly granularity when metrics such as computational time and error accuracy are compared. 2. Clustering decreases the amount of collected data, but the result for the grid side which is the predicted load will still be the same. 3. Considering computational power and time as well as error accuracy, boosted-tree, random forest, and ANN models were chosen for electricity demand pre- diction.	1. Two years of data were used in training the model and the prediction was performed using another year of untrained data. 2. Decision trees cannot be used well with continuous numerical variables. A small change in the data tends to cause a big difference in the tree structure, which causes instability. Calculations involved can also become complex compared to other algorithms, and it takes a longer time to train the model. 3. A forest is less interpretable than a single decision tree. Single trees may be visualized as a sequence of decisions. A trained forest may require significant memory for storage, due to the need for retaining the information from several hundred individual trees.

			4. ANNs usually require the use of expensive graphics processing units (GPUs) that allow parallel processing. They also need a lot of RAM. Once you have trained an ANN, sharing it is difficult. ANNs are prone to overfitting on the training data. ANNs do not guarantee convergence on a prediction or solution.
[6]	A building energy consumption prediction model based on rough set theory and deep learning algorithms	1. The integrated rough set and deep neural network was the most accurate. 2. The implementation of rough set theory was able to eliminate redundant influencing factors of building energy consumption. The DBN with reduced number of inputs had improved accuracy in building energy simulation. 3. The DBN had more accurate prediction of either short-term or long-term building energy consumption than the shallow neural networks such as BP, Elman and fuzzy neural networks.	1. Rough set don't handle data belong to quantitative 2. Following are the drawbacks of DBN : - DBN has hardware requirements. - DBN requires huge data to perform better techniques. - DBN is expensive to train because it has complex data models. - Hundreds of machines are required. - DBN is difficult to be used by less skilled people. - DBN requires classifiers to grasp the output.
[7]	Energy Consumption Prediction by using Machine Learning for Smart Building: Case Study in Malaysia	1. SVM result shows a lower mean absolute error. SVM predicted demand also had better accuracy when average consumption was	1. The SVM model took 13-18 hours to train. 2. Azure ML particularly does not

		calculated from the demand, in which. it achieved a lower MAPE than the rest of the methods for all tenants. 2. K-NN proves to be much faster in getting trained – by only taking 40 seconds.	have many algorithms needed for desired purpose. 3. SVM does not execute very well when the data set has more sound i.e., target classes are overlapping. In cases where the number of properties for each data point outstrips the number of training data specimens, the support vector machine will underperform. 4. Following are the demerits of K-NN: - Accuracy depends on the quality of the data. - With large data, the prediction stage might be slow. - Sensitive to the scale of the data and irrelevant features. - Require high memory – need to store all the training data. - Given that it stores all the training data, it can be computationally expensive
[8]	Superiority of the Neural Network Dynamic Regression Models for Ontario Electricity Demand Forecasting	1. The proposed NNDR model is superior to the commonly used DRSARIMA errors model and the Prophet model. 2. Long-term point forecasts and	1. Regression models cannot work properly if the input data has errors (that is poor quality data). If the data pre-processing is not performed well to remove missing values or redundant

		innovations are used to obtain two classes of prediction intervals (PIs) using data-driven probabilistic innovation distribution and bootstrapping for NNDR, DRSARIMA and Prophet models.	data or outliers or imbalanced data distribution, the validity of the regression model suffers. 2. Regression models work with datasets containing numeric values and not with categorical variables. There are ways to deal with categorical variables though by creating multiple new variables with a yes/no value. 3. Disadvantages of Prophet: - Works primarily on one time series - Requires data to be specified in a specific format - Performance varies by data set - All limitations that come with additive models
[9]	Pandemic-Aware Day-Ahead Demand Forecasting Using Ensemble Learning	1. Boosting and bagging-boosting models, are capable of accurate country-level demand forecast. 2. Utilizing the pandemic policy data as features increases the forecasting accuracy during the pandemic situation significantly. 3. The probabilistic quantile regression demonstrated high accuracy for the aforementioned case study.	1. Ensemble is less interpretable, the output of the ensembled model is hard to predict and explain. Hence the idea with ensemble is hard to sell and get useful business insights. 2. When using quantile classification gaps can occur between the attribute values. These gaps can sometimes lead to an

			over-weighting of the outlier in that class division.
[10]	Electrical Energy Demand Forecasting Using Artificial Neural Network	1. ANN was able to predict accurately the year of highest inflation - that too in a 15-year span. 2. ANN can make proper inferences, if given proper dataset.	- ANN can be a bit hefty to arrange and organize. - ANN can often create incomplete results or outputs. - ANNs are highly dependent on the data made available to them. This infers that the efficiency of any neural network is directly proportional to the amount of data it receives to process. - Minimal control, the trainers have over the actual performance and overall functioning of the ANNs.

2.2 CHALLENGES PRESENT IN EXISTING SYSTEM

In **[1]**, the project is carried out by considering normal environmental conditions – unlike during climate change or global warming, because they could not acquire data for the given 'unlike' condition. Using of hybrid univariate method by enhancing the same by a multi-variate method is still a topic of debate as there seems no dataset collected which stores data with respect other factors that can affect power production/consumption. Moreover, the analysis was done on non-renewable source of energy – rather than on the sustainable ones. The computational complexity of the proposed hybrid method in **[2]** get becoming incremental as the improvement process is more complex. There is no such investigation or analysis which claims that the forecasting performance can be improved if other predictive strategies and variables are integrated. Also, whether the model can balance the forecasting accuracy and the computational complexity such that the hybrid method performs well in performance under acceptable cost – is still doubted. Even though the proposed study of **[3]** claims that the monthly electric consumption can be accurately predicted – industries today are more inclined towards daily predictions, because nothing can be rigid for a long period (here, a months' time). The performance might have been evaluated by considering only one dataset, but what if we include other datasets, having chances of being quite imprecise. For **[4]**, they have not combined the forecasting methods across multiple temporal aggregation levels. Also, whether the outcome will be fruitful if combining temporal hierarchical levels to cross-sectional ones. A predictive probability distribution over future quantities or events of interest is a probabilistic forecast. Thus, expanding cross-temporal aggregation for probabilistic forecasting could have been tested. The inconsistency in the collected data and poor performance of ANN in **[5]** is a commonplace in most projects. The authors still must figure out deciding the levels at which the data should be used, the required computational speed and the required architecture for this conversion and which actors should be involved to make use of these data sets in an economical manner. Apart from the aforementioned, real-time analysis could have been employed which can eventually turn up industries to utilize on-site energy produced. In **[6]**, streaming data could not be analysed with the proposed model – though it may lack accuracy, but energy producers might be more interested whether the error rate of the model is high or low. In **[7]**, the study inculcated the use of Azure Machine Learning. But Azure ML has a major drawback - it offers fewer algorithms and other transformations built-in. Thus, one can of course reference Python and R to do this, but that is

much tougher than using the GUI. If one can figure out how to call Python/R from within Azure ML, he/she likely can do it on their own. This study's flaw is the length of time it took to run the SVM algorithm should thus be executed on a higher capable machine or platform. Second, given the limitations on the data collection in this study, more variables and data should be gathered as input. The focus of this work was more on the platform than the approach, hence a hybrid classifier was not presented. To distinguish the findings received, a comparison with another smart building might be added. It is challenging, as mentioned in **[10]**, to determine inflation accurately due to unpredictable reasons. By developing a high-accuracy prediction model, it will be possible to make reinforcements for the existing power system infrastructure more accurately and at the right time. In addition, as the management and control of the DG facilities to be established at the consumption points will be done more healthily, the power system management will be easier.

Chapter 3

Requirements

3.1 HARDWARE REQUIREMENTS

1. **Laptop** : HP Pavilion Gaming-15
2. **RAM** : Samsung DDR4 16GB (DDR-SDRAM)
3. **GPU** : NVIDIA GTX 1050
4. **Internal Memory Space** : 10GB (Giga-Byte)
5. **Processor** : Intel Core i5 (9^{th} Generation)

3.2 SOFTWARE REQUIREMENTS

1. Visual Studio Code (VS Code)
2. Google Colaboratory (Python 3.9.16)
3. Anaconda (Python 3.9.13 (base) & Python 3.9.16 (tf-gpu environment))
4. **Language used :** Python 3.11.2 (System Default)
5. Jupyter Notebook

Chapter 4

Analysis & Design

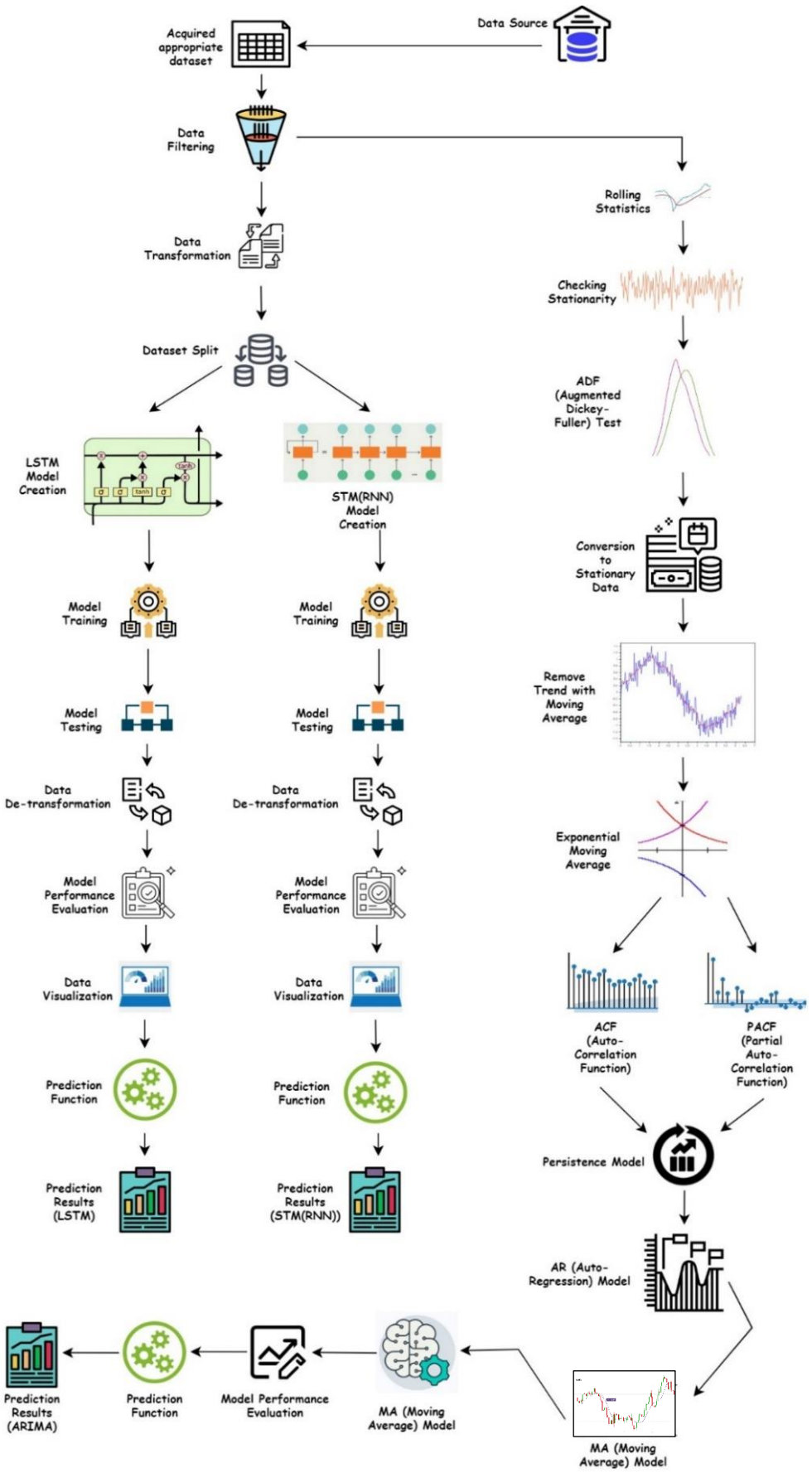
4.1 PROPOSED METHODOLOGY

The commencement of any analysis starts by selecting the appropriate data source. In this project, I will be using two websites by the name(s) Northern Regional Power Committee (NRPC) and Northern Regional Load Dispatch Centre (NRLDC), earlier known as POSOCO – Power System Operation Corporation Limited, which are regulated by the Ministry of Power, Government of India. Even though I can acquire dataset based on the power consumption of any state OR power production of any power-plant, I want to be constructive in this instance. I have another link (http://164.100.60.165/) to NRLDC which presents documents stating daily consumption & production of electricity for a month's time. Hence, it helped me to choose that power-plant which is catering to the State Governments (here, Clients), which are either densely populated OR have significant number of industries. Now that we have a dataset, it is time to choose an appropriate platform to carry out my implementation for my analysis. I have chosen **Google Colab**, as it offers enough RAM, storage space and GPU support for my project. After importing the dataset into the python notebook, I must filter out unnecessary attributes. Among the chosen attributes I have to check whether any anomalies are present in the dataset ('NA', 'NULL' values, blank cells, etc.). This project make use of 3 Time-Series Algorithms: **Recurrent Neural Network** (RNN), **Long Short-Term Memory** (LSTM) **Neural Network** – which is the evolved version of RNN (sometimes RNN is also known as Short Term Memory Neural Network) and **Auto-Regressive Integrated Moving Average** (ARIMA) **Neural Network**. The procedure for LSTM and RNN is same, except for model creation, while ARIMA goes for a completely different approach. For LSTM and RNN – after importing the dataset and checking for any anomalies, the data will be splitted in the ratio of (3:1) or simply 75% to 25% OR (0.75 : 0.25). The values of the dataset are significant; hence I am using **MinMaxScaler** to transform the values into a relatable value (decimal values) for model training, ranging between 0 and 1. Both algorithms will be first trained with 75% split data component – by ingesting the first 10 rows as input and learning that the 11^{th} row as output (Hence, Timesteps = 10). Then, from 2^{nd} row to 11^{th} row will be ingested to learn that 12^{th}

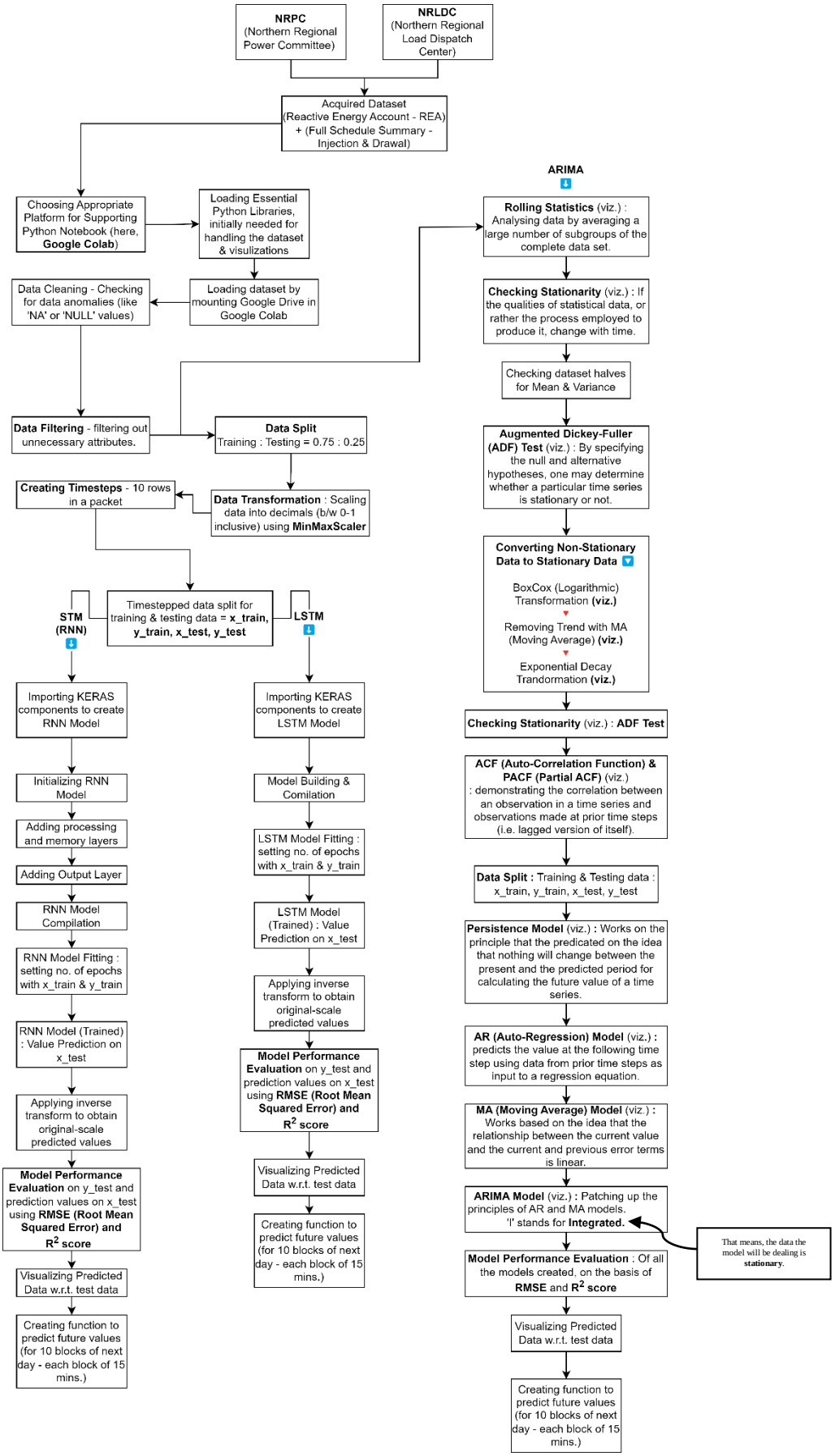
row is the answer (the very next row) and so on. Thus, in this way, the training data is used for model training. Hence, according to timesteps, training **(x_train & y_train)** and testing **(x_test & y_test)** components will be created. For LSTM and RNN Model creation, I will be using **Keras** library of Python as it contains all the required components to create the models. For RNN, layers will be added as processing & memory layers, including one last layer as output layer. Now, before the training of the models, I will have to make sure that LSTM and RNN ingest 3-Dimensioanl input data **(batch_size, timesteps, input_dim)**. Hence, I must reshape the data into that format. Afterwards, I will be setting the number of epochs (the no. of times the whole dataset will pass through the model training process) and start the training procedure. The prediction values will be acquired using the testing dataset (**x_test**). But here is a little complexity – I had used MinMaxScaler for data transformation for easing model training process. Hence, I now must apply inverse transformation function to the predicted values to get the original scaled values. The performance of the models (both LSTM & RNN) will not be evaluated based on accuracy OR precision because industries are more concerned about the safety and efficiency of the power generation procedures, and they also have the knowledge that immense model training can increase the accuracy and feeding larger dataset to the model will surely increase the precision. However, they (the power producers) are more concerned about the error rate. Hence, I will be evaluating the models based on RMSE (Root Mean Squared Error) and R^2 Score (For model prediction fitting) by comparing **y_test** and predicted values. Later the predicted values will also be shown through visualization to see how much deviation has occurred. But the LSTM & RNN component does not end here. Using the trained model, I will be creating a function to forecast how much power the power-station shall be generating for the initial 10 blocks (each block of 15 mins.) for the next day. The process is successful only for LSTM and RNN because they can deal with non-stationary data – something ARIMA cannot claim to work with. Hence, the first hurdle for ARIMA model implementation and training is to convert non-stationary data into stationary data. The first thing is to examine the data points through moving average (also known as **Rolling Statistics**), i.e., figuring out the average by creating subsets of data-points of the entire data set. Apart from that, I will obtain the Mean Value and Standard Deviation Value of 2 halves of the dataset – just to check the level of non-stationarity (in raw form). I will then call the function - **Augmented Dickey-Fuller Test (ADF Test)**, which takes the Null Hypothesis that the data is non-stationary, along with the Alternate Hypothesis, stating that the data is stationary. To de-trend the data (or say, adding stationarity) I will make use of the **BoxCox** (logarithmic) transformation, exponential (decay) transformation and try to remove trend

with moving average. Later, whole dataset is again tested through **ADF Test,** to check for any trace of non-stationarity. Now, I will make use of **ACF** (Auto-Correlation Function) and **PACF** (Partial Auto-Correlation Function) to check for correlation among data points with the current and a lagged version of the data. Before I create the ARIMA model, I will try to create 3 other models resembling the simple conditional working of ARIMA. The first model I will be creating is **Persistence Model** – assuming that nothing will change between the present and the projected period when calculating the future value of a time series. Next, I will create **AR (Auto-Regressive) Model** - a time series model that predicts the value at the following time step using observations from past time steps as the input to a regression equation. Then I will be creating **MA (Moving Average) Model –** working on the principle: a time series average that moves through the series by removing the top items from the most recent averaged group and include the next in each succeeding average. Now the **ARIMA** model is an ensemble of AR and MA model ('I' means integrated - enables the time series to become stationary by separating the raw observations (i.e., data values are replaced by the difference between the data values and the previous values)). Afterwards, all the models will be evaluated based on RMSE score and R^2 score. Finally, the values for 10 blocks for next day will be predicted. Lastly, the predicted values and evaluation metrics' values of ARIMA, LSTM and RNN will be compared to check which algorithm worked the best and also, the average of the predicted values will be compared to the original dataset values to check whether the error rate is lowered or not. Eventually, industries will be able to take decision which algorithm to inculcate with respect to their amount of data generated / acquired.

4.2 SYSTEM ARCHITECTURE



4.3 COMPLETE DESIGN



4.4 MODULE DESCRIPTION

Now that I have mentioned about the roadmap of my project about how everything will go by inculcating a top-to-bottom approach, it's time to know about the theoretical aspects and the working of our supreme actors of the play – the algorithms! But before that, we should also know about the category of the algorithms used – Time-Series.

WHAT IS TIME-SERIES?

We know that when dealing with ML, the most important aspect we give considerable amount of importance is DATA! The bigger the volume of data, the more the chances grow for extracting interesting patterns and that galvanizes us to dive deep and dig more to enhance the quality & quantity of meaningful questions as well as their answers – eventually, meeting a conclusion. When we talk about data – they depend on various factors – one such factor is **"TIME".** During analysis, when we conclude that the data acquired changes/repeats after certain **time-intervals** OR the data has a considerable amount of dependence on **time** – then the data is called – **Time Series Data.**

Now let us discuss some important terms that explain the topic in a detailed fashion –

1. **LEVEL:** To find the "level" of data (suppose you are given the data of SALES), the only action you will take is to find the **average**. Now average will be based on time. (e.g., in the SALES data, we can find the average sales value of 1 year – thus, time is included – the average value will be the **LEVEL** of data)

2. **TREND:** When we witness that – with time the results coming from data (as analysis output or just values of a particular attribute) is increasing OR decreasing – we call this as "**TREND**". Now trend are of 2 types – **Up-Trend** AND **Down-Trend** **(NOTE: the rate of increasing/decreasing of trend in reality is either steady, high OR low!).** Let us understand both the terms through a real-life example. Adidas manufactures the Argentina FIFA World-Cup Football Jersey, and so is Nike for France. When the Final Match approached, the sales of Adidas & Nike apparels soared exponentially – This is called "**UP-TREND**". Now, as FIFA World-Cup is finished, those apparels (Jerseys) are being bought at a lower frequency – this is called "**DOWN-TREND**".

3. **SEASONALIY:** Let us take an example. We are taking the sales data of Air Conditioners throughout a year. After some analysis, we figured out that during the summer season, sales of A/C units increase, but in the same year during the winter season, the units are bought in lower frequency – and it repeats every single year. What I mean to say is that, when we explore data – we see some interesting patterns as in increasing direction OR decreasing direction – which repeats over a fixed span of time (like here, say 1 year) – then we say that the data has **seasonality** feature.

4. **CYCLIC PATTERNS:** When you observe that the patterns in data are getting repeated after a longer span of time – then the patterns are called "**CYCLIC PATTERNS**". **For ex.:** Lok-Sabha election is held every 5 years. When election is closing fast, the effect can be seen on stock prices – so this is a cyclic pattern of the data of stocks of various organizations.

5. **NOISE:** Sometimes, data do not show any pattern w.r.t. time. They rather show randomness in values. Those kinds of data are known as "**NOISE**".

TIME-SERIES ANALYSIS

Studying the features of the response variable about time, as the independent variable, is done through "Time Series Analysis" (in short it is also known as TSA). TSA is the foundation for forecasting analysis and prediction, specifically for time-based issue statements like:

- Examining the trends in the historical dataset
- Recognizing and comparing patterns derived from the previous stage with the current circumstance.
- Recognising the factor(s) impacting a certain variable(s) at various times.

TIME-SERIES ALGORITHMS

Now that we have a clear idea about "Time-Series" and data needed for "Time-Series Analysis", it's time for discussing some prominent algorithms (used in this project) which are unique in their approach but share the same purpose.

1□ SHORT-TERM MEMORY (STM) NEURAL NETWORK – RNN

Let's talk a little bit about "feed-forward" neural network –

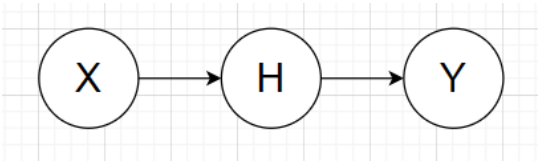


Fig. 1 : Basic Neural Network Layers

If you observe, there are no loops and only the current input state (X) is only considered. Thus, there is no scope of having a "memory" - that is why they cannot handle sequential data (because analysis of sequential data requires the consideration of previous inputs – that must be stored in a memory (let's say, a memory layer)). Thus, RNN (Recurrent Neural Network) inculcates the concept by including a memory layer in its working – hence, it will consider the current input state as well as previous input states. Below is a raw representation of RNN:

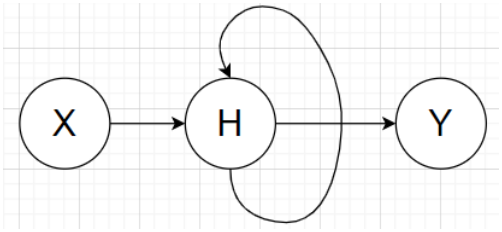


Fig. 2 : Basic Recurrent Neural Network (RNN) representation

So, the output that came out from the current step will act as the input for the next step – and it will keep on repeating, depending on the no. of layers included, i.e. , density of the model created. Now, let us expand the above diagram for better understanding –

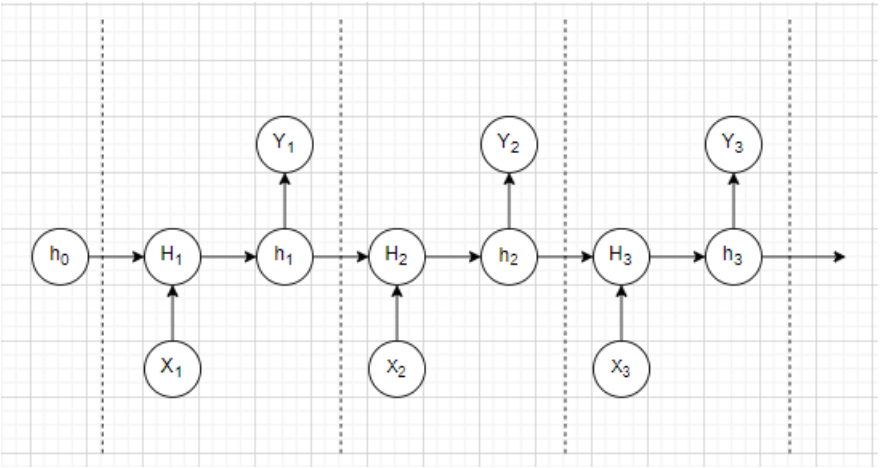


Fig. 3 : Detailed layer-representation of RNN

Here, $\mathbf{X_t}$ is considered as input in the given timestep (t). If you observe in the above diagram, every timestep can be represented as the components residing between 2 dotted lines. As I had mentioned earlier, RNN considers previous input states as well – but when talking about the first state – (Here, $\mathbf{h_0}$): consider it as 0 (ZERO) – is a hidden state. When fed with the first input state (X_1) to H_1, it gives out the current hidden state (h_1) and gives out the output Y_1 – and the cycle goes on. For the next hidden state (h_2), we will consider h_1 as the **previous hidden state.** Hence, we can figure out the next hidden state (h_t) can be figured out from previous hidden state (h_{t-1}) and input state (X_t). Thus, we can draft a formula –

$$\mathbf{h_t = f\,(h_{t-1}, X_t)}$$

The above formula will replace all the $\mathbf{H_t}$ (which is the substitute term for the formula above). Now, if you observe, the above function is used for every input state – hence the name is **"Recurrent". $\mathbf{h_t}$** can also be figured out using the following formula –

$$\mathbf{h_t = \tanh\,(W_{hh} \cdot h_{t-1} + W_{Xh} \cdot X_t)}$$

where, the tanh function is mostly used to classify data into two groups, using the graph of tanh function (where **y** value (f(x)) is searched for **x**), followed by the following formula –

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Also, $\mathbf{W_{hh}}$ is the weight at previous hidden state, and $\mathbf{W_{Xh}}$ is the weight at current hidden state (consider the edges as weights)

We also need to find the output (Y_t) as well. For that we will use the following formula –

$$\mathbf{Y_t = W_{hy} \cdot h_t}$$

where, $\mathbf{W_{hy}}$ is the weight at the output state.

2□ LONG SHORT-TERM MEMORY (LSTM) NEURAL NETWORK

This algorithm evolved from its predecessor – Short Term Memory (also known as STM). STM has the problem of vanishing "Gradient". To those who do not know, "Gradient" is the value used to update a neural network's weight –

Gradient Update Rule

New weight = initial weight – learning rate * gradient

The "vanishing gradient problem" is when a gradient strength as its back-propagates through time.

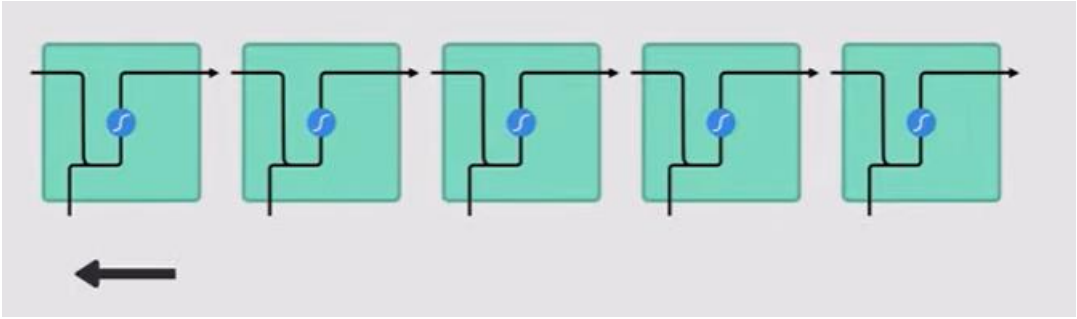


Fig. 4 : Previous memory representation of RNN with vanishing gradient

If a gradient's value becomes very small, it won't contribute much to learning. (Usually talking about the earlier layers). Hence, in longer sequences, the model won't learn much (thus, having a short-term memory issue). Hence, LSTM was created as a solution to Short-Term Memory. Now, LSTM is mainly comprised of some internal mechanisms (also called gates) and some functions which are responsible to regulate the flow of data. These gates have the capability to learn which information to keep for future use and which ones to

discard – for efficient learning. Below is a pictorial representation what STM does with the data it ingests –

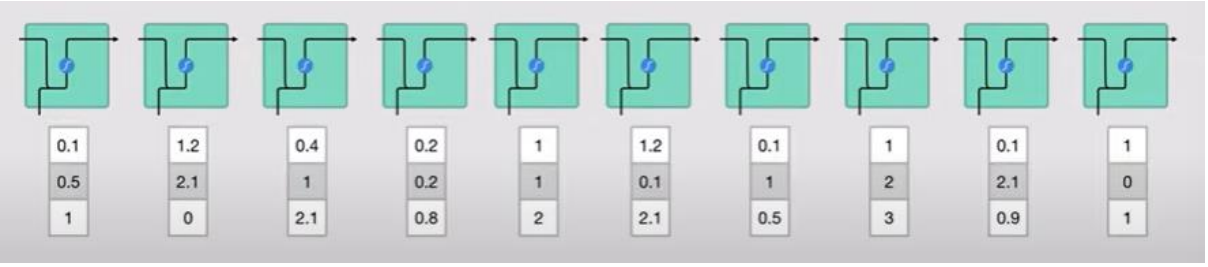


Fig. 5 : Working of RNN with numerically represented strings.

Suppose the model ingested a sentence, then each word is converted to machine readable vectors then sent each of the vectors one by one into the model's function.

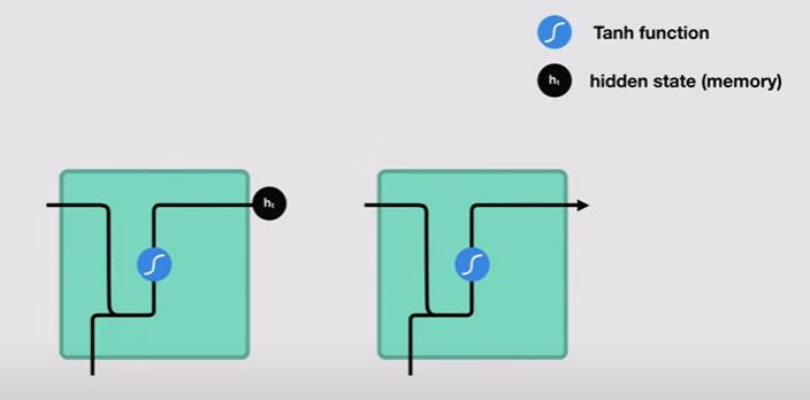


Fig. 6(A) : Working of RNN (1)

While processing, it passes the previous head state to next step of the sequence. The hidden state holds the information of neural network's memory – what the model saw earlier. Let us dive a little deep. Below is another picture which explains how the hidden state is calculated. The input in the previous hidden state is combined to form a vector – which has information on the current inputs on the previous inputs.

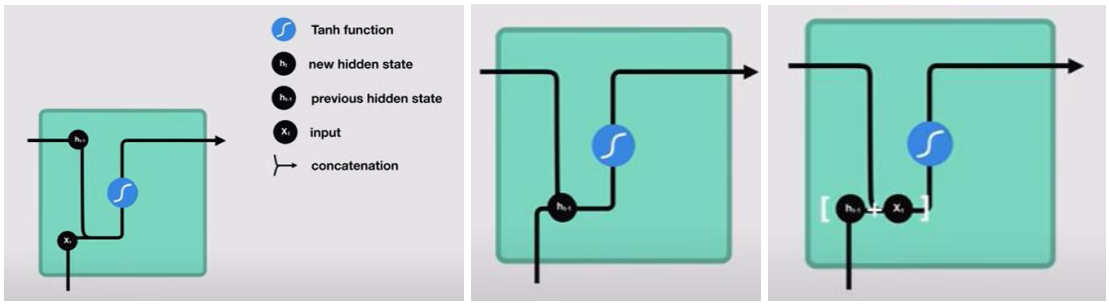


Fig. 6(B) : Working of RNN (2)

The vector goes through the tanh activation and the output is the new hidden state OR the memory of the network.

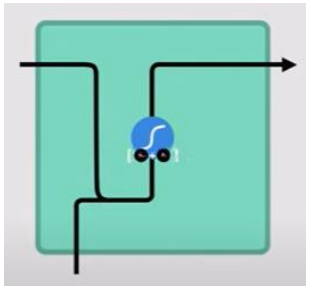


Fig. 6(C) : Working of RNN (3)

tanh activation function: It's used to help regulate the values flowing through the network. The function squishes values to always be between -1 and +1.

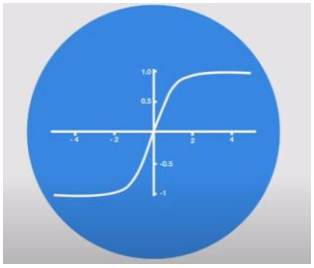


Fig. 7 : tanh activation function

If we keep on multiplying a set of numbers (let's say (15, 0.03, -5.44)) by a positive number (say 6) – some values can be very large, leading other values seem insignificant. On the other hand, the tanh function ensures that the values should lie between +1 and -1 – thus, regulated.

Now let us talk about **LSTM**. Below is a picture that shows the insides of the algorithm –

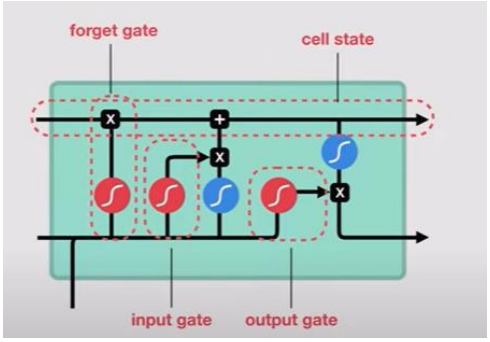


Fig. 8 : LSTM model representation (single layer)

The control flow of LSTM is same as STM – ingests and processes data sequentially and passing on information as it propagates forward. So where is the difference between STM and LSTM? The difference lies in the operations of LSTM Cells. These operations are used to allow LSTM to forget OR keep information. The cell state is like a highway, transferring relative information all the way down to the sequence chain (like memory of network) – from first to last time step (Thus, reducing the effects of STM). The dirtiness of added data keeps on getting cleaned because of the gates. Each gate themselves are different neural networks – which decides which information can be allowed to stay and which ones to discard during training. Gates contains **sigmoid** activations – which are similar to **tanh** – the only difference – sigmoid squishes values into 0 or 1, unlike tan (-1 and +1).

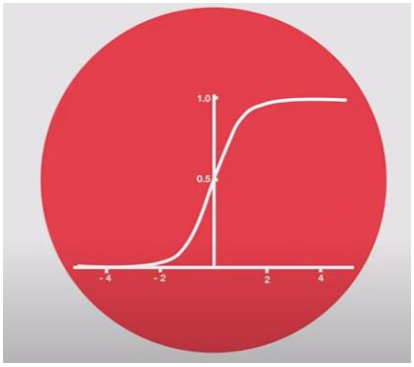


Fig. 9 : Sigmoid activation function

ADVANTAGE : any number multiplied by 0 will be 0, so can be forgotten easily. On the other hand , value multiplied by 1 will remain the same (so, these will be the data to be kept).

There are 3 different gates – regulating the information flow. The gates are **Input gate, Forget gate** and **Output gate**. **FORGET GATE** is responsible for throwing away / disposing the information which are not required. Information from previous hidden state and information from the current input is passed into sigmoid function. The value that will come up from the function will be either 0 or 1 (0 means forget, 1 means keep it).

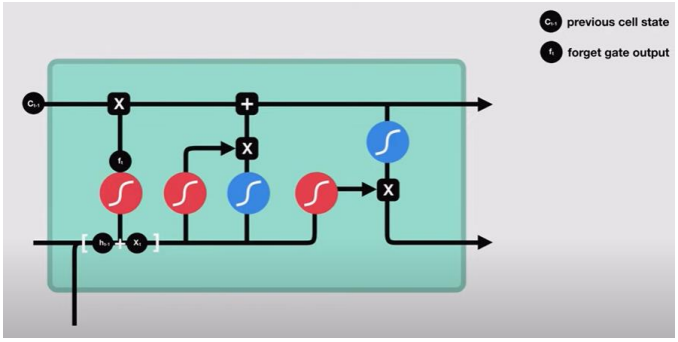


Fig. 10(A) : Working of LSTM(1)

INPUT GATE updates the cell state. First, we pass the previous hidden state and the current input to the sigmoid function, which decides which values will be updated – by transforming values to be between 0 and 1 (0 means unimportant, 1 means important). The hidden state & current input are also passed through the tanh function which will give out values b/w -1 and +1 to regulate the network. The output from sigmoid function and tanh function are then multiplied. The sigmoid output will decide which output from tanh function will be kept for the cell state –

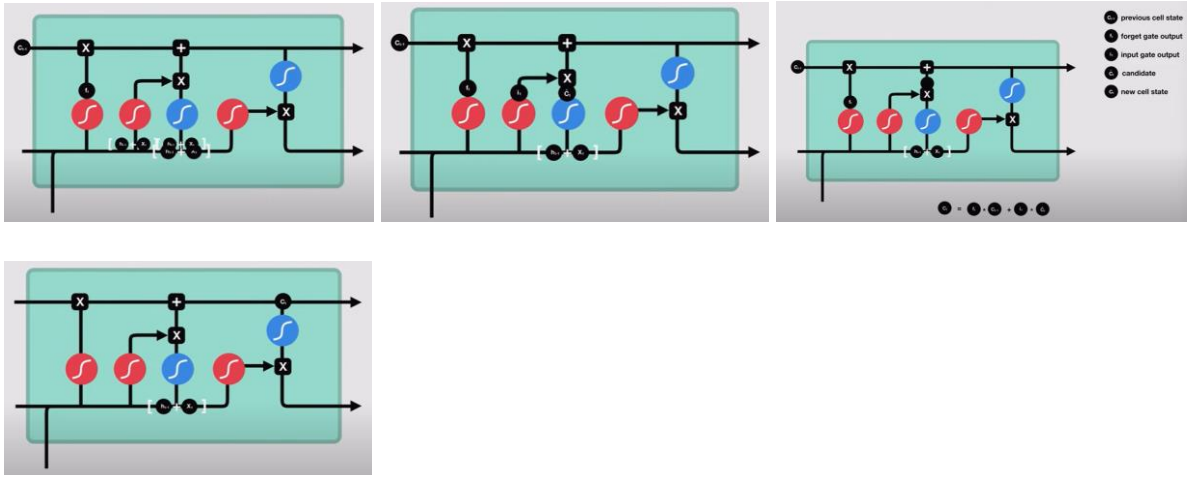


Fig. 10(B) : Working of LSTM(2)

OUTPUT GATE decides what the next hidden state should be. Remember that the hidden state contains information of previous inputs. The previous hidden state and the current input are sent to sigmoid function and then pass the newly created/modified cell state to the tanh function. We then multiply the Sigmoid o/p with the tanh output and then decides what information should the hidden state (the output) carry, and the cell state is continued to next stage of the sequence.

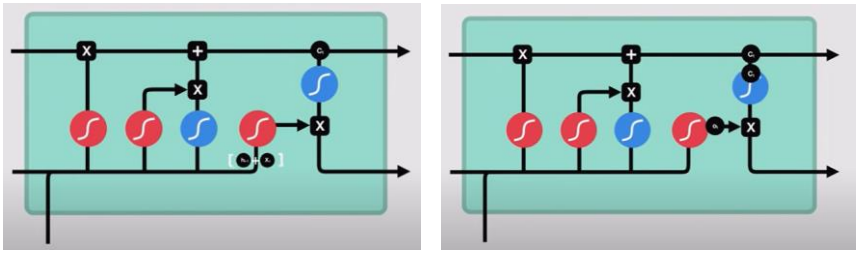
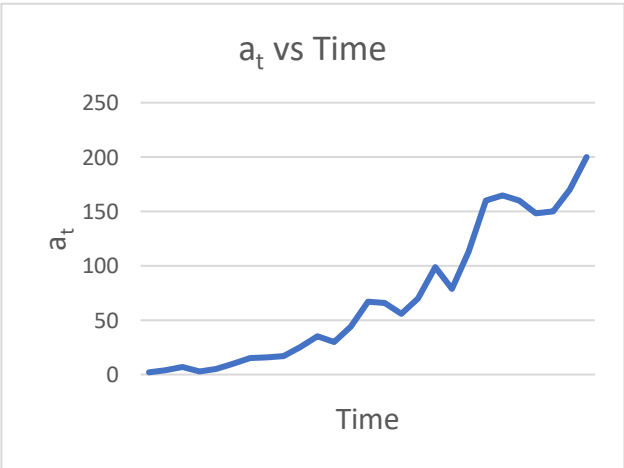


Fig. 10(C) : Working of LSTM(3)

3□ AUTO-REGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA) NEURAL NETWORK

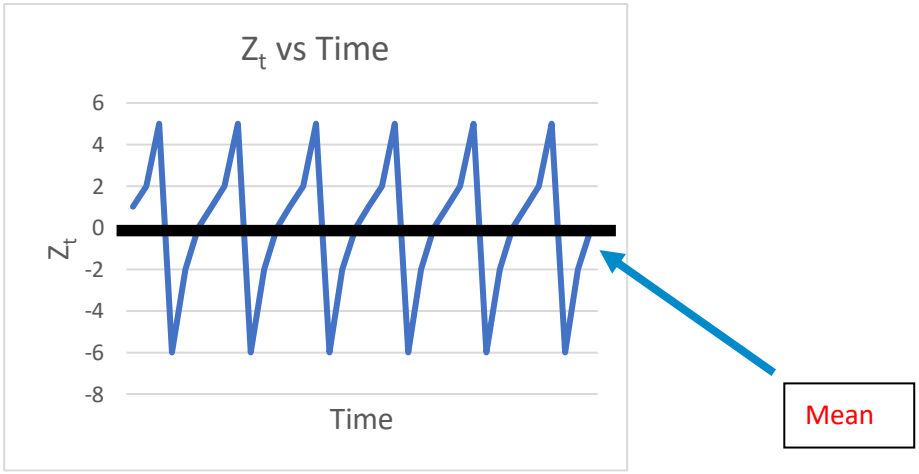
Let us start the explanation using a hypothetical example. Imagine that you are a salesman of "anchors" – for boats. Every month you sell a certain number of anchors – denoted by $\mathbf{a_t}$ (where 't' denotes the time-period). Now you want to predict the number of anchors you are expecting to be sold next month. So, for analysis, you will be needing a graph – "**a_t vs time**" where $\mathbf{a_t}$ is on Y-axis (the output) and time is an X-axis (that keeps on going forward) – in the graph (hypothetical). You saw a pattern –



But the above data doesn't seem to have depicting the quality of stationarity – which means that the time-series needs to have a constant mean, constant variance over time and has no seasonality. Here if you observe, you will not see constant mean because the pattern seems like increasing. **ARIMA** solves the above issue as they can handle situations where everything is stationary except the **MEAN**. So here, instead of predicting time-series itself, we will be predicting differences of time-series from one timestamp to the previous time-stamp – thus we are going to create a new time-series (called $\mathbf{Z_t}$) –

$$\mathbf{Z_t = a_{t+1} - a_t}$$

Let us say that the above formula works like this – $\mathbf{a_{t+1}}$ consists of anchor sold values of the month of February, and a_t denotes the same for January. Putting these values to the above formula will provide you with one time-point of the new series (known as $\mathbf{Z_t}$). If you check the graph, it seems linear with some noise (as it is not actually a straight line, but we can make that out the same by assuming it, that too inculcated with some noise). Now, for any linear function – if we want to go from one point to another – we just must add a 'constant' value. If it's so, then the difference between the current and the previous timestamp should tending to be constant (may be hovering around some constant – see the below pic) –



Now that we have obtained a constant mean, thus the issue of the same is also solved – with no seasonality. Now we are interested in finding out the difference between the consecutive time-points of anchor sales. So mathematically – The formula shows the concept of Auto-Regressive (AR) components – as we are making out the function of a variable dependent on the previous value of a factor to get the output as the same variable dependent on the current value of the factor (here time).

$$Z_t = \phi_1 * Z_{t-1} + \varnothing_1 * \varepsilon_{t-1} + \varepsilon_t$$

AR MA

ϕ_1 is the coefficient. So, $\varnothing_1 * \varepsilon_{t-1}$ is the moving average (MA) bit and ε_t is the error in the current time period. The 'Integrated' component is taken care by Z_t (the difference of consecutive points we originally started with). But we are not winding up by predicting Z_t – we are here to predict the no. of anchors we can sell (denoted by a_k). Let us assume that the last point (t) of the sales graph be **t = l → a_l. So, to calculate a_k –**

$$a_k = Z_{k-1} + a_{k-1} = Z_{k-1} + a_{k-2} = \ldots \infty$$

Formulating the formula (generalization):

$$\sum_{i=1}^{i=l} Z_{k-i} + a_l$$

We stop at a_l because this will be the last point on the graph. Thus, we will get to know how many anchors are sold (a_k) in a certain time-period (**k**)- eventually will predict many anchors will be sold in the next month.

Chapter 5

Implementation & Testing

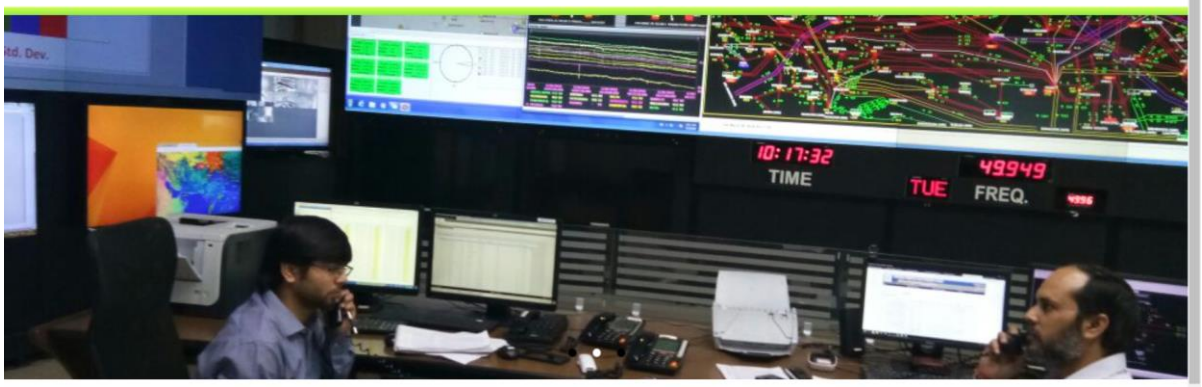
5.1 DATA SET

The dataset acquired here is an ensemble of many other datasets. The acquisition of all of them depends on the powerplant I am choosing. For the aforementioned, I visited the website of **Northern Regional Power Committee – NRPC (LINK: http://164.100.60.165/)** , which is regulated by the Ministry of Power, Government of India.

In this website, one can get information based on the division of power production and consumption among various states (NOTE: for a power plant, the clients are always the State Government(s)). There are files by the name, **"Provisional REA for the month of <month>, <year>"**, where **REA** stands for **Reactive Energy Account**. This file contains a table, consisting of information – which power plant caters which state (NOTE: one state can be catered by numerous powerplants and vice-versa). From there, I chose a powerplant whose energy production and client consumption values can be acquired from another website known as, **GRID CONTROLLER OF INDIA Ltd.** , earlier known as, **POSOCO** – Power System Operation Corporation Ltd. **(LINK : https://wbes.nrldc.in).** There is a page called **"Full Schedule" (LINK : https://wbes.nrldc.in/ReportFullSchedule),** where one can set different parameters to get the desired energy data.

GRID CONTROLLER OF INDIA LIMITED
Formerly known as Power System Operation Corporation Limited (POSOCO)
(A Government of India Enterprise) CIN: U40105DL2009GOI188682
Northern Regional Load Despatch Centre

View Reports Login



GRID-INDIA Scheduling Reports Schedule Revision 92 Login

Report

Schedule » Full Schedule

Full Schedule Details

Date: 03-03-2023 Region: NORTH Profile: Injection

Seller: UNCHAHAR4 Revision No: 89 View: Seller

Created On: 03-03-2023 09:02:35 Remarks: ConsoleApplication Show data

Time Block	Time Desc	UNCHAHAR4	Grand Total
1	00:00-00:15	468.754428	468.754428
2	00:15-00:30	468.750001	468.750001
3	00:30-00:45	468.750001	468.750001
4	00:45-01:00	468.750001	468.750001
5	01:00-01:15	468.753056	468.753056
6	01:15-01:30	468.750000	468.750000
7	01:30-01:45	468.750000	468.750000
8	01:45-02:00	468.760000	468.760000
9	02:00-02:15	468.760000	468.760000
10	02:15-02:30	468.757803	468.757803
11	02:30-02:45	404.800000	404.800000
12	02:45-03:00	332.800000	332.800000
13	03:00-03:15	260.800000	260.800000
14	03:15-03:30	271.590000	271.590000
15	03:30-03:45	343.590000	343.590000

Report

Schedule » Full Schedule

Full Schedule Details

Date: 03-03-2023 Region: NORTH Profile: Drawal

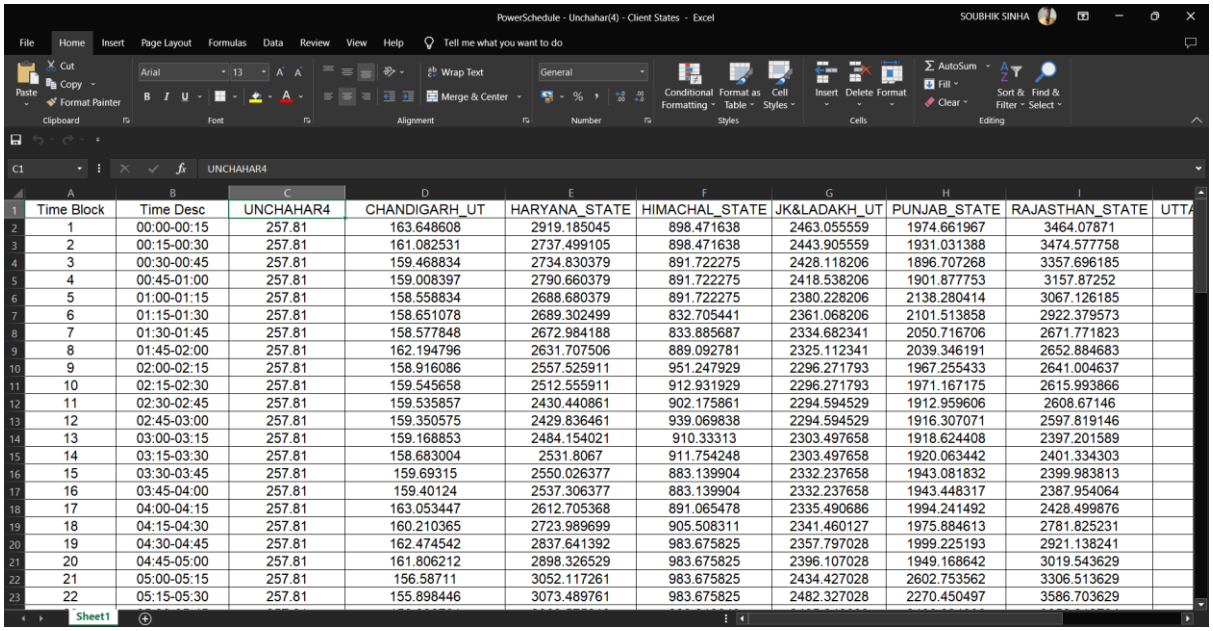
Buyer: UTTARPRADESH_... Revision No: 89 View: Buyer

Created On: 03-03-2023 09:02:35 Remarks: ConsoleApplication Show data

Time Block	Time Desc	UTTARPRADESH_STATE	Grand Total
1	00:00-00:15	6202.089727	6202.089727
2	00:15-00:30	6124.677657	6124.677657
3	00:30-00:45	6148.266303	6148.266303
4	00:45-01:00	6126.250445	6126.250445
5	01:00-01:15	6117.688866	6117.688866
6	01:15-01:30	6129.218866	6129.218866
7	01:30-01:45	6126.687742	6126.687742
8	01:45-02:00	6144.697742	6144.697742
9	02:00-02:15	5726.572929	5726.572929
10	02:15-02:30	5535.154636	5535.154636
11	02:30-02:45	5804.021681	5804.021681
12	02:45-03:00	5815.618887	5815.618887
13	03:00-03:15	5879.018887	5879.018887
14	03:15-03:30	5906.468887	5906.468887
15	03:30-03:45	5888.375886	5888.375886

https://wbes.nrldc.in/ReportFullSchedule#

The final dataset after combining the energy production data of powerplant (here, UNCHAHAR4) and energy consumption data of various states will look like this (NOTE : I am showing a cross-section of the dataset – the first few rows) –



PowerSchedule - Unchahar(4) - Client States - Excel

Time Block	Time Desc	UNCHAHAR4	CHANDIGARH_UT	HARYANA_STATE	HIMACHAL_STATE	JK&LADAKH_UT	PUNJAB_STATE	RAJASTHAN_STATE	UTTA
1	00:00-00:15	257.81	163.648608	2919.185045	898.471638	2463.055559	1974.661967	3464.07871	
2	00:15-00:30	257.81	161.082531	2737.499105	898.471638	2443.905559	1931.031388	3474.577758	
3	00:30-00:45	257.81	159.468834	2734.830379	891.722275	2428.118206	1896.707268	3357.696185	
4	00:45-01:00	257.81	159.008397	2790.660379	891.722275	2418.538206	1901.877753	3157.87252	
5	01:00-01:15	257.81	158.558834	2688.680379	891.722275	2380.228206	2138.280414	3067.126185	
6	01:15-01:30	257.81	158.651078	2689.302499	832.705441	2361.068206	2101.513858	2922.379573	
7	01:30-01:45	257.81	158.577848	2672.984188	833.885687	2334.682341	2050.716706	2671.771823	
8	01:45-02:00	257.81	162.194796	2631.707506	889.092781	2325.112341	2039.346191	2652.884683	
9	02:00-02:15	257.81	158.916086	2557.525911	951.247929	2296.271793	1967.255433	2641.004637	
10	02:15-02:30	257.81	159.545658	2512.555911	912.931929	2296.271793	1971.167175	2615.993866	
11	02:30-02:45	257.81	159.535857	2430.440861	902.175861	2294.594529	1912.959606	2608.67146	
12	02:45-03:00	257.81	159.350575	2429.836461	939.069838	2294.594529	1916.307071	2597.819146	
13	03:00-03:15	257.81	159.168853	2484.154021	910.33313	2303.497658	1918.624408	2397.201589	
14	03:15-03:30	257.81	158.683004	2531.8067	911.754248	2303.497658	1920.063442	2401.334303	
15	03:30-03:45	257.81	159.69315	2550.026377	883.139904	2332.237658	1943.081832	2399.983813	
16	03:45-04:00	257.81	159.40124	2537.306377	883.139904	2332.237658	1943.448317	2387.954064	
17	04:00-04:15	257.81	163.053447	2612.705368	891.065478	2335.490686	1994.241492	2428.499876	
18	04:15-04:30	257.81	160.210365	2723.989699	905.508311	2341.460127	1975.884613	2781.825231	
19	04:30-04:45	257.81	162.474542	2837.641392	983.675825	2357.797028	1999.225193	2921.138241	
20	04:45-05:00	257.81	161.806212	2898.326529	983.675825	2396.107028	1949.168642	3019.543629	
21	05:00-05:15	257.81	156.58711	3052.117261	983.675825	2434.427028	2602.753562	3306.513629	
22	05:15-05:30	257.81	155.898446	3073.489761	983.675825	2482.327028	2270.450497	3586.703629	

This dataset will be loaded to Google Drive, so that it can directly be imported in Google Colab Environment by mounting the Drive.

5.2 CODE & OUTPUT

First, I will start with common implementation components of LSTM & RNN –

The libraries imported (see below) are not limited till here; when I will be creating ML models, more libraries will be imported. On the initial run, I am importing Numpy, Pandas, Seaborn & Matplotlib to handle the dataset.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Importing Essential Modules / Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

After the mounting of Google Drive in the Google Colab environment, we need to have visuals of top rows of the dataset to have a look of the attributes and their value-types.

```
# Importing Dataset

df_power = pd.read_csv("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/CSV_PowerSchedule - Unchahar(4) - Client States.csv")
df_power
```

	Time Block	Time Desc	UNCHAHAR4	CHANDIGARH_UT	HARYANA_STATE	HIMACHAL_STATE	JK&LADAKH_UT	PUNJAB_STATE	RAJASTHAN_STATE	UTTARAKHAND_STATE	UTTARPRADESH_STATE
0	1	00:00-00:15	257.81	163.648608	2919.185045	898.471638	2463.055559	1974.661967	3464.078710	1227.060154	5707.888517
1	2	00:15-00:30	257.81	161.082531	2737.499105	898.471638	2443.905559	1931.031388	3474.577758	1201.031663	5118.275335
2	3	00:30-00:45	257.81	159.468834	2734.830379	891.722275	2428.118206	1896.707268	3357.696185	1159.794583	4695.923110
3	4	00:45-01:00	257.81	159.008397	2790.660379	891.722275	2418.538206	1901.877753	3157.872520	1153.094583	4785.328445
4	5	01:00-01:15	257.81	158.558834	2688.680379	891.722275	2380.228206	2138.280414	3067.126185	1178.954583	4997.936018
...	...	...	...	...	...	...	...	...	...	...	...
91	92	22:45-23:00	257.81	199.157260	3144.174897	1062.071746	2711.687438	2075.055469	3954.582035	1393.394830	6690.249163
92	93	23:00-23:15	257.81	197.907556	3149.252781	957.793316	2661.538508	2227.478591	3950.086806	1348.067944	5881.705452
93	94	23:15-23:30	257.81	198.145115	3144.109508	930.094679	2607.500621	2235.257041	3858.999919	1281.391341	5719.366297
94	95	23:30-23:45	257.81	169.357366	3132.282000	960.620107	2630.788986	2501.045239	3567.664726	1247.696984	5731.153531
95	96	23:45-24:00	257.81	163.921009	3088.459865	958.156168	2553.115742	2442.235814	3466.847376	1264.063223	5548.987368

96 rows × 11 columns

The general information of a dataset reveals whether any values under any attribute consists of data-type equivalent to ‘NA’ (Not Available) or ‘NULL’.

```
# let us see the General information of the dataset

df_power.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Time Block          96 non-null     int64  
 1   Time Desc           96 non-null     object 
 2   UNCHAHAR4           96 non-null     float64
 3   CHANDIGARH_UT       96 non-null     float64
 4   HARYANA_STATE       96 non-null     float64
 5   HIMACHAL_STATE      96 non-null     float64
 6   JK&LADAKH_UT        96 non-null     float64
 7   PUNJAB_STATE        96 non-null     float64
 8   RAJASTHAN_STATE     96 non-null     float64
 9   UTTARAKHAND_STATE   96 non-null     float64
 10  UTTARPRADESH_STATE  96 non-null     float64
dtypes: float64(9), int64(1), object(1)
memory usage: 8.4+ KB
```

In data filtering, I am filtering out those attributes (here, the states which are not consuming significant power from the powerplant). For ‘UNCHAHAR4’, the state of **Rajasthan** and **Uttar Pradesh** are the 2 major clients – hence, I will keep those attributes. Afterwards, the dataset will be splitted into training & testing data in the ration of 3:1 (OR 75% : 25%).

```
# Now, we create a new dataframe with all the required I/P attributes

# Here, we are more interested in witnessing the power demand for the state
# of Uttar Pradesh(48.238 %) and Rajasthan (23.378 %) - MAJOR BUYERS !
# SOURCE : Northern Regional Power Committee (Govt. of India)

data = df_power.filter(['UNCHAHAR4' , 'UTTARPRADESH_STATE' , 'RAJASTHAN_STATE'])

# Dataframe to Numpy Array Conversion
dataset = data.values

# The model should be knowing - how many rows we are taking
# for training (Here , the split will be 0.75 and 0.25)
training_data_len = int(np.ceil(len(dataset) * (0.75)))
training_data_len
```

```
72
```

ML models works best when the values ingested by them are considerably smaller than real life values (best, if the values hover between 0 and 1). Hence, I must transform the data by

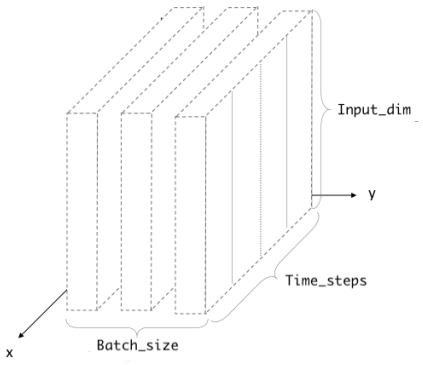
scaling them into values (decimal), that will lie between 0 and 1 (inclusive). Thus, I will be using **MinMaxScaler** for the same, as it maintains the original distribution's form.

```
# Data Scaling

from sklearn.preprocessing import MinMaxScaler

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))
scaled_data_x = scaler_x.fit_transform(dataset)

df_power_y = df_power[int(training_data_len) : ]
scaled_data_y = scaler_y.fit_transform(df_power_y.filter(['UNCHAHAR4']).values)

print(scaled_data_x[0:5])
print()
print(scaled_data_y[0:5])
```

```
[[0.         0.25230898 0.33354039]
 [0.         0.15047414 0.33679453]
 [0.         0.07752772 0.30056756]
 [0.         0.09296934 0.23863305]
 [0.         0.12968978 0.21050659]]

[[0.99995259]
 [0.99995259]
 [1.        ]
 [1.        ]
 [0.99996445]]
```

```
# Now that we have scaled our data - its time to know about the data that has
# to be fed to the [redacted] model. The model accepts data in 3D array.
# Thus , we will convert our scaled data into Numpy Array
# Later the array will be converted to 3D array

# Timesteps taken = 10 (VERY IMPORTANT )
```

You must always supply a three-dimensional array to your LSTM network as an input, where the first dimension is the batch size, the second is the number of time steps, and the third is the number of units that make up an input sequence. The data that is fed into the LSTM or RNN is shown in the following diagram.



Fog. 11 : Structure of 3-dimensional data

Here, I am taking the value of timesteps as 10.

Now that the timesteps are set, it's time to split our data components into **training** (**x_train & y_train**) and **testing** (**x_test & y_test**) data. Remember that model training will be done on **x_train & y_train**, predictions will be based on **x_test**, and model evaluation will be based on **y_test**.

```
# Training dataset creation and split (x_train , y_train)

train_data = scaled_data_x[0:int(training_data_len), :]

x_train = []
y_train = []

for i in range(10, len(train_data)):
  x_train.append(train_data[i-10:i, :])
  y_train.append(train_data[i,1])
  if i<=111:
    print(x_train)
    print(y_train)
    print()

# We have to convert x_train and y_train into Numpy Arrays
x_train , y_train = np.array(x_train) , np.array(y_train)
```

```
Streaming output truncated to the last 5000 lines.
       [0.99996247, 0.57629672, 0.97925607],
       [0.99996247, 0.60928543, 0.98852567],
       [0.99996247, 0.64610684, 0.9825616 ]]), array([[0.        , 0.31306641, 0.57925156],
       [0.34131648, 0.33787088, 0.8570207 ],
       [0.68263296, 0.33582486, 0.9372667 ],
       [0.99996247, 0.35782025, 0.62686986],
       [0.99996247, 0.39606151, 0.46274457],
       [0.99996247, 0.5433077 , 0.95584521],
       [0.99996247, 0.57629672, 0.97925607],
       [0.99996247, 0.60928543, 0.98852567],
       [0.99996247, 0.64610684, 0.9825616 ],
       [0.99996247, 0.70437006, 1.        ]]), array([[0.34131648, 0.33787088, 0.8570207 ],
       [0.68263296, 0.33582486, 0.9372667 ]
```

```
[ ]  1  # Creating Testing data
     2  test_data = scaled_data_x[int(training_data_len) - 10 : , :]
     3
     4  # Creating x_test and y_test
     5  x_test = []
     6  y_test = dataset[int(training_data_len) : , 1]
     7  for i in range(10, len(test_data)):
     8    x_test.append(test_data[i-10:i , :])
     9
    10  # Converting to Numpy Array
    11  x_test = np.array(x_test)
```

From here onwards, both the models will have different implementation and shall show unique values –

1□ IMPLEMENTATION OF LSTM MODEL

Now, I will create the model of LSTM. For that I will import **KERAS** module of Python library. As mentioned earlier, LSTM will take 3D data (**batch_size, timesteps** & **input_shape** OR **input_dim**). Batch Size is up to us, how one wants to set. Here, I have taken 20 as Batch Size. Also, I have taken 1000 epochs (1 epoch = whole dataset will pass through the model one time).

```
[ ]  1  # LET US CREATE THE LSTM MODEL 💥
```

```
[ ]  1  from keras.models import Sequential
     2  from keras.layers import Dense , LSTM
     3
     4  # Model Building
     5  model_lstm = Sequential()
     6
     7  model_lstm.add(LSTM(64, return_sequences=True, input_shape = (x_train.shape[1] , 3)))
     8  model_lstm.add(LSTM(64, return_sequences= False))
     9  model_lstm.add(Dense(32))
    10  model_lstm.add(Dense(1))
    11
    12  # Model Compilation
    13  model_lstm.compile(optimizer='adam', loss='mean_squared_error')
    14
    15  # Model Training
    16  # Then fit into the model
    17  model_lstm.fit(x_train, y_train, batch_size=20, epochs=1000)
```

```
Epoch 1/1000
4/4 [==============================] - 10s 14ms/step - loss: 0.1743
Epoch 2/1000
4/4 [==============================] - 0s 14ms/step - loss: 0.0343
Epoch 3/1000
4/4 [==============================] - 0s 14ms/step - loss: 0.0426
Epoch 4/1000
4/4 [==============================] - 0s 14ms/step - loss: 0.0098
Epoch 5/1000
4/4 [==============================] - 0s 13ms/step - loss: 0.0190
```

2□ IMPLEMENTATION FOR RNN MODEL

The implementation of RNN is a bit different because we have to add a number of layers and also include an output layer and a hidden layer at the starting (the hidden layer at the first will be equivalent to ZERO, as no memory will be contained inside it). Rest the code remains the same as LSTM.

```
# LET US CREATE THE RNN MODEL

# Importing essential libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout

# RNN Initialization
model_rnn = Sequential()

# adding first RNN layer and dropout regulatization
model_rnn.add(SimpleRNN(units = 50,activation = "tanh", return_sequences = True , input_shape = (x_train.shape[1],3)))
model_rnn.add(Dropout(0.2))

# adding second RNN layer and dropout regulatization
model_rnn.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True))
model_rnn.add(Dropout(0.2))
```

```
# adding third RNN layer and dropout regulatization
model_rnn.add(SimpleRNN(units = 50,activation = "tanh", return_sequences = False))
model_rnn.add(Dropout(0.2))

# adding the output layer
model_rnn.add(Dense(units = 1))

# compiling RNN

model_rnn.compile(optimizer = "adam", loss = "mean_squared_error")

# fitting the RNN
model_rnn.fit(x_train, y_train, epochs = 1000, batch_size = 32)
```

```
Epoch 1/1000
2/2 [==============================] - 3s 15ms/step - loss: 1.2357
Epoch 2/1000
2/2 [==============================] - 0s 18ms/step - loss: 0.3795
Epoch 3/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.5037
Epoch 4/1000
```

□ Creating ARIMA Model

The implementation of ARIMA Model is way different of what we saw in LSTM & RNN. Here, I will be importing many other library functions apart from what we saw for the aforementioned.

Loading necessary Libraries / Packages

```
[ ]
import pandas as pd
from pandas.plotting import autocorrelation_plot
from pandas import DataFrame
from pandas import concat
import numpy as np
from math import sqrt

from sklearn.metrics import mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.arima_model import ARIMA
# from scipy.stats import boxcox
from scipy.special import boxcox, inv_boxcox

import seaborn as sns
sns.set_style("whitegrid")
import matplotlib.pyplot as plt
```

```
from matplotlib.pylab import rcParams
from matplotlib import colors
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Mounting the Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Loading and Extracting Appropriate Data

```
[ ]
df_power = pd.read_csv("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/CSV_PowerSchedule - Unchahar(4) - Client States.csv")
df_power
```

	Time Block	Time Desc	UNCHAHAR4	CHANDIGARH_UT	HARYANA_STATE	HIMACHAL_STATE	JK&LADAKH_UT	PUNJAB_STATE	RAJASTHAN_STATE	UTTARAKHAND_STATE	UTTARPRADESH_STATE
0	1	00:00-00:15	257.81	163.648608	2919.185045	898.471638	2463.055559	1974.661967	3464.078710	1227.060154	5707.888517
1	2	00:15-00:30	257.81	161.082531	2737.499105	898.471638	2443.905559	1931.031388	3474.577758	1201.031663	5118.275335
2	3	00:30-00:45	257.81	159.468834	2734.830379	891.722275	2428.118206	1896.707268	3357.696185	1159.794583	4695.923110
3	4	00:45-01:00	257.81	159.008397	2790.660379	891.722275	2418.538206	1901.877753	3157.872520	1153.094583	4785.328445
4	5	01:00-01:15	257.81	158.558834	2688.680379	891.722275	2380.228206	2138.280414	3067.126185	1178.954583	4997.936018
...	...	...	...	...	...	...	...	...	...	...	...
91	92	22:45-23:00	257.81	199.157260	3144.174897	1062.071746	2711.687438	2075.055469	3954.582035	1393.394830	6690.249163
92	93	23:00-23:15	257.81	197.907556	3149.252781	957.793316	2661.538508	2227.478591	3950.086806	1348.067944	5881.705452
93	94	23:15-23:30	257.81	198.145115	3144.109508	930.094679	2607.500621	2235.257041	3858.999919	1281.391341	5719.366297
94	95	23:30-23:45	257.81	169.357366	3132.282000	960.620107	2630.788986	2501.045239	3567.664726	1247.696984	5731.153531
95	96	23:45-24:00	257.81	163.921009	3088.459865	958.156168	2553.115742	2442.235814	3466.847376	1264.063223	5548.987368

96 rows × 11 columns

```
[ ]  1  # let us see the General information of the dataset
     2
     3  df_power.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Time Block          96 non-null     int64
 1   Time Desc           96 non-null     object
 2   UNCHAHAR4           96 non-null     float64
 3   CHANDIGARH_UT       96 non-null     float64
 4   HARYANA_STATE       96 non-null     float64
 5   HIMACHAL_STATE      96 non-null     float64
 6   JK&LADAKH_UT        96 non-null     float64
 7   PUNJAB_STATE        96 non-null     float64
 8   RAJASTHAN_STATE     96 non-null     float64
 9   UTTARAKHAND_STATE   96 non-null     float64
 10  UTTARPRADESH_STATE  96 non-null     float64
dtypes: float64(9), int64(1), object(1)
memory usage: 8.4+ KB
```

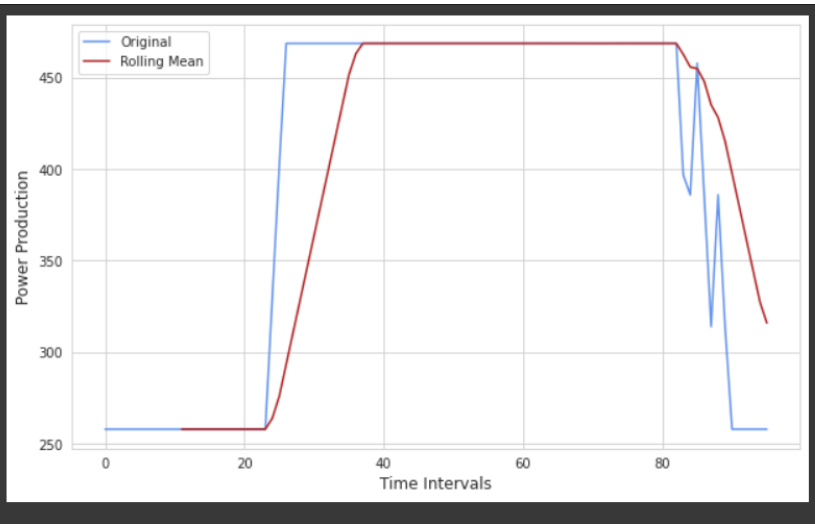
```
[ ]  1  # Now, we create a new dataframe with all the required I/P attributes
     2
     3  # Here, we are more interested in witnessing the power demand for the state
     4  # of Uttar Pradesh(48.238 %) and Rajasthan (23.378 %) - MAJOR BUYERS !
     5  # SOURCE : Northern Regional Power Committee (Govt. of India)
     6
     7  data = df_power.filter(['UNCHAHAR4'])
```

Rolling Statistics

Plotting the data is the initial step in every data analysis assignment. Graphs make it possible to see a variety of data characteristics, such as patterns, peculiar findings, changes through time, and correlations between variables. The forecasting techniques that will be utilised must thus, to the greatest extent feasible, take into account the characteristics that can be observed in plots of the data.

A moving average, also known as a rolling average or running average, is a method used in statistics to examine data points by averaging many subsets of the entire data set.

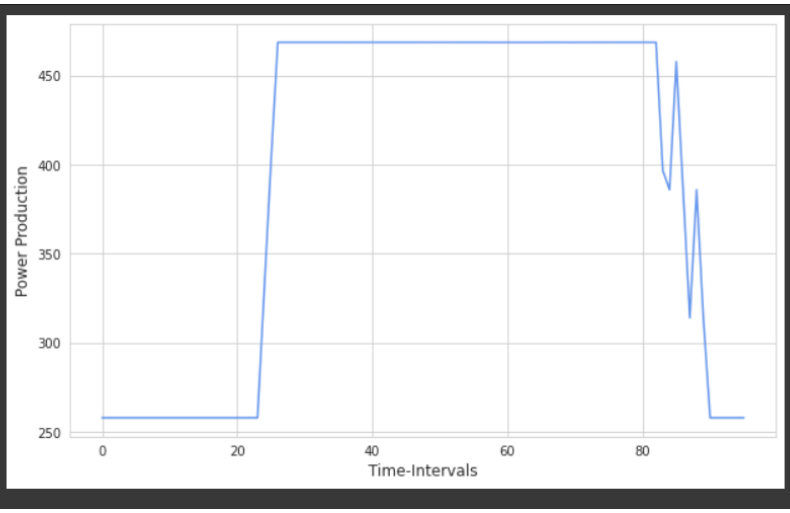
```
[ ]  1  rolling_mean = data.rolling(window=12).mean()
     2  rolling_std = data.rolling(window=12).std()
     3  plt.figure(figsize = (10,6))
     4  plt.plot(data, color='cornflowerblue', label='Original')
     5  plt.plot(rolling_mean, color='firebrick', label='Rolling Mean')
     6  # plt.plot(rolling_std, color='limegreen', label='Rolling Std')
     7  plt.xlabel('Time Intervals', size = 12)
     8  plt.ylabel('Power Production', size  = 12)
     9  plt.legend(loc = 'upper left')
    10  # plt.title('Rolling Statistics', size = 20)
    11  plt.show()
```



Checking Stationarity

A time series is said to be stationary if its statistical characteristics, or rather the method used to create it, do not vary over time.

```
[ ]  1  plt.figure(figsize = (10,6))
     2  plt.plot(data, color = 'cornflowerblue')
     3  plt.xlabel('Time-Intervals', size = 12)
     4  plt.ylabel('Power Production', size = 12)
     5  plt.show()
```




```
     1  print("Data Shape: {}".format(data.shape))
     2  value_1 = data[0:47]
     3  value_2 = data[48:95]
```

```
Data Shape: (96, 1)
```

We will proceed by splitting the data into two parts so that we can then check the **mean** and **variance** of the data.

Mean of Data

```
[ ]  1  print("Mean of value_1: {}".format(round(value_1.mean()[0],3)))
     2  print("Mean of value_2: {}".format(round(value_2.mean()[0],3)))
```

```
Mean of value_1: 356.655
Mean of value_2: 432.673
```

Variance of Data

```
[ ]  1  print("Variance of value_1: {}".format(round(value_1.var()[0],3)))
     2  print("Variance of value_2: {}".format(round(value_2.var()[0],3)))
```

```
Variance of value_1: 10893.847
Variance of value_2: 5111.329
```

Augmented Dickey-Fuller Test (ADF Test)

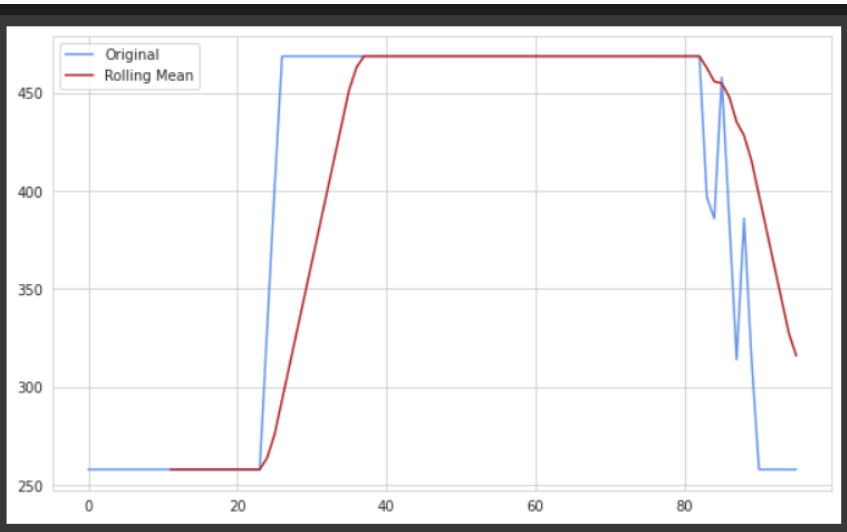
Augmented Dickey-Fuller Test is a common statistical test used to test whether a given time series is stationary or not. We can achieve this by defining the null and alternate hypothesis.

Null Hypothesis: Time Series is non-stationary. It gives a time-dependent trend.
Alternate Hypothesis: Time Series is stationary. In another term, the series doesn't depend on time.

ADF or t Statistic < critical values: Reject the null hypothesis, time series is stationary.
ADF or t Statistic > critical values: Failed to reject the null hypothesis, time series is non-stationary.

```python
def adfuller_test(ts, window = 12):

    movingAverage = ts.rolling(window).mean()
    movingSTD = ts.rolling(window).std()

    plt.figure(figsize = (10,6))
    orig = plt.plot(ts, color='cornflowerblue',
                    label='Original')
    mean = plt.plot(movingAverage, color='firebrick',
                    label='Rolling Mean')
    # std = plt.plot(movingSTD, color='limegreen',
    #                label='Rolling Std')
    plt.legend(loc = 'upper left')
    # plt.title('Rolling Statistics', size = 14)
    plt.show(block=False)

    adf = adfuller(ts, autolag='AIC')
```

```python

    print('\nADF Statistic: {}'.format(round(adf[0],3)))
    print('p-value: {}'.format(round(adf[1],3)))
    print("-----------------------------------")
    print('Critical Values:')

    for key, ts in adf[4].items():
        print('{}: {}'.format(key, round(ts,3)))
    print("-----------------------------------")

    if adf[0] > adf[4]["5%"]:
        print("ADF > Critical Values")
        print ("Failed to reject null hypothesis, time series is non-stationary.")
    else:
        print("ADF < Critical Values")
        print ("Reject null hypothesis, time series is stationary.")

adfuller_test(data, window = 12)
```



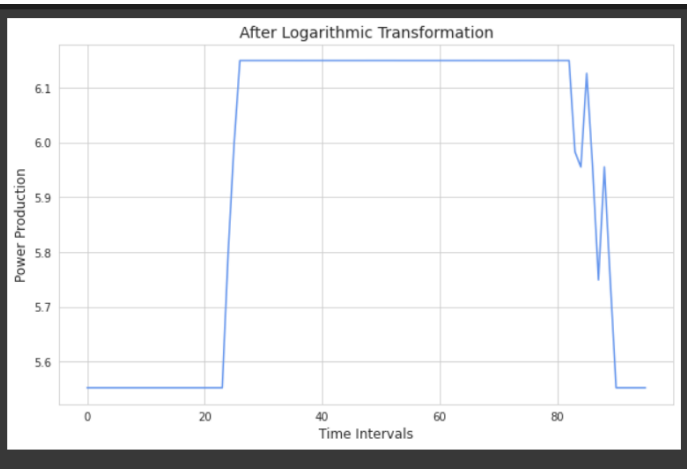

```
ADF Statistic: -1.753
p-value: 0.404
-----------------------------------
Critical Values:
1%: -3.504
5%: -2.894
10%: -2.584
-----------------------------------
ADF > Critical Values
Failed to reject null hypothesis, time series is non-stationary.
```

Converting Data to Stationary

Logarithmic Transformation with Box-Cox

The Box-Cox transform is a configurable data transform method that supports both square root and log transform, as well as a suite of related transforms.

```
data_log_scaled = data
# data_log_scaled['UNCHAHAR4'] = boxcox(data_log_scaled['UNCHAHAR4'], lmbda=0.0)
data_log_scaled['UNCHAHAR4'] = boxcox(data_log_scaled['UNCHAHAR4'], 0.0)
plt.figure(figsize = (10,6))
plt.plot(data_log_scaled, color = 'cornflowerblue')
plt.xlabel('Time Intervals', size = 12)
plt.ylabel('Power Production', size = 12)
plt.title("After Logarithmic Transformation", size = 14)
plt.show()
```



Removing Trend with Moving Average

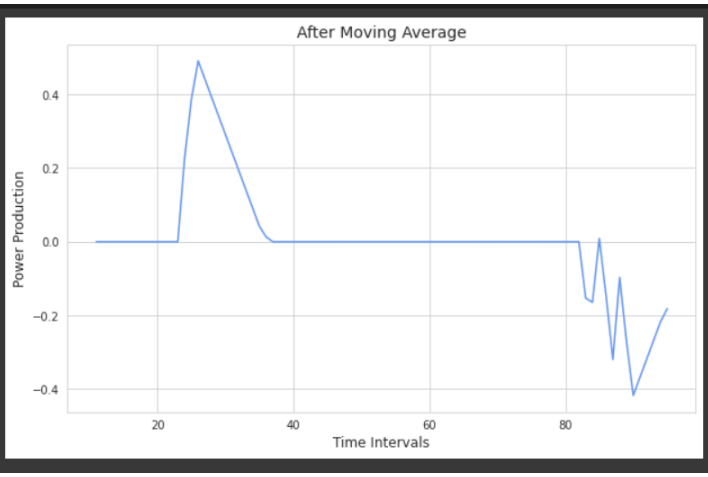
`Non-stationary` time series are those that exhibit a trend.

It is possible to simulate a discovered trend. It may be taken out of the time series dataset after modelling. Detrending the time series is what this is known as.

A dataset is referred to as trend stationary if it lacks a trend or if the trend can be successfully eliminated.

+ Code + Text

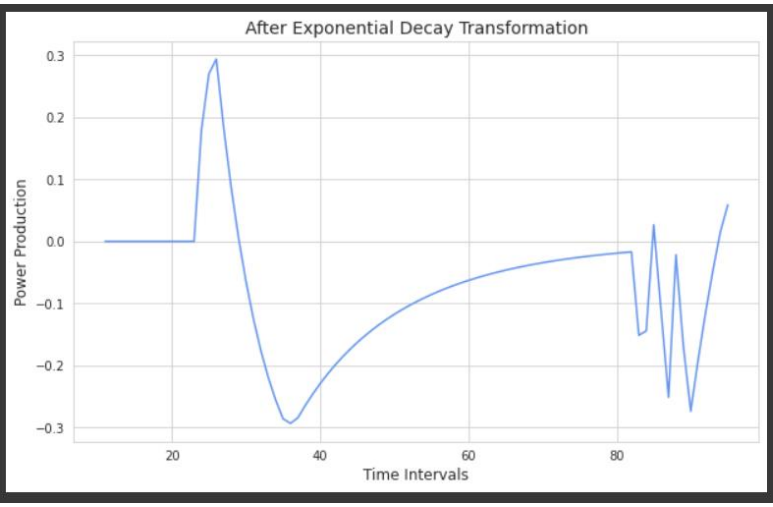
```
[ ]
moving_avg = data_log_scaled.rolling(window=12).mean()
data_log_scaled_ma = data_log_scaled - moving_avg
data_log_scaled_ma.dropna(inplace=True)
plt.figure(figsize = (10,6))
plt.plot(data_log_scaled_ma, color = 'cornflowerblue')
plt.xlabel('Time Intervals', size = 12)
plt.ylabel('Power Production', size = 12)
plt.title("After Moving Average", size = 14)
plt.show()
```



By replacing a monomial function's variable with an exponential variable, an exponential transformation is a straightforward algebraic transformation. If a quantity declines at a pace proportionate to its current value, exponential decay may be present.

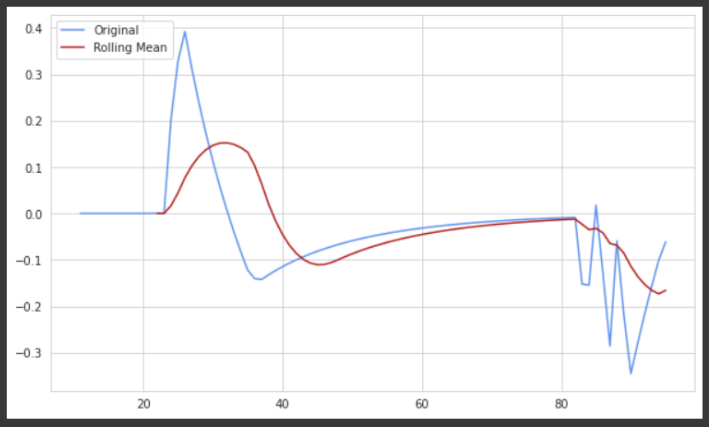
Exponential Decay Transformation

```
1  data_log_scaled_ma_ed = data_log_scaled_ma.ewm(halflife=12, min_periods=0, adjust=True).mean()
2  data_lsma_sub_data_lsma_ed = data_log_scaled_ma - data_log_scaled_ma_ed
3  plt.figure(figsize = (10,6))
4  plt.plot(data_lsma_sub_data_lsma_ed - data_log_scaled_ma_ed, color='cornflowerblue')
5  plt.xlabel('Time Intervals', size = 12)
6  plt.ylabel('Power Production', size = 12)
7  plt.title("After Exponential Decay Transformation", size = 14)
8  plt.show()
```



Let's test stationarity again.

```
[ ]  1  adfuller_test(data_lsma_sub_data_lsma_ed, window = 12)
```



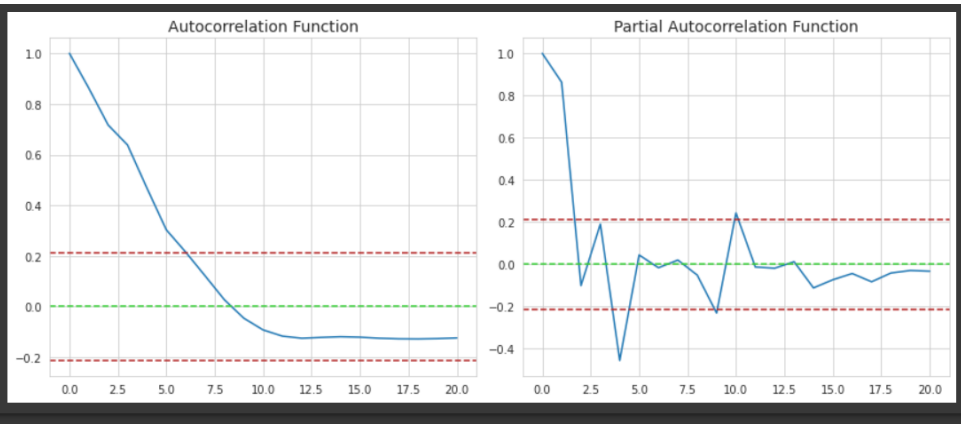
```
ADF Statistic: -3.231
p-value: 0.018
----------------------------------
Critical Values:
1%: -3.514
5%: -2.898
10%: -2.586
----------------------------------
ADF < Critical Values
Reject null hypothesis, time series is stationary.
```

ACF & PACF

Time series analysis and forecasting frequently make use of autocorrelation and partial autocorrelation plots.

These graphs illustrate the degree to which an observation in a time series is correlated with observations made at earlier time steps.

```python
auto_c_f = acf(data_lsma_sub_data_lsma_ed, nlags=20)
partial_auto_c_f = pacf(data_lsma_sub_data_lsma_ed, nlags=20, method='ols')

fig, axs = plt.subplots(1, 2, figsize =(12,5))

plt.subplot(121)
plt.plot(auto_c_f)
plt.axhline(y=0, linestyle='--', color='limegreen')
plt.axhline(y=-1.96/np.sqrt(len(data_lsma_sub_data_lsma_ed)),
            linestyle='--', color='firebrick')
plt.axhline(y=1.96/np.sqrt(len(data_lsma_sub_data_lsma_ed)),
            linestyle='--', color='firebrick')
plt.title('Autocorrelation Function', size = 14)

plt.subplot(122)
plt.plot(partial_auto_c_f)
plt.axhline(y=0, linestyle='--', color='limegreen')
plt.axhline(y=-1.96/np.sqrt(len(data_lsma_sub_data_lsma_ed)),
            linestyle='--', color='firebrick')
plt.axhline(y=1.96/np.sqrt(len(data_lsma_sub_data_lsma_ed)),
            linestyle='--', color='firebrick')
plt.title('Partial Autocorrelation Function', size = 14)

plt.tight_layout()
```



Prediction Models

Persistence Model

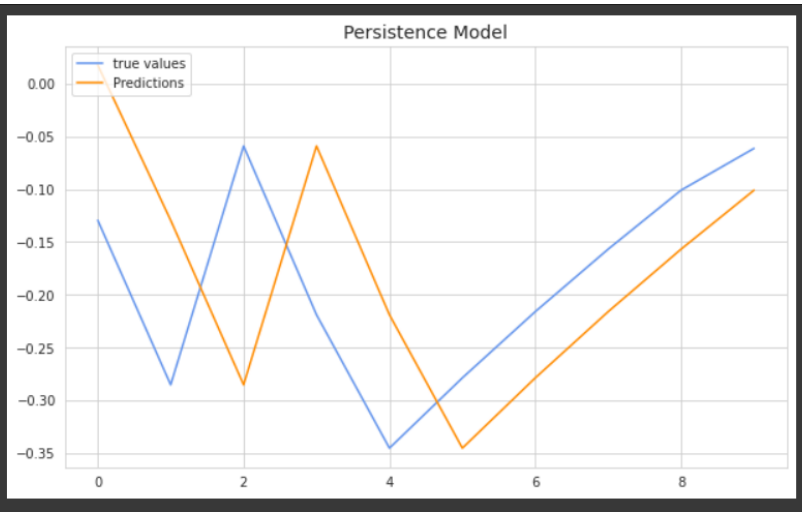
Let's imagine we want to create a model that, given all previous observations, can predict the last 10 blocks' power production values of the dataset's electric generation. Persisting the most recent observation would be the most basic model we might employ to generate predictions. This model, which we might refer to as a persistence model, offers a baseline of performance for the issue that we can compare to an autoregression model.

```
[ ]  values = DataFrame(data_lsma_sub_data_lsma_ed.values)
     persistence_df = concat([values.shift(1), values], axis=1)
     persistence_df.columns = ['t-1', 't+1']
     per_values = persistence_df.values

     train = per_values[1:len(per_values)-10]
     test = per_values[len(per_values)-10:]

     X_train, y_train = train[:,0], train[:,1]
     X_test, y_test = test[:,0], test[:,1]
```

```
def persistence(x):
    return x

predictions = []
for i in X_test:
    y_pred = persistence(i)
    predictions.append(y_pred)

persistence_score = mean_squared_error(y_test, predictions)
print('Persistence MSE: {}'.format(round(persistence_score,4)))
```

```
Persistence MSE: 0.0155
```

```
[ ]  plt.figure(figsize = (10,6))
     plt.plot(y_test, label = "true values", color = "cornflowerblue")
     plt.plot(predictions,label = "Predictions", color='darkorange')
     plt.title("Persistence Model", size = 14)
     plt.legend(loc = 'upper left')
     plt.show()
```



Autoregression Model

A linear regression model that incorporates lagged data as input variables is known as an autoregression model.

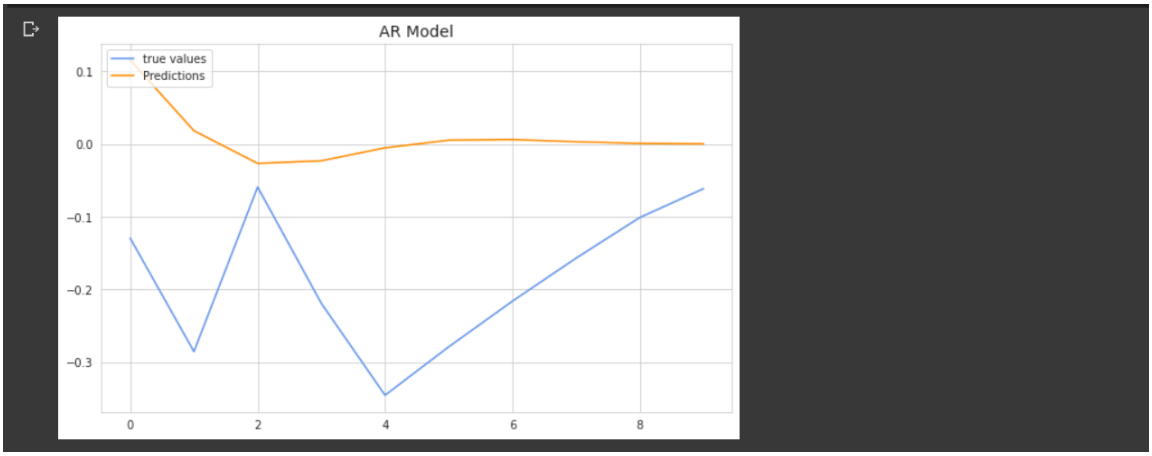
Using the LinearRegession class in scikit-learn, we could manually construct the linear regression model and provide the lag input variables to utilise.

The statsmodels package also trains a linear regression model and offers an autoregression model where you must supply an acceptable lag value. It is accessible through the AutoReg class.

```
[ ]  1  ar_values = data_lsma_sub_data_lsma_ed.values
     2  train = ar_values[1:len(ar_values)-10]
     3  test = ar_values[len(ar_values)-10:]
     4  model = ARIMA(train, order=(2,1,0))
     5  AR_model = model.fit()
     6
     7  predictions = AR_model.predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)
     8  ar_score = mean_squared_error(test, predictions)
     9  print('AR MSE: {}'.format(round(ar_score,4)))
```

```
AR MSE: 0.0477
```

```
[ ]  1  plt.figure(figsize = (10,6))
     2  plt.plot(test, label = "true values", color = "cornflowerblue")
     3  plt.plot(predictions,label = "Predictions", color='darkorange')
     4  plt.title("AR Model", size = 14)
     5  plt.legend(loc = 'upper left')
     6  plt.show()
```



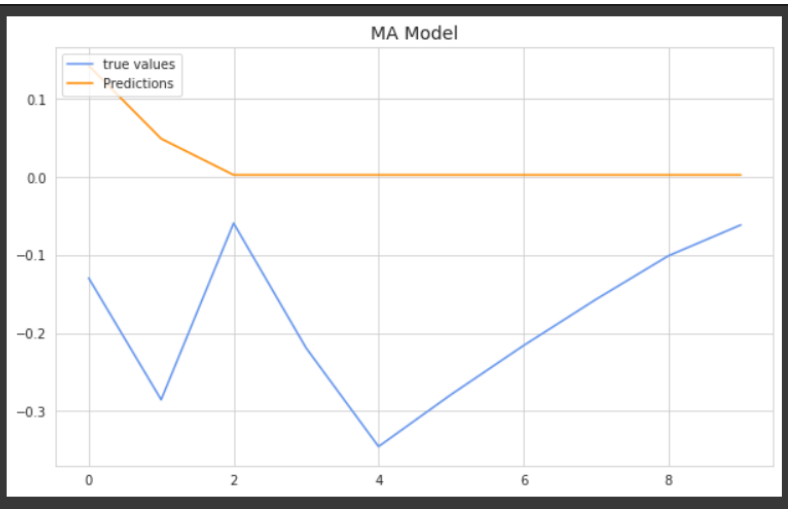
Moving average smoothing is used to determine the trend-cycle of previous values, whereas a moving average model is used to predict future values.

Moving Average Model

```
1  model = ARIMA(train, order=(0,1,2))
2  MA_model = model.fit()
3
4  predictions = MA_model.predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)
5  ma_score = mean_squared_error(test, predictions)
6  print('MA MSE: {}'.format(round(ma_score,4)))
```

```
MA MSE: 0.0528
```

```
[ ]  1  plt.figure(figsize = (10,6))
     2  plt.plot(test, label = "true values", color = "cornflowerblue")
     3  plt.plot(predictions,label = "Predictions", color='darkorange')
     4  plt.title("MA Model", size = 14)
     5  plt.legend(loc = 'upper left')
     6  plt.show()
```



ARIMA Model

An ARIMA model may be fitted using the statsmodels package.

The statsmodels package may be used to build an ARIMA model as follows:

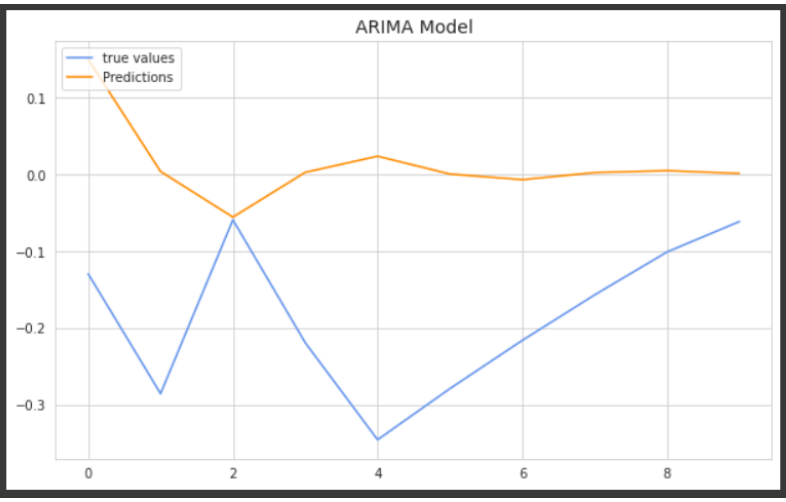
1. Call ARIMA() and pass the **p, d, and q** parameters to define the model.
2. The fit() method is used to prepare the model using the training data.
3. Calling the predict() method with the index of the time or times to be forecasted will provide a prediction.

Let's begin with something straightforward. We will analyse the complete dataset for Electric Production using an ARIMA model.

```
[ ]  1  model = ARIMA(train, order=(2,1,2))
     2  ARIMA_model = model.fit()
     3
     4  predictions = ARIMA_model.predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)
     5  arima_score = mean_squared_error(test, predictions)
     6  print('ARIMA MSE: {}'.format(round(arima_score,4)))
```

```
ARIMA MSE: 0.051
```

```
[ ]  1  plt.figure(figsize = (10,6))
     2  plt.plot(test, label = "true values", color = "cornflowerblue")
     3  plt.plot(predictions,label = "Predictions", color='darkorange')
     4  plt.title("ARIMA Model", size = 14)
     5  plt.legend(loc = 'upper left')
     6  plt.show()
```



5.3 TESTING & PERFORMANCE METRICS

1□ **LSTM MODEL**

Now that I have got the trained LSTM Model, it's time to get the prediction on the basis of **x_test** data component. Note that I had transformed the original values by scaling them using **MinMaxScaler.** Hence, I have to apply inverse transform function to the prediction output.

```
# Now let us acquire the predicted values
# with respect to the model created

# LSTM
predictions_lstm = model_lstm.predict(x_test)
predictions_lstm = scaler_y.inverse_transform(predictions_lstm)
```

```
1/1 [==============================] - 1s 961ms/step
```

```
predictions_lstm
```

```
array([[432.30017],
       [435.07883],
       [432.22836],
       [427.938  ],
       [423.60574],
       [453.3386 ],
       [471.8967 ],
       [485.84302],
       [479.8924 ],
       [468.65912],
       [459.58618],
       [450.6375 ],
       [442.01584],
       [439.95786],
       [421.9976 ],
       [397.33292],
       [380.59216],
       [366.23718],
       [363.0653 ],
       [360.15814],
       [353.60632],
       [337.80508],
       [327.0375 ],
       [321.72382]], dtype=float32)
```

The Model performance evaluation will be based on **ERROR Rates – Why?** Industries know that a model's accuracy can easily be increased by increasing the number of epochs while training. But the greater the number of epochs, the greater the time for training – not a feasible solution in real-time scenario. On the other hand, precision can also be increased if we feed a huge dataset, consisting of numerous rows. But that seems to be pretty much denoting the historical data – again not a real-time on-spot solution scenario. But as industries

are more concerned with efficiency and safety, **ERROR RATES** seems to work best for them. Hence, I have used **RMSE (Root Mean Squared Error)** and **R^2 score.**

```
[ ]  # As now we have stored our predicted close price values
     # above , it's time for evaluation metrics to come into the
     # scene

     from sklearn import metrics

     # Get the root mean squared error (RMSE) for LSTM model
     mse_lstm = metrics.mean_squared_error(y_test, predictions_lstm)
     rmse_lstm = np.sqrt(mse_lstm)
     print("LSTM Model RMSE : ", rmse_lstm)

     # Get r2 score (R-squared - Coefficient of Determination) for LSTM Model
     r2_lstm = metrics.r2_score(y_test, predictions_lstm)
     print("LSTM Model r2 : ", r2_lstm)
```

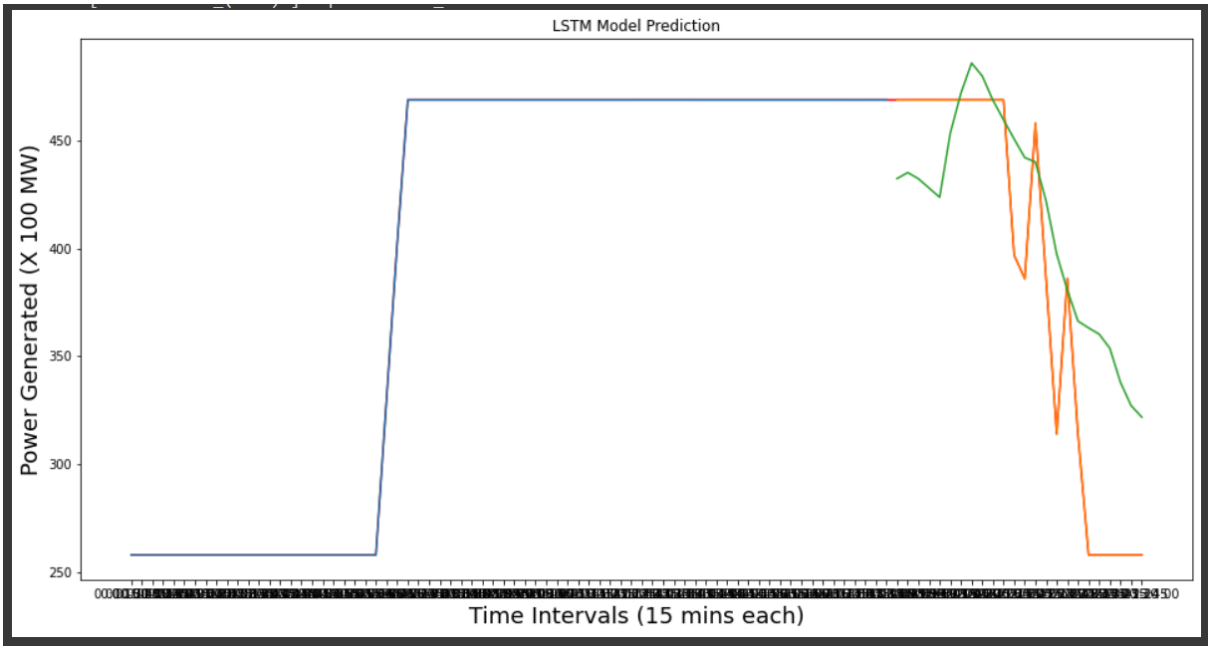
```
LSTM Model RMSE :  7754.755009745294
LSTM Model r2 :  -29.81066974097717
```

RMSE will provide how much the predicted values have deviated from the original testing values, while R^2 score will tell the model fitting (how far are the predicted values from the actual values).

Let's plot our predicted data, to visualize and get an explanation to the above 2 metrics.

```
     # Now we shall plot the training data , validation data and Predictions
     # of LSTM Model

     # Data Plotting
     train = data[:training_data_len]
     valid1 = data[training_data_len:]
     valid1['Predictions_(LSTM)'] = predictions_lstm

     # Data Visualization
     plt.figure(figsize=(16,8))
     plt.title('LSTM Model Prediction')
     default_x_ticks = range(len(df_power['Time Desc']))
     plt.plot(default_x_ticks , df_power['UNCHAHAR4'] , color='Red')
     plt.xticks(default_x_ticks , df_power['Time Desc'])
     plt.xlabel('Time Intervals (15 mins each)', fontsize=18)
     plt.ylabel('Power Generated (X 100 MW)', fontsize=18)
     plt.plot(train['UNCHAHAR4'])
     plt.plot(valid1[['UNCHAHAR4' , 'Predictions_(LSTM)']])
     plt.show()
```

```
<ipython-input-25-00692d403923>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
  valid1['Predictions_(LSTM)'] = predictions_lstm
```




```
[ ]  1  # Let us see how the predictions goes with the Valid data
     2  # for the LSTM Model
     3
     4  valid1['Time Desc'] = df_power['Time Desc'][training_data_len:]
     5  valid1[['Time Desc' , 'UNCHAHAR4' , 'Predictions_(LSTM)']][:5]
```

```
<ipython-input-17-8fe38c1d9d99>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
  valid1['Time Desc'] = df_power['Time Desc'][training_data_len:]
```

	Time Desc	UNCHAHAR4	Predictions_(LSTM)
72	18:00-18:15	468.7475	432.300171
73	18:15-18:30	468.7475	435.078827
74	18:30-18:45	468.7575	432.228363
75	18:45-19:00	468.7575	427.937988
76	19:00-19:15	468.7500	423.605743

```
[ ]  1  # Now , our main motive is to predict the demand OR the power
     2  # to be generated for 10 blocks of next day
```

```
[ ]  1  len (test_data)
```

```
34
```

```
[ ]  1  predict_test_data = test_data[:,0].reshape((-1))
```

The following is the function will forecast the values for the next day, with respect to the testing data.

```
def predict(num_prediction , model):
  prediction_list = predict_test_data[-30:]

  for _ in range(num_prediction-1):
    x = prediction_list[-30:]
    x = x.reshape((1,10,3))
    out = model.predict(x)[0][0]
    prediction_list = np.append(prediction_list , out)
  prediction_list = prediction_list[30-1:]

  return prediction_list

num_prediction = 10 # we need the values of 10 blocks of the next day
forecast_next_day = predict(num_prediction,model_lstm)
```

```
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 41ms/step
```

```
[ ] forecast_next_day
```

```
array([0.        , 0.07597898, 0.0910871 , 0.14623788, 0.00084411,
       0.11674237, 0.13650909, 0.04332064, 0.17885856, 0.18048553])
```

```
[ ] forecast_next_day = np.reshape(forecast_next_day , (2,5))
    forecast_next_day = scaler_y.inverse_transform(forecast_next_day)
```

```
[ ] forecast_next_day
```

```
array([[257.81      , 273.83757577, 277.02459507, 288.65851519,
        257.98806285],
       [282.43651005, 286.6062514 , 266.94838011, 295.53976678,
        295.8829717 ]])
```

```
[ ] # Let us save this numpy array in a text file for later comparsion

    np.savetxt("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_LSTM.txt", forecast_next_day)
```

```
[ ] forecast_next_day = np.reshape(forecast_next_day , (10,1))
```

```
forecast_next_day
```

```
array([[257.81      ],
       [273.83757577],
       [277.02459507],
       [288.65851519],
       [257.98806285],
       [282.43651005],
       [286.6062514 ],
       [266.94838011],
       [295.53976678],
       [295.8829717 ]])
```

```
df = pd.DataFrame(forecast_next_day , columns = ["UNCHAHAR4"] , index = (range(1,11)))
df
```

	UNCHAHAR4
1	257.810000
2	273.837576
3	277.024595
4	288.658515
5	257.988063
6	282.436510
7	286.606251
8	266.948380
9	295.539767
10	295.882972

```
new_time_desc = df_power['Time Desc'][:10]
new_time_desc
```

```
0    00:00-00:15
1    00:15-00:30
2    00:30-00:45
3    00:45-01:00
4    01:00-01:15
5    01:15-01:30
6    01:30-01:45
7    01:45-02:00
8    02:00-02:15
9    02:15-02:30
Name: Time Desc, dtype: object
```

```
new_time_desc = np.asarray(new_time_desc)
new_time_desc
```

```
array(['00:00-00:15', '00:15-00:30', '00:30-00:45', '00:45-01:00',
       '01:00-01:15', '01:15-01:30', '01:30-01:45', '01:45-02:00',
       '02:00-02:15', '02:15-02:30'], dtype=object)
```

Power Generation for initial 10 blocks of Next Day

```
df.insert(0,"Time Desc",new_time_desc,True)
df
```

	Time Desc	UNCHAHAR4
1	00:00-00:15	257.810000
2	00:15-00:30	273.837576
3	00:30-00:45	277.024595
4	00:45-01:00	288.658515
5	01:00-01:15	257.988063
6	01:15-01:30	282.436510
7	01:30-01:45	286.606251
8	01:45-02:00	266.948380
9	02:00-02:15	295.539767
10	02:15-02:30	295.882972

2□ **RNN**

The structure of the code remains the same as LSTM (except the visualizations and values predicted).

```
# Now let us acquire the predicted close price values
# with respect to the models created

# RNN
predictions_rnn = model_rnn.predict(x_test)
predictions_rnn = scaler_y.inverse_transform(predictions_rnn)
```

```
1/1 [==============================] - 0s 363ms/step
```

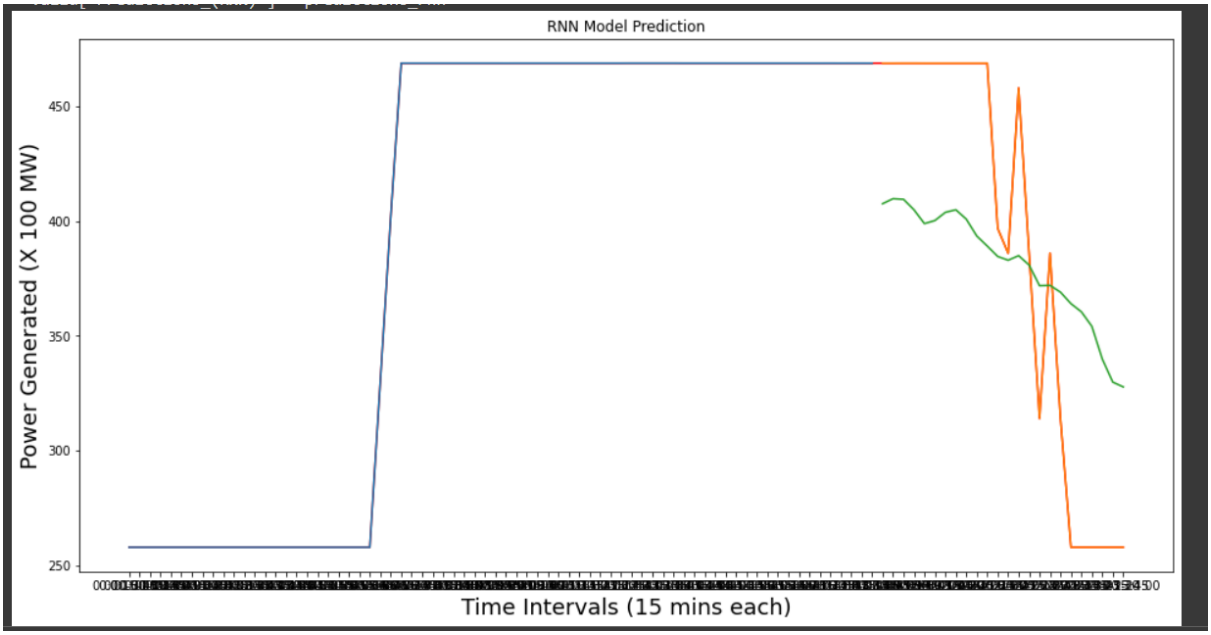
```
predictions_rnn
```

```
array([[407.50784],
       [409.6961 ],
       [409.4269 ],
       [404.8542 ],
       [398.82864],
       [400.17313],
       [403.7411 ],
       [404.86765],
       [400.77893],
       [393.437  ],
       [389.06003],
       [384.5317 ],
       [382.87808],
       [384.8727 ],
       [380.78918],
       [371.77203],
       [371.91702],
       [368.9618 ],
       [363.96396],
       [360.3886 ],
       [354.0768 ],
       [339.8038 ],
       [329.77057],
       [327.67822]], dtype=float32)
```

```python
# As now we have stored our predicted values
# above , it's time for evaluation metrics to come into the
# scene

from sklearn import metrics

# Get the root mean squared error (RMSE) for RNN Model
mse_rnn = metrics.mean_squared_error(y_test, predictions_rnn)
rmse_rnn = np.sqrt(mse_rnn)
print("RNN Model RMSE: ", rmse_rnn)

# Get r2 score for RNN Model
r2_rnn = metrics.r2_score(y_test, predictions_rnn)
print("RNN Model r2: ", r2_rnn)
```

```
RNN Model RMSE:  7791.473711149366
RNN Model r2:  -30.103137058059662
```

```python
# Now we shall plot the training data , validation data and Predictions
# of RNN Model

# Data Plotting
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions_(RNN)'] = predictions_rnn

# Data Visualization
plt.figure(figsize=(16,8))
plt.title('RNN Model Prediction')
default_x_ticks = range(len(df_power['Time Desc']))
plt.plot(default_x_ticks , df_power['UNCHAHAR4'] , color='Red')
plt.xticks(default_x_ticks , df_power['Time Desc'])
plt.xlabel('Time Intervals (15 mins each)', fontsize=18)
plt.ylabel('Power Generated (X 100 MW)', fontsize=18)
plt.plot(train['UNCHAHAR4'])
plt.plot(valid[['UNCHAHAR4' , 'Predictions_(RNN)']])
plt.show()
```



```
# Let us see how the predictions goes with the Valid data
# for the RNN Model

valid['Time Desc'] = df_power['Time Desc'][training_data_len:]
valid[['Time Desc' , 'UNCHAHAR4' , 'Predictions_(RNN)']][:5]
```

```
<ipython-input-15-534b2a2ad666>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vie
  valid['Time Desc'] = df_power['Time Desc'][training_data_len:]
```

	Time Desc	UNCHAHAR4	Predictions_(RNN)
72	18:00-18:15	468.7475	407.507843
73	18:15-18:30	468.7475	409.696106
74	18:30-18:45	468.7575	409.426910
75	18:45-19:00	468.7575	404.854187
76	19:00-19:15	468.7500	398.828644

```
# Now , our main motive is to predict the demand OR the power
# to be generated for 10 blocks of next day
```

```
len (test_data)
```

```
34
```

```
predict_test_data = test_data[:,0].reshape((-1))
```

```
def predict(num_prediction , model):
  prediction_list = predict_test_data[-30:]

  for _ in range(num_prediction-1):
    x = prediction_list[-30:]
    x = x.reshape((1,10,3))
    out = model.predict(x)[0][0]
    prediction_list = np.append(prediction_list , out)
  prediction_list = prediction_list[30-1:]

  return prediction_list

num_prediction = 10 # we need the values of 10 blocks of the next day
forecast_next_day = predict(num_prediction,model_rnn)
```

```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
```

```
forecast_next_day
```

```
array([0.        , 0.22867465, 0.33221281, 0.34319603, 0.16955587,
       0.34631678, 0.25476673, 0.11832099, 0.36723951, 0.269187  ])
```

```
forecast_next_day = np.reshape(forecast_next_day , (2,5))
forecast_next_day = scaler_y.inverse_transform(forecast_next_day)
```

```
forecast_next_day
```

```
array([[257.81      , 306.04834577, 327.88946084, 330.20634547,
        293.57738745],
       [330.86465992, 311.55240529, 282.76951633, 335.27825549,
        314.59432539]])
```

```
[ ]  1  # Let us save this numpy array in a text file for later comparsion
     2
     3  np.savetxt("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_RNN.txt", forecast_next_day)
```

```
[ ]  1  forecast_next_day = np.reshape(forecast_next_day , (10,1))
```

```
[ ]  1  forecast_next_day
```

```
array([[257.81      ],
       [306.04834577],
       [327.88946084],
       [330.20634547],
       [293.57738745],
       [330.86465992],
       [311.55240529],
       [282.76951633],
       [335.27825549],
       [314.59432539]])
```

```
[ ]  1  df = pd.DataFrame(forecast_next_day , columns = ["UNCHAHAR4"] , index = (range(1,11)))
     2  df
```

	UNCHAHAR4
1	257.810000
2	306.048346
3	327.889461
4	330.206345
5	293.577387
6	330.864660
7	311.552405
8	282.769516
9	335.278255
10	314.594325

```
     1  new_time_desc = df_power['Time Desc'][:10]
     2  new_time_desc
```

```
0    00:00-00:15
1    00:15-00:30
2    00:30-00:45
3    00:45-01:00
4    01:00-01:15
5    01:15-01:30
6    01:30-01:45
7    01:45-02:00
8    02:00-02:15
9    02:15-02:30
Name: Time Desc, dtype: object
```

```
[ ]  1  new_time_desc = np.asarray(new_time_desc)
     2  new_time_desc
```

```
array(['00:00-00:15', '00:15-00:30', '00:30-00:45', '00:45-01:00',
       '01:00-01:15', '01:15-01:30', '01:30-01:45', '01:45-02:00',
       '02:00-02:15', '02:15-02:30'], dtype=object)
```

Power Generation for initial 10 blocks of Next Day

```
[ ]  1  df.insert(0,"Time Desc",new_time_desc,True)
     2  df
```

	Time Desc	UNCHAHAR4
1	00:00-00:15	257.810000
2	00:15-00:30	306.048346
3	00:30-00:45	327.889461
4	00:45-01:00	330.206345
5	01:00-01:15	293.577387
6	01:15-01:30	330.864660
7	01:30-01:45	311.552405
8	01:45-02:00	282.769516
9	02:00-02:15	335.278255
10	02:15-02:30	314.594325

3□ **ARIMA**

Now that we have all the trained models and tested them to predict the testing data, it's time to evaluate their performance on the basis of RMSE and R^2 score.

Mean Squared Error

```
1  errors = pd.DataFrame()
2  errors["Model"] = ["Persistence", "Autoregression", "Moving Average", "ARIMA"]
3  errors["MSE"] = [persistence_score, ar_score, ma_score, arima_score]
4  errors = errors.sort_values("MSE", ascending = True, ignore_index = True)
5  errors.index = errors.Model
6  del errors["Model"]
7
8  def coloring_bg(s, min_, max_, cmap='Reds', low=0, high=0):
9      color_range = max_ - min_
10     norm = colors.Normalize(min_ - (color_range * low), max_ + (color_range * high))
11     normed = norm(s.values)
12     c = [colors.rgb2hex(x) for x in plt.cm.get_cmap(cmap)(normed)]
13     return ['background-color: %s' % color for color in c]
14
15 errors.style.apply(coloring_bg,min_ =errors.min().min(),
16                    max_ = errors.max().max(), low = 0.1, high = 0.85)
```

Model	MSE
Persistence	0.015525
Autoregression	0.094622
ARIMA	0.111008
Moving Average	0.157764

```
# Lets have a look to the predicted values

predictions
```

```
array([0.16596171, 0.16751943, 0.10976177, 0.1110285 , 0.13345898,
       0.13223883, 0.12355145, 0.12430786, 0.1276633 , 0.12726021])
```

```
# As now we have stored our predicted values
# above , it's time for evaluation metrics to come into the
# scene

from sklearn import metrics

# Get the root mean squared error (RMSE) for ARIMA Model
rmse = np.sqrt(arima_score)
print("ARIMA Model RMSE : ",rmse)

# Get r2 score for RNN Model
r2_rnn = metrics.r2_score(test, predictions)
print("ARIMA Model r2: ", r2_rnn)
```

```
ARIMA Model RMSE :  0.3331780365528068
ARIMA Model r2:  -11.49937439667004
```

```
# The prediction values we got are still transformed
# Thus, we have to apply inverse transformation to get the
# original values

predictions = inv_boxcox(predictions, 0.0)
```

```
df_actual_values = df_power['UNCHAHAR4'][86:96].copy()
df_actual_values
```

```
86    385.930000
87    313.930000
88    385.939821
89    313.930000
90    257.810000
91    257.810000
92    257.810000
93    257.810000
94    257.810000
95    257.810000
Name: UNCHAHAR4, dtype: float64
```

```
print(type(df_actual_values))
```

```
<class 'pandas.core.series.Series'>
```

```
[ ]  1  df_actual_values = list(df_actual_values)
     2  df_actual_values
```

```
[385.93,
 313.93,
 385.939821,
 313.93,
 257.81,
 257.81,
 257.81,
 257.81,
 257.81,
 257.81]
```

```
[ ]  1  df_actual_values = np.array(df_actual_values)
     2  df_actual_values
```

```
array([385.93    , 313.93    , 385.939821, 313.93    , 257.81    ,
       257.81    , 257.81    , 257.81    , 257.81    , 257.81    ])
```

```
[ ]  1  Actual_Prediction_Values = np.multiply(df_actual_values, predictions)
     2  Actual_Prediction_Values
```

```
array([455.60112982, 371.18086788, 430.71353769, 350.79377998,
       294.61866433, 294.2594056 , 291.71413469, 291.93487218,
       292.91608712, 292.79803899])
```

```
[ ]  1  # Lets put it in a more visually attractive & understandable manner
     2
     3  valid = df_power[86:96]
     4  valid['Predictions_(ARIMA)'] = Actual_Prediction_Values
     5  valid['Time Desc'] = df_power['Time Desc'][86:96]
     6  valid[['Time Desc' , 'UNCHAHAR4' , 'Predictions_(ARIMA)']]
```

	Time Desc	UNCHAHAR4	Predictions_(ARIMA)
86	21:30-21:45	385.930000	455.601130
87	21:45-22:00	313.930000	371.180868
88	22:00-22:15	385.939821	430.713538
89	22:15-22:30	313.930000	350.793780
90	22:30-22:45	257.810000	294.618664
91	22:45-23:00	257.810000	294.259406
92	23:00-23:15	257.810000	291.714135
93	23:15-23:30	257.810000	291.934872
94	23:30-23:45	257.810000	292.916087
95	23:45-24:00	257.810000	292.798039

```
[ ]  1  # Now , our main motive is to predict the demand OR the power
     2  # to be generated for 10 blocks of next day
```

```
     1  len (test)
```

```
10
```

```
[ ]  predict_test_data = test[:, 0].reshape((-1))
```

```
def predict(num_prediction , model):
  prediction_list = predict_test_data[:]

  for _ in range(num_prediction-1):
    x = prediction_list[:]
    # x = x.reshape((1,10,1))
    out = model.predict(x)[0]
    prediction_list = np.append(prediction_list , out)
  prediction_list = prediction_list[:]

  return prediction_list

num_prediction = 10
forecast_next_day = predict(num_prediction,model)
```

```
[ ]  forecast_next_day
```

```
array([-0.12942015, -0.28531759, -0.05905403, -0.21898986, -0.34536364,
       -0.27879997, -0.21601409, -0.15678796, -0.10091651, -0.06131603,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ])
```

```
[ ]  forecast_next_day = forecast_next_day[:10]
     forecast_next_day
```

```
array([-0.12942015, -0.28531759, -0.05905403, -0.21898986, -0.34536364,
       -0.27879997, -0.21601409, -0.15678796, -0.10091651, -0.06131603])
```

```
forecast_next_day = inv_boxcox(forecast_next_day, 0.0)
forecast_next_day
```

```
array([0.87860474, 0.75177546, 0.94265584, 0.80332986, 0.70796286,
       0.75669125, 0.80572395, 0.85488531, 0.90400851, 0.94052595])
```

```
[ ]  a = df_power['UNCHAHAR4'][0:10]
     a = np.array(a)
     Actual_Prediction_Next_Day = np.multiply(forecast_next_day, a)
```

```
[ ]  Actual_Prediction_Next_Day
```

```
array([226.51308751, 193.81523165, 243.02610121, 207.10647182,
       182.51990482, 195.08257033, 207.72369095, 220.39798232,
       233.06243339, 242.47699645])
```

```
[ ]  # Let us save this numpy array in a text file for later comparsion

     np.savetxt("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_ARIMA.txt", Actual_Prediction_Next_Day)
```

```
[ ]  Actual_Prediction_Next_Day = np.reshape(Actual_Prediction_Next_Day , (10,1))
     Actual_Prediction_Next_Day
```

```
array([[226.51308751],
       [193.81523165],
       [243.02610121],
       [207.10647182],
       [182.51990482],
       [195.08257033],
       [207.72369095],
       [220.39798232],
       [233.06243339],
       [242.47699645]])
```

```
1  Predicted_df = pd.DataFrame(Actual_Prediction_Next_Day, columns = ['UNCHAHAR4'], index = (range(1,11)))
2  Predicted_df
```

	UNCHAHAR4
1	226.513088
2	193.815232
3	243.026101
4	207.106472
5	182.519905
6	195.082570
7	207.723691
8	220.397982
9	233.062433
10	242.476996

```
1  new_time_desc = df_power['Time Desc'][:10]
2  new_time_desc
```

```
0    00:00-00:15
1    00:15-00:30
2    00:30-00:45
3    00:45-01:00
4    01:00-01:15
5    01:15-01:30
6    01:30-01:45
7    01:45-02:00
8    02:00-02:15
9    02:15-02:30
Name: Time Desc, dtype: object
```

```
1  new_time_desc = np.asarray(new_time_desc)
2  new_time_desc
```

```
array(['00:00-00:15', '00:15-00:30', '00:30-00:45', '00:45-01:00',
       '01:00-01:15', '01:15-01:30', '01:30-01:45', '01:45-02:00',
       '02:00-02:15', '02:15-02:30'], dtype=object)
```

Power Generation for initial 10 blocks of Next Day

```
[ ]  1  Predicted_df.insert(0,"Time Desc",new_time_desc,True)
     2  Predicted_df
```

	Time Desc	UNCHAHAR4
1	00:00-00:15	226.513088
2	00:15-00:30	193.815232
3	00:30-00:45	243.026101
4	00:45-01:00	207.106472
5	01:00-01:15	182.519905
6	01:15-01:30	195.082570
7	01:30-01:45	207.723691
8	01:45-02:00	220.397982
9	02:00-02:15	233.062433
10	02:15-02:30	242.476996

If compared the performance of all the models with respect to the evaluation metrics (i.e., RMSE and R^2 score) then the following comparison table shall be helpful to refer –

Algo. ➡ Metrics ⬇	RNN	LSTM	ARIMA Model
RMSE	7791.47	7754.76	0.33
R2 (R-square) score	-30.10	-29.81	-11.50

Chapter 6

Results

Though we are done the prediction values and evaluation metrics result, these entities are still vague towards an unsophisticated brain (say, investors, businessmen, business analysts, etc.) who are more concerned about real life values. Hence, I must present some kind of pictorial representation to make anyone understand in the easiest possible way, the results of my analysis.

For putting forward my analysis result, I have to perform a few steps related to comparison. The predicted values from RNN, LSTM and ARIMA will be compared to the data values of the dataset of the next day. Moreover, I will try to establish a correlation among those values to get a clear idea of dependency (if it exists). The below implantation denotes the above mentioned:

Now that we have predicted the desired values through all the 3 models - RNN, LSTM and ARIMA and also evaluated their performance using metrics like - RMSE Score and R^2 (R-square) score, these can be tough to explain to an unspohisticated brain (someone who might not be having expertise in ML). Hence, I will be comparing these predicted results on the basis of numerical measure known as, `CORRELATION COEFFICIENT`. The relationship between the ***correlation coefficient matrix***, **R**, and the ***covariance matrix***, **C**, is 🔽

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} C_{jj}}}$$

NOTE ⚡ : ***Covariance Matrix*** is a square matrix giving the covariance between each pair of elements of a given random vector / 2 vectors (comparing elements pair-wise from both the vectors).

```
[ ]  1  # Lets import the necessary libraries
     2
     3  import numpy as np
     4  import pandas as pd
```

```
[ ]  1  # Mounting Google Drive
     2
     3  from google.colab import drive
     4  drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]  1  # Loading the dataset, to which comparison has to be done
     2
     3  df = pd.read_csv("drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Power datasets/FullSchedule-InjectionSummary-UNCHAHAR4(216)-06-01-2023.csv")
     4  df
```

	Time Block	Time Desc	UNCHAHAR4	Grand Total
0	1	00:00-00:15	257.81	257.81
1	2	00:15-00:30	257.81	257.81
2	3	00:30-00:45	257.81	257.81
3	4	00:45-01:00	257.81	257.81
4	5	01:00-01:15	257.81	257.81
...	...	...	...	...
91	92	22:45-23:00	396.75	396.75
92	93	23:00-23:15	324.75	324.75
93	94	23:15-23:30	257.81	257.81
94	95	23:30-23:45	257.81	257.81
95	96	23:45-24:00	257.81	257.81

96 rows × 4 columns

```
1  # Let us see the General information of the dataset
2
3  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Time Block   96 non-null     int64
 1   Time Desc    96 non-null     object
 2   UNCHAHAR4    96 non-null     float64
 3   Grand Total  96 non-null     float64
dtypes: float64(2), int64(1), object(1)
memory usage: 3.1+ KB
```

```
1  # Now that we only want the power generation values of the powerplant
2  # of the initial 10 blocks of the day, we will filter that out and store
3  # in a separate variable
4
5  data = df.filter(['UNCHAHAR4'])
6  data_comp = data['UNCHAHAR4'][:10]
7  data_comp
```

```
0    257.81
1    257.81
2    257.81
3    257.81
4    257.81
5    257.81
6    257.81
7    257.81
8    257.81
9    257.81
Name: UNCHAHAR4, dtype: float64
```

```
[ ]  1  # Converting the extracted data into Numpy Array
     2
     3  data_comp_array = data_comp.values
     4  print(data_comp_array)
```

```
[257.81 257.81 257.81 257.81 257.81 257.81 257.81 257.81 257.81 257.81]
```

```
     1  # Now we will import the predicted values of the models. These are
     2  # stored in Text files (.txt files)
     3
     4  # RNN
     5  filename1 = 'drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_RNN.txt'
     6  data_RNN = np.loadtxt(filename1, delimiter=' ', skiprows=0, dtype=float)
     7  data_RNN = np.round(data_RNN, decimals = 2)
     8  print("RNN : 🔽\n")
     9  print(data_RNN)
    10
    11  print("\n\n")
    12
    13  # LSTM
    14  filename2 = 'drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_LSTM.txt'
    15  data_LSTM = np.loadtxt(filename2, delimiter=' ', skiprows=0, dtype=float)
    16  data_LSTM = np.round(data_LSTM, decimals = 2)
    17  print("LSTM : 🔽\n")
    18  print(data_LSTM)
    19
    20  print("\n\n")
    21
    22  # ARIMA
    23  filename3 = 'drive/Othercomputers/My Laptop/CAPSTONE PROJECT/Forecast_ARIMA.txt'
    24  data_ARIMA = np.loadtxt(filename3, delimiter=' ', skiprows=0, dtype=float)
    25  data_ARIMA = np.round(data_ARIMA, decimals = 2)
    26  print("ARIMA : 🔽\n")
    27  print(data_ARIMA)
```

```
RNN : 🔽

[257.81 306.05 327.89 330.21 293.58 330.86 311.55 282.77 335.28 314.59]


LSTM : 🔽

[257.81 273.84 277.02 288.66 257.99 282.44 286.61 266.95 295.54 295.88]


ARIMA : 🔽

[226.51 193.82 243.03 207.11 182.52 195.08 207.72 220.4  233.06 242.48]
```

```python
# Now that we have loaded all the required vectors, it's time to compare them
# using the CORRELATION COEFFICIENT.

# RNN
df_Corr_RNN = pd.DataFrame({'Data_Original' : data_comp, 'Data_RNN' : data_RNN})
Corr_RNN = df_Corr_RNN['Data_Original'].corr(df_Corr_RNN['Data_RNN'])
print("Correlation Coefficient (RNN & Original Data) : " + str(float(Corr_RNN)))

# LSTM
df_Corr_LSTM = pd.DataFrame({'Data_Original' : data_comp, 'Data_LSTM' : data_LSTM})
Corr_LSTM = df_Corr_LSTM['Data_Original'].corr(df_Corr_LSTM['Data_LSTM'])
print("Correlation Coefficient (LSTM & Original Data) : " + str(float(Corr_LSTM)))

# ARIMA
df_Corr_ARIMA = pd.DataFrame({'Data_Original' : data_comp, 'Data_ARIMA' : data_ARIMA})
Corr_ARIMA = df_Corr_ARIMA['Data_Original'].corr(df_Corr_ARIMA['Data_ARIMA'])
print("Correlation Coefficient (ARIMA & Original Data) : " + str(float(Corr_ARIMA)))

```

```
Correlation Coefficient (RNN & Original Data) : nan
Correlation Coefficient (LSTM & Original Data) : nan
Correlation Coefficient (ARIMA & Original Data) : nan
```

Hence, no such correlation exists between the original values and the predicted values of the models, as `NaN` refers to `ZERO` correlation.

```python
# Lastly, have a look to the combined dataframe of all the data

Final_df = pd.DataFrame({'Data_Original' : data_comp_array , 'Data_RNN' : data_RNN , 'Data_LSTM' : data_LSTM , 'Data_ARIMA' : data_ARIMA})
Final_df
```

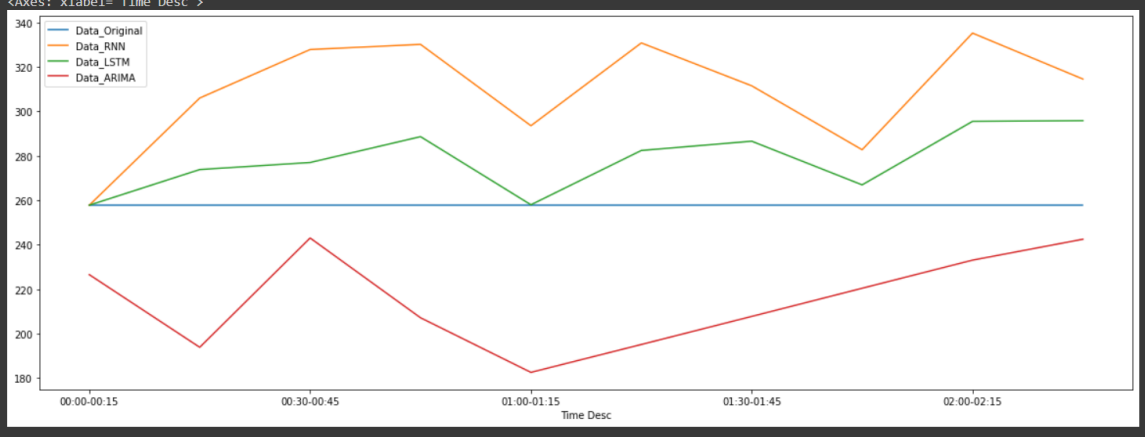
	Data_Original	Data_RNN	Data_LSTM	Data_ARIMA
0	257.81	257.81	257.81	226.51
1	257.81	306.05	273.84	193.82
2	257.81	327.89	277.02	243.03
3	257.81	330.21	288.66	207.11
4	257.81	293.58	257.99	182.52
5	257.81	330.86	282.44	195.08
6	257.81	311.55	286.61	207.72
7	257.81	282.77	266.95	220.40
8	257.81	335.28	295.54	233.06
9	257.81	314.59	295.88	242.48

```python
# Adding Time slabs as the first column

new_time_desc = df['Time Desc'][:10]
new_time_desc = np.asarray(new_time_desc)
Final_df.insert(0,"Time Desc",new_time_desc,True)
Final_df
```

	Time Desc	Data_Original	Data_RNN	Data_LSTM	Data_ARIMA
0	00:00-00:15	257.81	257.81	257.81	226.51
1	00:15-00:30	257.81	306.05	273.84	193.82
2	00:30-00:45	257.81	327.89	277.02	243.03
3	00:45-01:00	257.81	330.21	288.66	207.11
4	01:00-01:15	257.81	293.58	257.99	182.52
5	01:15-01:30	257.81	330.86	282.44	195.08
6	01:30-01:45	257.81	311.55	286.61	207.72
7	01:45-02:00	257.81	282.77	266.95	220.40
8	02:00-02:15	257.81	335.28	295.54	233.06
9	02:15-02:30	257.81	314.59	295.88	242.48

```
# Creating line graph for the above Data-Frame

Final_df.plot(x="Time Desc", y=["Data_Original", "Data_RNN", "Data_LSTM" , "Data_ARIMA"], kind="line", figsize=(20, 7))
```

```
<Axes: xlabel='Time Desc'>
```



Now that I have obtained the pictorial result(s), it is time to provide an explanation of the plot. **RNN** (Recurrent Neural Network) and **LSTM** (Long Short-Term Neural Network) can work with **non-stationary data**, something **ARIMA** (Auto-Regressive Integrated Moving Average) cannot claim to work with. Hence, for ARIMA, the data was passed through the procedure of **"De-Trending"** to make it stationary data. The prediction results clearly show that even though ARIMA is fed with stationary data, the deviation(s) of the predicted data points is way beyond what is shown by RNN and LSTM. Hence, ARIMA cannot work with small number of values. On the other hand, though RNN and LSTM showed better performance, LSTM surpassed RNN because RNN has the problem of vanishing gradients, i.e., the new weights will only be updated by considering only a few of the previous weights and not the very old ones. Hence, LSTM clearly wins the situation.

Let us have a look to the RMSE and R^2 score of the predicted values w.r.t. yo the original values –

```
[41]
# Lets calculate the RMSE and R-Square for the predicted values
# w.r.t. original data-points

from sklearn import metrics

# RMSE for LSTM model prediction
mse_lstm = metrics.mean_squared_error(data_comp_array, data_LSTM)
rmse_lstm = np.sqrt(mse_lstm)
print("LSTM ▶ RMSE : ", rmse_lstm)

# R-Square score for LSTM model prediction
r2_lstm = metrics.r2_score(data_comp_array, data_LSTM)
print("LSTM ▶ r2 : ", r2_lstm)
```

```
print("\n")

# RMSE for RNN model prediction
mse_rnn = metrics.mean_squared_error(data_comp_array, data_RNN)
rmse_rnn = np.sqrt(mse_rnn)
print("RNN ▶ RMSE : ", rmse_rnn)

# R-Square score for LSTM model prediction
r2_rnn = metrics.r2_score(data_comp_array, data_RNN)
print("RNN ▶ r2 : ", r2_rnn)

print("\n")

# RMSE for ARIMA model prediction
mse_arima = metrics.mean_squared_error(data_comp_array, data_ARIMA)
rmse_arima = np.sqrt(mse_arima)
print("ARIMA ▶ RMSE : ", rmse_arima)

# R-Square score for ARIMA model prediction
r2_arima = metrics.r2_score(data_comp_array, data_ARIMA)
print("ARIMA ▶ r2 : ", r2_arima)
```

```
LSTM ▶ RMSE :  24.434062699436623
LSTM ▶ r2 :  0.0


RNN ▶ RMSE :  56.42022500841342
RNN ▶ r2 :  0.0


ARIMA ▶ RMSE :  47.141354562634284
ARIMA ▶ r2 :  0.0
```

```
# You see, LSTM has the lowest RMSE score (or, just showing the lowest error
# rate) among all.
```

We know that the power of average (mean) value can be beneficial in analysis. Hence, I opted an extremely rudimentary level of hybrid approach – taking the mean of prediction value and comparing the same with the original values –

```
# Now that we have a plot of all th required data-points, let us try to do an
# experiment. I will now try to find out the average of the data-points
# from all the arrays of predicted values of (LSTM ,RNN, ARIMA), and compare
# that to the original values

import math

res_list = []
for i,j,k in zip(data_RNN,data_LSTM,data_ARIMA):
    res_list.append(float(("{0:.2f}".format(math.ceil((i+j+k)/3)))))

res_array = np.array(res_list)
print(res_array)
```

```
[248. 258. 283. 276. 245. 270. 269. 257. 288. 285.]
```

```
[42]  1  # Let us try the metrics
      2
      3  # RMSE for Resultant (AVG) values
      4  mse_res = metrics.mean_squared_error(data_comp_array, res_array)
      5  rmse_res = np.sqrt(mse_res)
      6  print("Resultant ▶ RMSE : ", rmse_res)
      7
      8  # R-Square score for Resultant (AVG) values
      9  r2_res = metrics.r2_score(data_comp_array, res_array)
     10  print("ARIMA ▶ r2 : ", r2_res)

Resultant ▶ RMSE :  17.75100278857507
ARIMA ▶ r2 :  0.0
```

NOTE⚡: The **R^2 score** is 0 (ZERO), that means the datapoints related to the predicted values are no-where near to the original data points (OR, if interpreting according to the line plot, the predicted data-points are far away than the original data-points).

```
  1  # Thus, taking the average of the prediction values, have significantly
  2  # lowered the error rate.
```

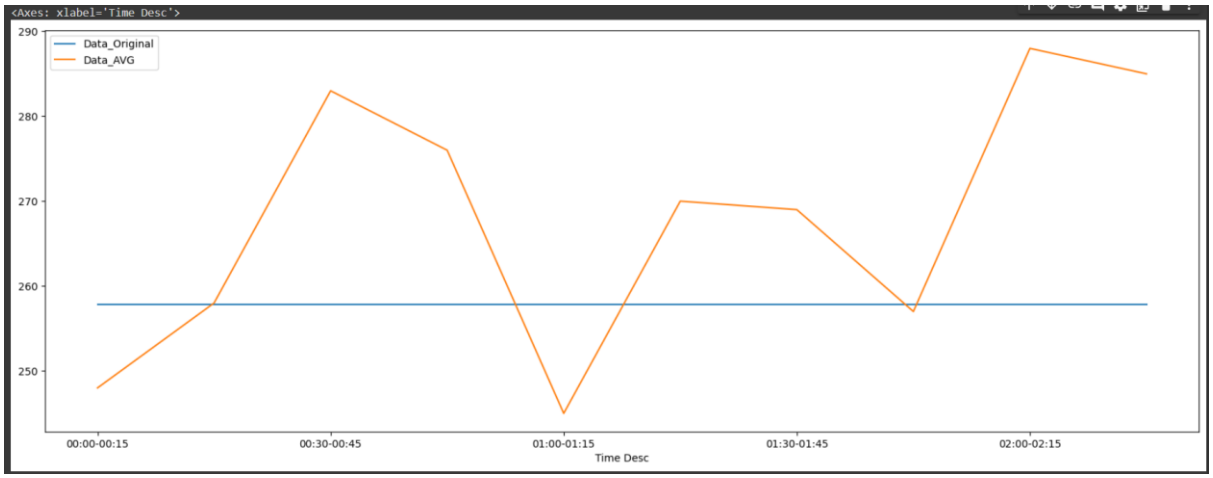
```
[34]  1  # Lets compare through a visual represenation using a line plot
      2
      3  # First I have to create a data frame with new columns and values
      4  Final_df_new = pd.DataFrame({'Data_Original' : data_comp_array , 'Data_AVG' : res_array})
      5  Final_df_new
```

	Data_Original	Data_AVG
0	257.81	248.0
1	257.81	258.0
2	257.81	283.0
3	257.81	276.0
4	257.81	245.0
5	257.81	270.0
6	257.81	269.0
7	257.81	257.0
8	257.81	288.0
9	257.81	285.0

```
# Adding Time slabs as the first column to the new dataframe

new_time_desc = df['Time Desc'][:10]
new_time_desc = np.asarray(new_time_desc)
Final_df_new.insert(0,"Time Desc",new_time_desc,True)
Final_df_new
```

	Time Desc	Data_Original	Data_AVG
0	00:00-00:15	257.81	248.0
1	00:15-00:30	257.81	258.0
2	00:30-00:45	257.81	283.0
3	00:45-01:00	257.81	276.0
4	01:00-01:15	257.81	245.0
5	01:15-01:30	257.81	270.0
6	01:30-01:45	257.81	269.0
7	01:45-02:00	257.81	257.0
8	02:00-02:15	257.81	288.0
9	02:15-02:30	257.81	285.0

```
# Creating line graph for the new Data-Frame

Final_df_new.plot(x="Time Desc", y=["Data_Original", "Data_AVG"], kind="line", figsize=(20, 7))
```



Hence, taking the average proved to be beneficial, as it has significantly reduced the RMSE score.

Chapter 7

Conclusion & Future Work

Efficiency of energy utility organizations is difficult to achieve. But in given time, some enhancements can surely be made to reduce errors and contribute to smooth functioning. This project was carried out by keeping in mind the amount of electricity to be generated for the initial quarter-hour time-slabs of next day by analysing current day's production. Thanks to the clean data acquired from the Government regulated websites, the analysis was commenced in relatively lesser time than expected. After choosing appropriate algorithms, namely RNN, LSTM and ARIMA, it was later discovered that ARIMA goes with a completely different approach than the other two – thanks to it inability to ingest non-stationary data. The introduction of timesteps decreased the loss score while model training as it elongates the whole dataset. The predictions however showed some unexpected visualizations, as time-series algorithms are way different that conventional supervised OR unsupervised algorithms, which shows a level of definiteness. Time-series algorithms are propelled by the concepts of fuzziness (as both being Neural Networks). The problem of "unexpected visualizations" also occurred with ARIMA, even though it was fed with stationary data, which was checked numerous times by ADF (Augmented Dickey Fuller) Test and ACF (Auto-Correlation Function) - PACF (Partial-ACF) for correlation among datapoints of current and lagged versions. If talking w.r.t. evaluation metrics (RMSE [Root Mean Square Error] and R^2 score), in case of RNN, the RMSE score is 7791.47 and, R^2 score is -30.10. In the case of LSTM, the RMSE score is 7754.76 and, R^2 score is -29.81. For ARIMA, the RMSE score is 0.33 and, R^2 score is -11.50. Clearly, we can see that ARIMA is dominating over the other algorithms. But that is only the training-n-testing component. To a naïve eye (i.e., those without the knowledge of ML), its hard to explain. Thus, a new comparative visualization is created, where the predicted values of the models are compared to that of the next day's power production data. Initially the data was passed through processes to find out whether any correlation exists or not to which, the outcome was negative for all the models. The final visualization tells a different story with an obvious explanation that ARIMA has a higher error rate as compared to RNN and LSTM. Also, as RNN has the problem of vanishing gradient, LSTM becomes the winner. Apart from the that, it was confirmed that the average of the predicted values of the models proved to be

providing lesser error rate than LSTM. The outcome of this project will benefit the companies of power production to inculcate that algorithm appropriate to the dataset they want to feed. Eventually, this will increase the efficiency of electricity generation and minimize the cost on equipment failures, excessive coal buying error and overall safety. Many web-articles have maintained that ARIMA is the best time-series algorithm, but they gave no evidence of its disadvantages w.r.t. real-world data. In this project I uncovered its major disadvantage – the usage of stationary data, which barely exists in the real world. The key take-away points from this project includes an above-rudimentary-level understanding of "Time-Series," the algorithms associated with it, the kind of data to work with, an exploration ability to find appropriate data source, correct usage of python libraries, data analysis and visualization skills, and lastly, storytelling. The existing works with this same project objective have not compared the analysis results of the models based on stationary and non-stationary data. Moreover, they have mostly given themselves a full-stop after obtaining the prediction results or evaluation metrics score. Hence, this project exists as a separate entity with absolute uniqueness. Future work includes implementation of an API which is connected to one of the trained models. The API will be inculcated in web sites of power utility companies where relevant data can be directly fed as parameters of the trained model. Moreover, the values can be predicted through streaming data, which will initially be very small but eventually become significant. For the initial values to be predicted, the algorithm shall be optimized to obtain a model which when trained produce minimum loss, thus shall reduce the intensity of error.

References

[1] Mohammad-Rasool Kazemzadeh, Ali Amjadian, Turaj Amraee. (2020). “A hybrid data mining driven algorithm for long term electric peak load and energy demand forecasting”. Energy, Volume 204, 117948, ISSN 0360-5442, https://doi.org/10.1016/j.energy.2020.117948.

[2] Ranran Li, Ping Jiang, Hufang Yang, Chen Li. (2020). “A novel hybrid forecasting scheme for electricity demand time series”. Sustainable Cities and Society, Volume 55, 102036, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2020.102036.

[3] Weiheng Jiang, Xiaogang Wu, Yi Gong, Wanxin Yu, Xinhui Zhong. (2020). “Holt–Winters smoothing enhanced by fruit fly optimization algorithm to forecast monthly electricity consumption”. Energy, Volume 193, 116779, ISSN 0360-5442, https://doi.org/10.1016/j.energy.2019.116779.

[4] Evangelos Spiliotis, Fotios Petropoulos, Nikolaos Kourentzes, Vassilios Assimakopoulos. (2020). “Cross-temporal aggregation: Improving the forecast accuracy of hierarchical electricity consumption”. Applied Energy, Volume 261, 114339, ISSN 0306-2619, https://doi.org/10.1016/j.apenergy.2019.114339.

[5] Shalika Walker, Waqas Khan, Katarina Katic, Wim Maassen, Wim Zeiler. (2020). “Accuracy of different machine learning algorithms and added-value of predicting aggregated-level energy performance of commercial buildings”. Energy and Buildings, Volume 209, 109705, ISSN 0378-7788, https://doi.org/10.1016/j.enbuild.2019.109705.

[6] Lei Lei, Wei Chen, Bing Wu, Chao Chen, Wei Liu. (2021). “A building energy consumption prediction model based on rough set theory and deep learning algorithms”. Energy and Buildings, Volume 240, 110886, ISSN 0378-7788, https://doi.org/10.1016/j.enbuild.2021.110886.

[7] Mel Keytingan M. Shapi, Nor Azuana Ramli, Lilik J. Awalin. (2021). “Energy consumption prediction by using machine learning for smart building: Case study in Malaysia”. Developments in the Built Environment, Volume 5, 100037, ISSN 2666-1659, https://doi.org/10.1016/j.dibe.2020.100037.

[8] S. Bowala, M. Makhan, Y. Liang, A. Thavaneswaran and S. S. Appadoo. (2022). "Superiority of the Neural Network Dynamic Regression Models for Ontario Electricity Demand Forecasting". IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 182-187, doi: 10.1109/CCECE49351.2022.9918212.

[9] A. Arjomandi-Nezhad, A. Ahmadi, S. Taheri, M. Fotuhi-Firuzabad, M. Moeini-Aghtaie and M. Lehtonen. (2022). "Pandemic-Aware Day-Ahead Demand Forecasting Using Ensemble Learning". IEEE Access, vol. 10, pp. 7098-7106, 2022, doi: 10.1109/ACCESS.2022.3142351.

[10] Y. E. UNUTMAZ, A. DEMİRCİ, S. M. Tercan and R. Yumurtaci. (2021). "Electrical Energy Demand Forecasting Using Artificial Neural Network". 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 2021, pp. 1-6, doi: 10.1109/HORA52670.2021.9461186.