A DATA-DRIVEN PERSPECTIVE ON INDIA'S RENEWABLE ENERGY GROWTH AND TRENDS: VISUALIZATION AND TIME SERIES ANALYSIS

Submitted in partial fulfilment of the requirements for the degree of

Master of Science

In

Business Statistics

by

NEDUNURI SAI SRIVIDYA

(22MBS0032)

Under the guidance of

Dr. JITENDRA KUMAR

School of Advanced Sciences

VIT, Vellore

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

May, 2024

DECLARATION

I hereby declare that the thesis entitled "A Data driven perspective on India's Renewable energy growth and trends: Visualization and time series analysis " submitted by me, for the award of the degree of *Master of Science in Business Statistics* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Jitendra Kumar.

I further declare that the work reported in this thesis has 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.

Place: Vellore

Date: 7/5/24

[illegible]

Signature of the Student

Nedunuri Sai Srividya

CERTIFICATE

This is to certify that the thesis entitled "A Data-Driven Perspective On India's Renewable Energy Growth And Trends: Visualization And Time Series Analysis" submitted by Nedunuri Sai Srividya (Reg. No.: 22MBS0032), School of Advanced Sciences, VIT, for the award of the degree of *Master of Science in Business Statistics*, is a record of bonafide work carried out by him / her under my supervision during the period, 03.01.2024 to 07.05.2024, as per the VIT code of academic and research ethics.

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 thesis fulfils the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 7/5/24

[illegible] 7/5/24.

Signature of the Guide

Department of Mathematics

SAS, VIT- Vellore

[illegible] 8/8/2024

Signature of External Examiner

[illegible] 8/5/2024

Head, Department of Mathematics
Dr. JAGADEESH KUMAR M.S

SAS, VIT- Vellore

ACKNOWLEDGMENT

With immense pleasure and deep sense of gratitude, I wish to express my sincere thanks to my guide Dr. Jitendra Kumar, School of Advanced Sciences, VIT, Vellore without his motivation and continuous encouragement, this research would not have been successfully completed.

I am grateful to the Chancellor of VIT, Vellore, Dr. G. Viswanathan, the Vice Presidents and the Vice Chancellor for motivating me to carry out research in the Vellore Institute of Technology, Vellore and also for providing me with infrastructural facilities and many other resources needed for my research.

I express my sincere thanks to Dr. Arunai Nambi Raj N, Dean, School of Advanced Sciences, VIT, Vellore, for her kind words of support and encouragement. I like to acknowledge the support rendered by my classmates in several ways throughout my research work.

I wish to thank Dr. JAGADEESH KUMAR M.S, Head of the Department of Mathematics, School of Advanced Sciences, VIT, Vellore for his encouragement and support.

I wish to extend my profound sense of gratitude to my parents and friends for all the support they made during my research and also providing me with encouragement whenever required.

[illegible]

Signature of the Student

Nedunuri Sai Srividya

ABSTRACT

The term "renewable energy" describes energy produced from naturally occurring resources, such as biomass, geothermal heat, wind, sunshine, and water (hydro). In contrast to fossil fuels, which are limited and release pollutants into the atmosphere and contribute to climate change when burned, renewable energy sources are thought to be sustainable and have negligible environmental effects.

The utilization of renewable energy sources is deemed essential in the fight against climate change and the shift to a more ecologically friendly and sustainable energy system. They have many advantages, including as lowering greenhouse gas emissions, improving energy security, generating employment, and fostering economic growth.

The report starts with a summary of the state of renewable energy in India, emphasizing investment patterns, regulatory measures, and technical developments that have impacted the sector's expansion in the last ten years. Visualization has been carried out to identify the growth and trends in various sources of renewable energy installation capacities and power generation in every state using PowerBi. In order to predict future patterns in renewable energy consumption, a variety of time series models are then used to the data, such as autoregressive integrated moving average (ARIMA) and other techniques. This analysis is helpful in identifying the future scope of renewable energy in India and its contribution in the energy sector.

Key words: Renewable energy, Sustainability, Solar energy, Time series analysis, PowerBi.

CONTENTS

LIST OF FIGURES

1.INTRODUCTION

1.1. Renewable energy:

With the increase in Energy Consumption every year, switching to sustainable energies is very much important rather than depending on the Fossil fuels or non-renewable energies which are major source of energy right now throughout the world. These Non-renewable energies have a significant impact on the environment by emitting greenhouse gases like Carbon dioxide, Methane, Nitrous Oxide etc which are very hazardous to the nature. Fossil fuels like these are the largest contributors for the environmental changes and climate impacts, being responsible for nearly 75% of the Greenhouse gas emissions and 90% of the Co2 emissions. This Greenhouse gas emissions are responsible for many climate changes like irregular rainfalls, increased global warming, Variations, and disruption of seasonal cycles etc. This causes a severe threat to human life and existence of other living beings on the earth.

Renewable energy is one of the greatest alternative energy sources. Renewable energy is defined as energy generated from naturally replenishing sources that are always available or can be replenished on a human timeline. These energy sources are considered renewable since they do not deplete when consumed and can be sustained for extended periods of time. Common renewable energy sources include:

Solar energy is the use of sunlight to create electricity or heat water for a variety of purposes via photovoltaic panels or solar thermal systems.

Wind energy is the use of wind's kinetic energy to power turbines and create electricity.

Hydroelectric energy is produced by harnessing the gravitational force of flowing or falling water, usually through dams or run-of-river systems.

Biomass energy is the use of organic resources such as wood, agricultural wastes, or garbage to generate heat, electricity, or biofuels via processes such as combustion and gasification.

All these renewable energies have zero or minimal negative impact on the nature making them the best alternative for the non-renewable energy sources and generation of electricity for household and daily use from these renewable sources helps us in reducing the burden of greenhouse gas emissions on earth very drastically and also it is commercially viable too. Countries like Iceland have been using almost 100% renewable energy for all the purposes.

1.2. Renewable Energy Forecasting:

The future of the energy world is Renewable sources and the forecasting the demand or need for this Renewable energy is very much important in order to estimate the production and the other factors like cost, equipment etc. These estimations and forecasting will help the bureaucrats and the officials in power to make plans accordingly and take necessary actions for a better and safer environment. This involves various regression and time series models along with the advanced machine learning models and neural networks. All these models give an idea and forecast on the required energy based on the past data or the data obtained from previous periods. The effective integration and control of renewable energy sources, such as hydroelectric, solar, and wind power, into the grid depends heavily on renewable energy forecasting. The production of renewable energy is intrinsically unpredictable and reliant on elements like the weather, the time of day, and the season, in contrast to conventional energy sources like fossil fuels. Therefore, precise forecasting plays a crucial role in maintaining grid stability, scheduling maintenance, and optimising resource allocation. Models of renewable energy are created by combining meteorological information, past energy production records, and sophisticated modelling techniques. Utilising advanced algorithms such as neural networks, machine learning, and statistical techniques, these models examine large datasets and identify trends that impact the production of renewable energy. These models have the ability to produce forecasts with a range of time horizons, from minutes to days ahead, by integrating variables including wind speed, solar irradiance, cloud cover, and weather patterns.

The intrinsic variability of weather patterns, the influence of complicated terrain, and the uncertainties surrounding long-term climate trends remain obstacles in renewable energy forecasting methodologies, despite notable breakthroughs in the field. Furthermore, because of the variety of these installations and how they interact with regional microclimates, the incorporation of distributed sources of renewable energy at the community level poses additional forecasting issues. Improved prediction accuracy and reliability could be achieved by more research and development into renewable energy forecasting techniques, as well as improvements in data analytics and sensor technology. Improving prediction skills not only helps renewable energy systems run more smoothly, but it also makes it easier to move towards an energy infrastructure that is more resilient and sustainable. Accurate forecasting will continue to be a vital tool for optimising the advantages of clean, renewable energy sources while reducing their environmental impact as the renewable energy industry grows internationally.

1.3. Objectives:

The main objective of this research is firstly to collect the state wise data about monthly generation of energy from renewable sources such as solar, wind, bio mass, Small hydro etc and also the data about the installed capacities of renewable energies in various states.

The next objective is to visualize this data to gather insights about various states generation of energy from these renewable sources and the information about the capacities of these energies installed in those states.

And the third objective of this research is to forecast the monthly power generation in the states for the coming periods in order to understand the future requirement or demand of the energy.

Various time series models like Autoregressive Integrated Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving Average (SARIMA) and Machine learning models like Gradient boosting, Random Forest and Extreme Gradient Boosting (XGBoost) with different hyperparameter tuning techniques like GridSearch, Random search have been used to forecast the monthly power generation of various states, And also an ensemble model of all these models has also been tried in this analysis to compare the forecast of different models.

2. LITERATURE REVIEW

Renewable energy forecasting is very important and necessary step in the field of energy and its impact on the environment. In recent years, a significant amount of work and research has been done on the forecasting techniques and mechanisms in the field of renewable energy. Many methods like Time series analysis and machine learning models have been used in this forecasting. Along with this few deep learning models and neural networks have also been used for the forecasting with much more advanced developments.

Huaizhi Wang, Zhenxing Lei, Xian Zhang, Bin Zhou, Jianchun Peng(2019) proposed the application of deep learning methods for renewable energy forecasting. They have focussed more on the methods such as recurrent neural networks and convolutional neural networks to compare their performance with the traditional forecasting methods. Both these methods have given decent results and became successful in capturing complex temporal and nonlinear patterns. (Huaizhi Wanga, 2019)

It involves identifying the practical problems in model selection such as data requirements, scalability etc. They have also examined various data preprocessing techniques and error correction techniques which is helpful in increasing the accuracy of the forecast. (Huaizhi Wanga, 2019)

In the paper proposed by José R. Andrade and Ricardo J. Bessa (2016) , numerical weather predictions (NWPs) have been used for wind and solar energy forecast integrating the capturing of weather conditions which effects these renewable energy sources distributed over different geographical areas. (Bessa, 2017)

They have also combined this NWPs with machine learning models like support vector machines and artificial neural networks to improve the accuracy. Real world data has been used for training and testing the models. A decent accuracy has been achieved with this combination of high resolution NWPs and machine learning algorithms in forecasting the renewable energy. (Bessa, 2017)

In another review article written by Ricardo J. Bessa , CorinnaMöhrlen , Vanessa Fundel ,Malte Siefert , Jethro Browell ,Sebastian Haglund El Gaidi , Bri-Mathias Hodge , Umit Cali and George Kariniotakis, The authors have discussed various uncertainties that can arise in renewable energy adoption. One major uncertainty is the weather or meteorological conditions which effects the renewable energy demand fluctuations and market. The authors tried to discuss the challenges and solutions for these uncertainties. They also suggested or examined the state of the art techniques or machine learning algorithms in predicting or quantifying these uncertainties which includes probabilistic methods, stochastic optimization models. They have also examined the performance of short term forecasting uncertainty in long term decisions. (Bessa, 2017)

In the paper discussed by Tanveer Ahmad and Huanxin Chen (2019) , they have explored the usage or application of various deep learning algorithms in smart energy forecasting among multiple scales. With increase in the scale of the renewable energy day by day, the traditional forecasting methods might not be able to capture the patterns and trends efficiently in the large scale and high variability data. (Tanveer Ahmad, 2019)

So the authors have considered the application of deep learning methods like recurrent neural networks (RNNs) and convolutional neural networks (CNNs) etc in dealing with multi scale energy forecasting tasks. The objective of the deep learning models is to produce precise predictions over a range of time periods and geographical locations by combining past energy data with pertinent external elements like demand patterns or weather forecasts. The research shows how deep learning techniques may overcome the difficulties associated with multi-scale energy forecasting and outperform conventional forecasting techniques through rigorous testing and validation using real-world datasets. (Tanveer Ahmad, 2019)

In a paper discussed by Aowabin Rahman, Vivek Srikumar, and Amanda D. Smith, the authors focussed on using deep recurrent neural networks in forecasting the electricity consumption. The prediction or forecasting of the electricity consumption plays a pivotal role in understanding the demand and energy management. It helps in building demand side management strategies. (Aowabin Rahmana, 2017)

The deep recurrent neural networks are capable of capturing both short term and long-term fluctuations in the data and understand the complex relationships. Through the usage of real-world data sets as validation sets, it has been observed that these deep recurrent neural networks have performed better than the traditional methods with better accuracy. The paper also discussed optimizing utility resource planning, demand response programs. The research advances the field's understanding of power consumption forecasts through the application of deep learning techniques, which helps to create more intelligent and environmentally friendly energy systems for both the residential and commercial sectors. (Aowabin Rahmana, 2017)

In the paper written by Roberto Corizzo, Michelangelo Ceci, Hadi Fanaee-T, and Joao Gama, the authors discussed multi-aspect forecasting in renewable energy. Multi aspect forecasting includes the forecasting of multiple aspects in renewable energy generation like quantity, temporal patterns etc. In order to maximise the integration of renewable energy sources into the grid, this all-encompassing strategy strives to produce more precise and trustworthy projections, acknowledging the complex nature of these systems. (Roberto Corizzo, 2020)

The authors have considered predictive clustering trees along with the state-of-the-art methods. Further, they have also discussed the challenges coming with the multi aspect renewable energy forecasting such as weather and complex interactions between energy systems etc. (Roberto Corizzo, 2020)

In the paper authored by Ye Ren, P.N. Suganthan, and N. Srikanth, the authors have discussed the idea of using ensemble methods which combines multiple individual state of the art mechanisms like bagging, boosting, stacking etc. These methods are helpful in dealing with the drawbacks faced in the individual models. (Ye Ren, 2015)

They have also discussed the integration of machine learning algorithms like random forests, gradient boosting, and neural networks to improve the forecasting accuracy. The wind and solar energy forecasting plays a significant role in optimizing the power generation, maintaining the system and integrating the renewable energy sources in the existing power grids. (Ye Ren, 2015)

3.METHODOLOGY

3.1. Time series Analysis and Forecasting:

Time series forecasting is a statistical technique of forecasting or predicting the future values based on the past values or the historical data points in a timeline or chronological order. It is a very useful and important technique in many fields such as finance, meteorology like weather forecasting, Sales forecast etc. This time series analysis is useful in identifying the seasonal trends, patterns and dependencies or interrelation within the data points to make accurate future data points.

The most traditional time series forecasting models include Autoregressive integrated moving average (ARIMA), Exponential smoothing models, Advanced machine learning algorithms like neural networks etc. These models are useful in analysing the past data points and identifying the underlying patterns, trends and the interrelations between data points.

The quality of the historical data decides the accuracy of the forecast or predictions of the future timeline. The accuracy of the forecast also depends on other various factors like the selection of the model for forecast, rightful assumptions about the data, External influences etc. All these factors should be considered carefully to improve the predictive performance of the models.

Time series forecasting is not without its difficulties. Common challenges addressed by analysts include choosing the best forecasting horizon, handling outliers, and handling missing data. Furthermore, conventional forecasting models may become inadequate due to the dynamic character of many real-world systems, necessitating ongoing improvement and modification.

Despite these difficulties, time series forecasting is nevertheless a very useful tool that helps researchers and businesses alike predict market trends, make well-informed decisions, and deploy resources effectively. The accuracy and applicability of time series forecasting are constantly improving due to technological and data analytics breakthroughs, enabling organisations to take advantage of opportunities and manage uncertainty in a constantly changing global environment.

3.1.1. Autoregressive Integrated Moving Average (ARIMA):

Autoregressive Integrated Moving Average (ARIMA) is a very traditional time series forecasting model which is a combination of autoregression, differencing (integrated part) and moving average components. It is useful in identifying various temporal patterns and trends in the data. With the use of a linear relationship between previous observations and the order of their differences, ARIMA models are used to predict future values. The ARIMA models generally have the order (p.d.q) where p represents the autoregression parameter, d represents the differencing order and q represents the moving average parameter.

The autoregression component develops a model for the observation and its lagged values which in turn gives the dependency of the observation on its past values. The value of p is the number of lagged observations included in the model.

The differencing component in the ARIMA model is to make the data stationary. This is because the primary assumption of ARIMA is the stationarity of the data. Stationarity is the independence of the data on the period at which they are taken. The data with trends or seasonality etc will definitely effect the forecast values and thus it is not stationary. This differencing is useful in converting such nonstationary data into stationary data.

The moving average component establishes a relationship between the observations and its residuals with a moving average model on lagged observations. The value of q is the number of lagged residuals.

ARIMA models are good in forecasting for short-term and not really advisable for long term predictions and data with high volatility. Notwithstanding, ARIMA continues to be a popular and useful method in time series forecasting, especially when modelling univariate time series data that has a distinct temporal structure. It is a flexible option for a wide range of forecasting applications across many industries because of its simplicity, interpretability, and capacity to capture both short-term variations and long-term trends.

3.1.2. Seasonal Autoregressive Integrated Moving average (SARIMA):

Seasonal Autoregressive Integrated Moving average (SARIMA) is an extension of traditional ARIMA model with seasonal components to identify and capture the underlying seasonal patterns in the data which cannot be achieved properly by ARIMA. SARIMA is useful when the data is having recurring seasonal patterns. The examples for recurring seasonal patterns can be monthly sales data, meteorological data etc.

SARIMA is similar to ARIMA in that it has seasonal variations in addition to the three primary components of autoregressive (AR), differencing (I), and moving average (MA). SARIMA (p, d, q) (P, D, Q) m is the notation for the seasonal ARIMA model, in which (p, d, q) stands for the non-seasonal ARIMA parameters, (P, D, Q) for the seasonal ARIMA parameters, and "m" for the number of time points in each seasonal cycle.

The seasonal component of SARIMA explains the regular variations in the data that are seen at specific intervals, like patterns that occur every day, month, or year. SARIMA can more accurately predict outcomes over extended periods of time by including seasonal variables into the model, which helps the model better capture the underlying seasonal patterns.

Compared to typical ARIMA models, SARIMA models have a number of benefits when handling data from seasonal time series. For data with significant seasonal swings, they can produce more accurate forecasts and offer a more reliable framework for capturing complicated seasonal trends. Furthermore, SARIMA models can be applied widely in a variety of sectors since they are adaptive and flexible to diverse seasonal patterns.

Although SARIMA is a powerful tool, it is not without limitations. For example, in order to effectively estimate the model parameters, a significant quantity of historical data must be available, and choosing and understanding the right seasonal parameters can be challenging.

3.2. Machine Learning Models and Forecasting:

3.2.1. Extreme Gradient Boosting (XGBoost):

Extreme Gradient Boosting shortly known as XGBoost is a widely used machine learning algorithm that implements the idea of gradient boosting. It is a supervised learning algorithm. It is highly recommended because of its efficiency, ability to deal with large data sets and exceptional predictive performance. Fundamentally, XGBoost creates a powerful predictive model by optimising a predetermined objective function and gradually adding weak learners (decision trees). Using the residuals—the differences between expected and actual values—of the previous model, a new tree is fitted in each iteration of the method with the goal of minimising the total prediction error.

Decision Trees:

XGBoost uses base learners, or decision trees, which are straightforward models that decide what to do depending on features in the input. Every decision tree is trained to forecast the residuals of the preceding model, which essentially extracts the remaining patterns from the data that the earlier trees were unable to identify.

Gradient Boosting:

Using a gradient boosting framework, XGBoost trains each new tree to minimise the loss function with regard to the loss function's negative gradient. As a result, XGBoost can focus on the situations in which the model performs badly and use this information to iteratively repair the mistakes made by earlier models.

Gradient boosted trees are a concept introduced by XGBoost. Using this method, the new model is trained to forecast the loss function's negative gradient in relation to the predictions made by the ensemble. This sophisticated method enhances the convergence of the model and speeds up the learning process.

Gradient Descent Optimisation:

XGBoost minimises a regularised objective function, which is made up of a regularisation term and a loss function, in order to optimise the new tree model. The regularisation term penalises complicated models to prevent overfitting, while the loss function quantifies the difference between the actual target values and the predicted values. To expedite the training process, XGBoost employs effective optimisation methods including approximation tree learning and coordinate descent.

Regularisation:

Regularisation techniques are integrated into the model training process by XGBoost in order to better generalise and prevent overfitting. This consists of regularisation terms L1 (Lasso) and L2 (Ridge) that are introduced to the objective function and which promote simplicity and penalise complex models.

Lasso Regularization (L1):

By using Lasso regularisation, the objective function gains a penalty component proportional to the absolute values of the model parameters. Mathematically, it increases the loss function by multiplying the total of the coefficients' absolute values by a regularisation parameter (λ). Because it tends to decrease some coefficients to exactly zero, hence deleting some characteristics from the model, this promotes sparsity in the model.

The alpha parameter in XGBoost determines the intensity of the L1 regularisation term and controls the L1 regularisation. More regularisation and sparsity in the model are the outcomes of a higher alpha value.

Ridge Regularisation (L2):

The objective function gains a penalty component proportional to the square of the model parameters through the use of Ridge regularisation. In mathematical terms, it increases the loss function by multiplying the sum of the squares of the coefficients by a regularisation parameter (λ). Ridge regularisation pushes the coefficient values towards a more equal distribution within the model, but not necessarily to zero.The lambda parameter in XGBoost regulates L2 regularisation by indicating the intensity of the L2 regularisation term. Greater regularisation and a stop to the model overemphasising any one characteristic are achieved with larger values of lambda.

Learning rate:

The learning rate parameter, known as shrinkage, is a feature of XGBoost that regulates how much each new tree contributes to the final prediction. Better generalisation and more stable convergence might result from a lower learning rate, which calls for the addition of additional trees to the ensemble.

Feature Importance:

The significance of features in the prediction process can be assessed through the mechanism offered by XGBoost. Reliability and contribution to loss function reduction of features are taken into account while determining feature significance scores for decision trees.

3.2.2. Hyper Parameter Tuning:

Hyper Parameters are the settings which are not related to the data in the machine learning model. These parameters control or influence the behaviour of the machine learning model in the training phase and impacts the model in a significant manner. Hyperparameters are decided before the training phase of the model and are constant throughout the implementation.

Hyperparameters include learning rate, Hidden layers and units, Regularization parameters, Batch size etc. The process of selection of the proper hyperparameters and defining the suitable values for them in order to determine the ideal set of parameters that gives the best or optimal performance of the model is called hyperparameter tuning. Hyperparameter tuning includes searching for the best and optimal parameter space that increases or optimizes the metrics like accuracy, recall etc on the testing or validation set.

There are different techniques for hyperparameter tuning like Grid search, Random Search, Bayesian Optimization etc.

3.2.2.1. Grid search:

Grid search is one of the hyperparameter tuning techniques used to find the best set of hyperparameters from the hyperparameter space. It involves steps like defining a grid, cross validation, evaluating the performance and selecting the best combination. In defining the grid we define the parameters with different values. Afterwards, you run k-fold cross-validation for every set of hyperparameters the grid contains. When a model is trained and assessed k times, utilising a different subset as the validation set and the remaining subsets as the training set, the training dataset is divided into k subsets. This technique is known as k-fold cross-validation. Based on the performance on the validation set, you choose the best set of the parameters and is used on the training dataset. Grid search is a simple and easy way to perform hyperparameter tuning but it can be expensive in terms of computation especially in the cases of larger data sets.

3.2.2.2. Randomsearch:

Similar to Grid search, in random search also the first step is to define the search space and values of the parameters. But the main or significant difference between grid search and random search is the selection of combinations of parameters. Unlike grid search, Random search does not evaluate or go through all possible combinations of parameters. Instead, random search evaluates the combinations by selecting them randomly. The values are chosen randomly for each parameter within its specified range. After selection of this combinations, the steps are same as the ones which are performed in Grid search, and the best combination will be selected to train the model.

Random search is comparatively better than grid search because of less computational time by selecting the combinations in a random manner in a large hyperparameter space. However, because it depends on random sampling, there is a chance that the optimal hyperparameters won’t always be found.

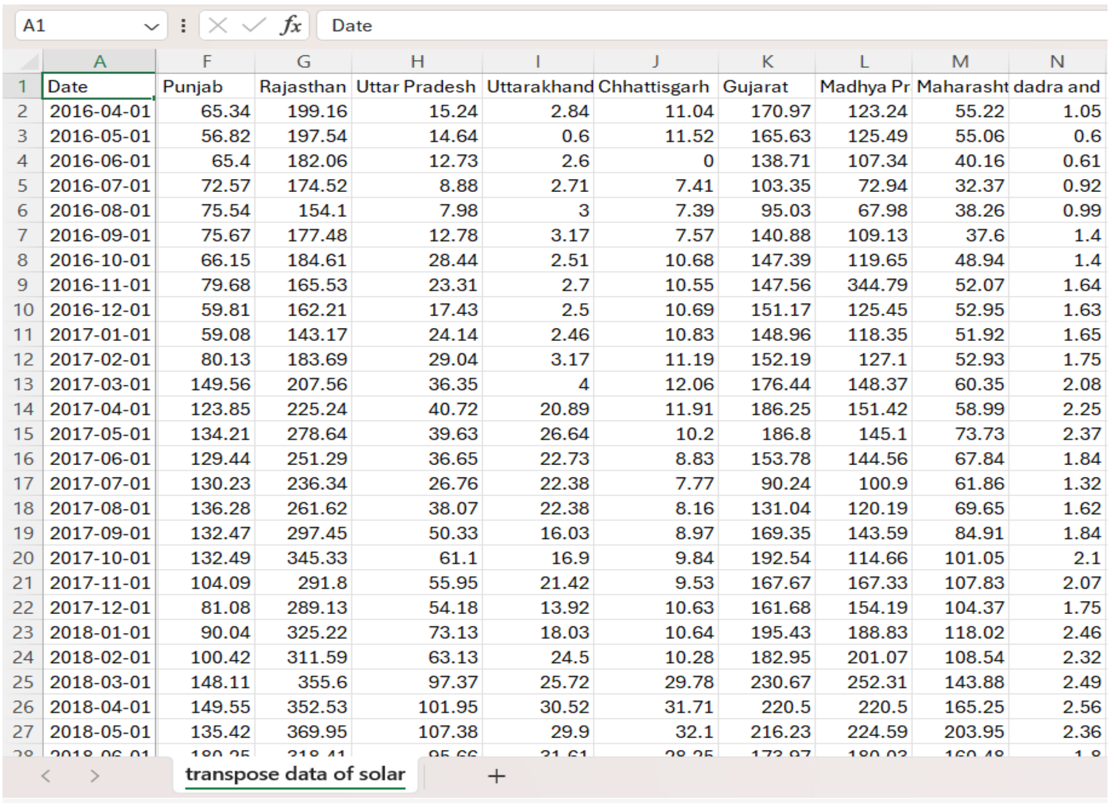
4.DATA, LANGUAGE AND TOOLS

4.1. Data:

The data for any timeseries forecasting should be a sequential data collected or recorded over a certain time period. In this study, the data used for the analysis has been collected from the Ministry of the Renewable Energy, India. The renewable energies considered in this analysis are Solar energy, Wind energy, Bio energy, Small Hydro (up to 25MW).

The data of the installed capacities of these energies in each state and the monthly power generation in each state throughout the period of April 2016 to February 2024 has been used for the analysis and forecasting of the power generation.

The solar energy industry is also dependent on the common household power generation along with the industrial plants where as the other power generation sources like bio energy, Wind energy and Small hydro are mostly dependent on the industries. Solar energy is one of the rapid growing renewable energies in India. The goal of Indian government is to install 100000 MW of solar energy grid in India.



A1 | *fx* Date

	A	F	G	H	I	J	K	L	M	N
1	Date	Punjab	Rajasthan	Uttar Pradesh	Uttarakhand	Chhattisgarh	Gujarat	Madhya Pr	Maharasht	dadra and
2	2016-04-01	65.34	199.16	15.24	2.84	11.04	170.97	123.24	55.22	1.05
3	2016-05-01	56.82	197.54	14.64	0.6	11.52	165.63	125.49	55.06	0.6
4	2016-06-01	65.4	182.06	12.73	2.6	0	138.71	107.34	40.16	0.61
5	2016-07-01	72.57	174.52	8.88	2.71	7.41	103.35	72.94	32.37	0.92
6	2016-08-01	75.54	154.1	7.98	3	7.39	95.03	67.98	38.26	0.99
7	2016-09-01	75.67	177.48	12.78	3.17	7.57	140.88	109.13	37.6	1.4
8	2016-10-01	66.15	184.61	28.44	2.51	10.68	147.39	119.65	48.94	1.4
9	2016-11-01	79.68	165.53	23.31	2.7	10.55	147.56	344.79	52.07	1.64
10	2016-12-01	59.81	162.21	17.43	2.5	10.69	151.17	125.45	52.95	1.63
11	2017-01-01	59.08	143.17	24.14	2.46	10.83	148.96	118.35	51.92	1.65
12	2017-02-01	80.13	183.69	29.04	3.17	11.19	152.19	127.1	52.93	1.75
13	2017-03-01	149.56	207.56	36.35	4	12.06	176.44	148.37	60.35	2.08
14	2017-04-01	123.85	225.24	40.72	20.89	11.91	186.25	151.42	58.99	2.25
15	2017-05-01	134.21	278.64	39.63	26.64	10.2	186.8	145.1	73.73	2.37
16	2017-06-01	129.44	251.29	36.65	22.73	8.83	153.78	144.56	67.84	1.84
17	2017-07-01	130.23	236.34	26.76	22.38	7.77	90.24	100.9	61.86	1.32
18	2017-08-01	136.28	261.62	38.07	22.38	8.16	131.04	120.19	69.65	1.62
19	2017-09-01	132.47	297.45	50.33	16.03	8.97	169.35	143.59	84.91	1.84
20	2017-10-01	132.49	345.33	61.1	16.9	9.84	192.54	114.66	101.05	2.1
21	2017-11-01	104.09	291.8	55.95	21.42	9.53	167.67	167.33	107.83	2.07
22	2017-12-01	81.08	289.13	54.18	13.92	10.63	161.68	154.19	104.37	1.75
23	2018-01-01	90.04	325.22	73.13	18.03	10.64	195.43	188.83	118.02	2.46
24	2018-02-01	100.42	311.59	63.13	24.5	10.28	182.95	201.07	108.54	2.32
25	2018-03-01	148.11	355.6	97.37	25.72	29.78	230.67	252.31	143.88	2.49
26	2018-04-01	149.55	352.53	101.95	30.52	31.71	220.5	220.5	165.25	2.56
27	2018-05-01	135.42	369.95	107.38	29.9	32.1	216.23	224.59	203.95	2.36

Fig 1. Monthly solar power generation

4.2. Power BI:

Power BI is a very powerful business visualization tool developed by Microsoft. With its user-friendly interface and wide range of visualization options Power BI helps the users to visualize their raw data and gain insights from the reports and dashboards.

The vast collection of editable graphs, charts, and other visual components that Power BI visualisation offers is one of its main advantages. To effectively express diverse types of data and insights, users may select from a variety of chart formats, including scatter plots, pie charts, bar charts, line charts, and heatmaps, among others. Furthermore, Power BI has sophisticated functions like drill-down, filtering, and cross-filtering that let users interactively examine and evaluate data at various granularities of information.

The smooth integration of Power BI visualisation with other Microsoft services and products, including Excel, Azure, and SQL Server, is another noteworthy feature. With the help of this integration, users can quickly establish connections to data sources, import data into Power BI, and produce dynamic visualisations that show changes and updates to the underlying data in real time. Additionally, Power BI offers data modelling and transformation features that enable users to prepare their data for visualisation by cleaning, shaping, and enriching it.

A vast array of formatting choices and customisation tools are also provided by Power BI, enabling users to modify their visualisations in accordance with certain branding standards and design specifications. Users may alter fonts, colours, labels, and other visual components to produce visually appealing and polished reports and dashboards that appeal to their target audience.

Moreover, Power BI facilitates collaboration and sharing by allowing users to share their dashboards and reports directly with stakeholders and colleagues or post them to the Power BI service. The ability for users to view and engage with the same set of visualisations from any location and on any device promotes collaboration, decision-making, and knowledge sharing inside organisations.

4.3. Python:

Python is one of the powerful and versatile programming languages. It is best known for its simple nature and also user-friendly structure. It is the recommended choice for all the beginners in the coding field and also for experienced coders. It is created by Guido van Rossum in the late 1980s.

The key strength of the python is its collection of libraries which are useful in performing humongous number of tasks in the field of analytics and software development. A thriving community of developers is fostered by this rich environment, which shortens development times and promotes code reuse. These developers also contribute libraries and frameworks to further expand Python's capabilities.

Python’s syntax is heavily dependent on the indentation to define the blocks instead of braces and keywords making it easier for the user ensuring clarity and readability. This simpler syntax makes python easy to understand and maintain in a team environment.

Python is very much useful in the field of data science and analytics, machine learning and artificial intelligence. Strong tools for data analysis and manipulation are provided by libraries like NumPy, pandas, and scikit-learn, while frameworks like TensorFlow and PyTorch make it simple for developers to create complex machine learning models.

Python is also platform independent which makes it compatible with various kinds of operating systems and platforms. The code written on one kind of operating system easily runs on the other kind. This flexibility helps developers taking python as their first choice to develop anything from desktop software to mobile applications.

This easy syntax and countless number of functions makes Python user-friendly and best choice for the learners of all levels.

4.3.1. Jupyter Notebook

Jupyter Notebook is an open-source application which is initially developed as an interface to work on python but now has more than 40 programming languages like R, Julia and Scala. Jupyter notebook supports various kinds of data analysis, scientific computing etc.

One of the main advantages of jupyter notebook is its user interface which is very interactive and helps the users to write and run the codes in a dynamic environment. It also enables the users to write markdown text or raw content in the cells along with the code. Users can execute each cell in the code individually or collectively.

The inclusion of Markdown and Latex in jupyter notebook allows the users to make effective documentation with code, text, visualizations. It also provides various kinds of visualizations with built-in libraries. Jupyter notebook also laid the foundation for other projects like JupyterLab, Binder etc.

All things considered, Jupyter Notebook's blend of interactivity, adaptability, and user-friendliness has made it an essential resource for scholars, instructors, and data scientists worldwide, enabling them to investigate, evaluate, and convey their work with unmatched flexibility and effectiveness.

4.3.2. Libraries used:

1.Pandas:

Pandas is a powerful python open source library which is useful for data manipulation and analysis. It provides data structures to store and use the data in an efficient manner. It is very much useful in dealing with large data sets. We can perform various tasks like data cleaning, filtering, grouping etc using this pandas library functions.

Pandas provide Series and DataFrame structures to handle the large relational data sets. Series is a one-dimensional array like structure whereas DataFrame is a two-dimensional structure like a table resembling a spreadsheet.

Using Pandas, we can access the data from various file formats like Excel, CSV, SQL etc. It is also useful in processing time series data.

2.Numpy:

Numpy is the basic library which is used for mathematical computations in python. The main and important feature of Numpy is its ability to deal with n-dimensional arrays and matrices. Using this library we can perform various mathematical and scientific computations on large datasets. Many statistical analysis operations can also be performed using this library.

It is widely used in the fields of data science, machine learning and other scientific analysis areas.

3.Matplotlib:

Matplotlib is a visualization library which is useful in creating static, interactive visualizations in python. It offers various types of plots, charts, and visualizations with customization. It is helpful in conducting a visualization analysis using various kinds of bar plots, histograms, three-dimensional plots and drawing insights from the large datasets.

4.Pmdarima:

Pyramid’s AutoARIMA is a python library is a library which is useful in fitting an Autoregressive Integrated Moving average (ARIMA) model to a time series data. ARIMA models are used for analysing a time series data and forecasting the future values using that data. However selecting the suitable order for fitting an ARIMA model is a challenging task. This challenge can be tackled using the autoARIMA function in the Pmdarima library which automates the process of selecting the order and checks all the orders and decides the best order using the information criterion values like AIC,BIC etc. This library is also useful in dealing with the data having seasonal components by supporting SARIMA models. It also offers diagnostic tools for models like residual analysis and model performance metrics etc.

5.Statsmodels.tsa:

This library provides functions for interpreting various statistical models and hypothesis testing. It contains various statistical models like linear regression models, Generalized linear models etc. It is also useful in time series analysis since it contains AutoRegressive Integrated Moving Average (ARIMA), Seasonal ARIMA (SARIMA), Vector AutoRegressive (VAR), and State Space Models. It also includes functions for conducting various statistical hypothesis tests.

It also has a feature of integrating with other python libraries allowing users to combine statistical models with data manipulation and visualization functions.

6.Scikit-learn:

Scikit-learn which is often written as sklearn is a python machine learning package. It offers various tools and functions of many different supervised and unsupervised learning methods like support vector machines (SVMs), Decision trees, Random Forest etc.

This package is very much useful in the applications of data science and machine learning like natural language processing (NLP), predictive analysis, dimensionality reduction, Clustering etc using these machine learning models.

7.XGBoost:

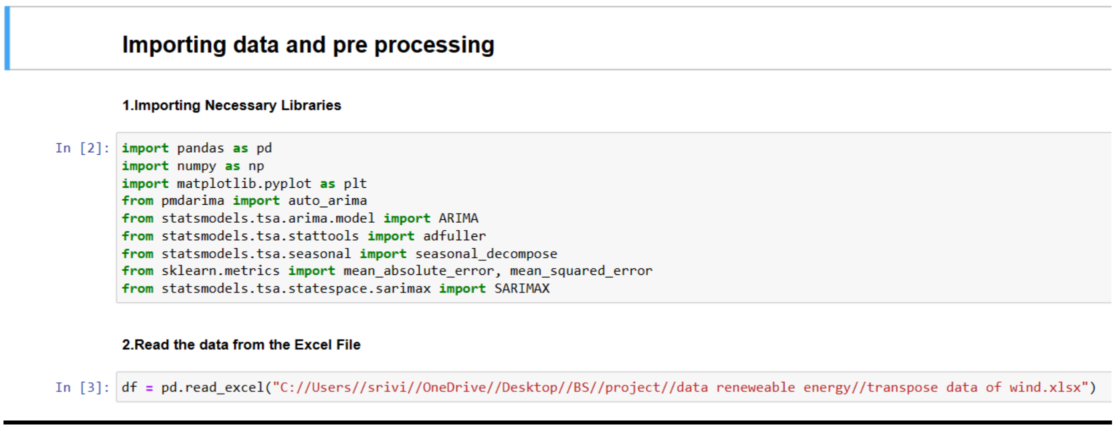
XGBoost is an open-source library designed specifically for the implementation gradient boosting algorithms of machine learning regression and classification goals. XGBoost is suitable for both regression and classification tasks making it an effective choice for predictive analysis or forecasting. It uses the concept of boosting where the decision trees are corrected in every step based on their previous errors. This library is known for its scalability, speed and effective performance.

8.Seaborn:

Seaborn is also a python visualization library which is based on Matplotlib with high-level visualizations such as heatmaps, Cluster maps, Pair plots etc. It also has in-built colour palettes and themes.

5.CODE SNIPPETS AND VISUALIZATIONS

5.1. Code snippets:



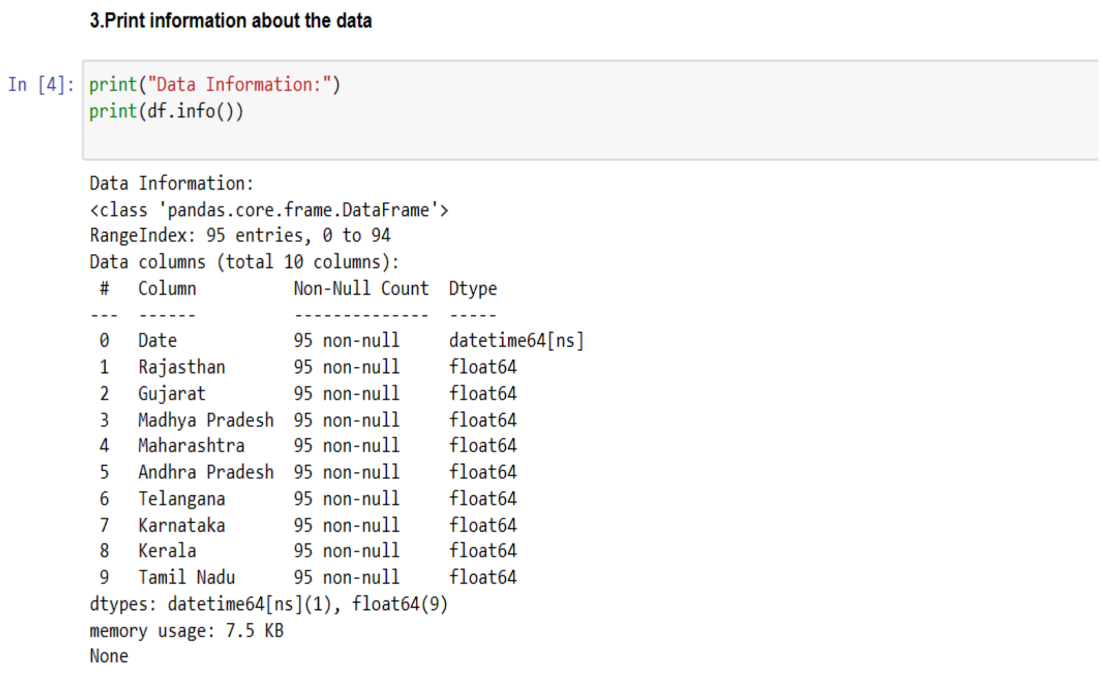
Importing data and pre processing

1.Importing Necessary Libraries

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from pmdarima import auto_arima
        from statsmodels.tsa.arima.model import ARIMA
        from statsmodels.tsa.stattools import adfuller
        from statsmodels.tsa.seasonal import seasonal_decompose
        from sklearn.metrics import mean_absolute_error, mean_squared_error
        from statsmodels.tsa.statespace.sarimax import SARIMAX
```

2.Read the data from the Excel File

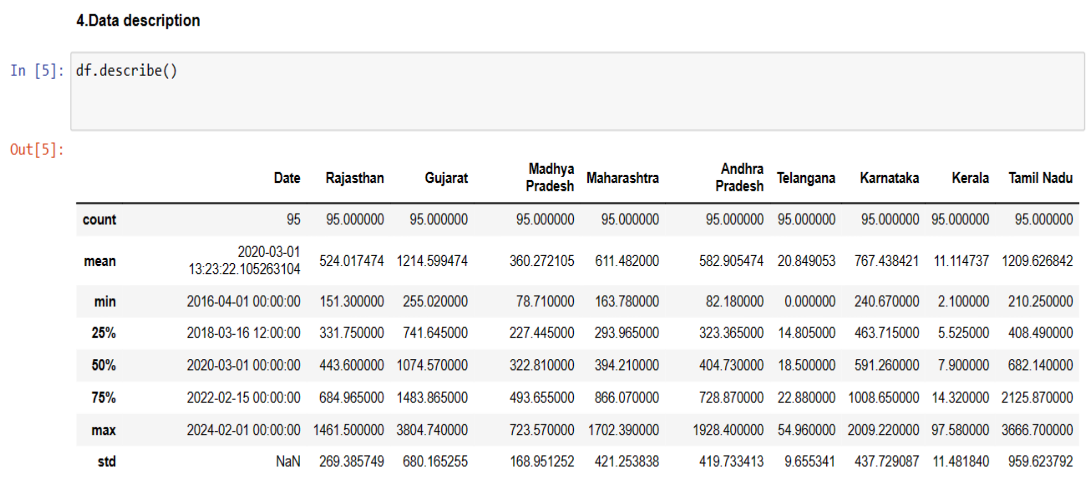
```
In [3]: df = pd.read_excel("C://Users//srivi//OneDrive//Desktop//BS//project//data reneweable energy//transpose data of wind.xlsx")
```



3.Print information about the data

```
In [4]: print("Data Information:")
        print(df.info())

Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95 entries, 0 to 94
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            95 non-null     datetime64[ns]
 1   Rajasthan       95 non-null     float64
 2   Gujarat         95 non-null     float64
 3   Madhya Pradesh  95 non-null     float64
 4   Maharashtra     95 non-null     float64
 5   Andhra Pradesh  95 non-null     float64
 6   Telangana       95 non-null     float64
 7   Karnataka       95 non-null     float64
 8   Kerala          95 non-null     float64
 9   Tamil Nadu      95 non-null     float64
dtypes: datetime64[ns](1), float64(9)
memory usage: 7.5 KB
None
```

These steps involve importing the data from the excel file and getting the information about the data. Python has the feature of reading data from different file formats like Excel, CSV, SQL etc.

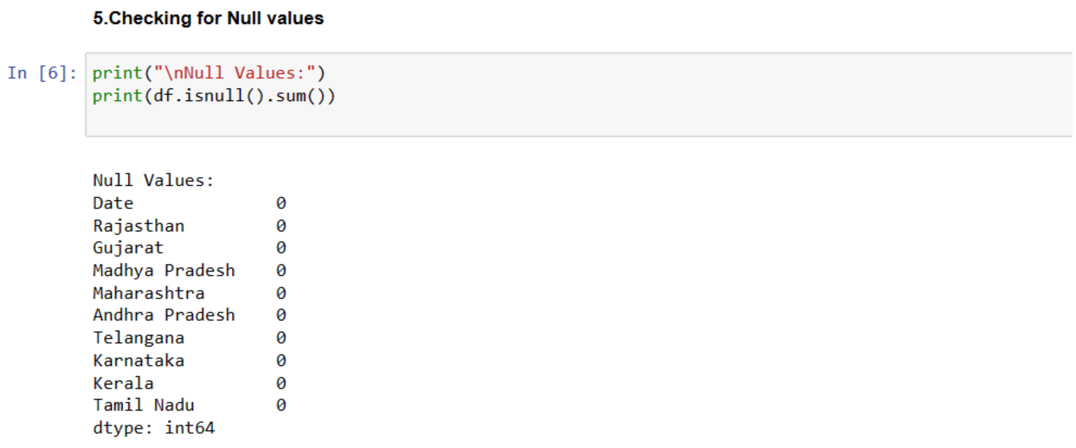


4.Data description

```
In [5]: df.describe()
```

Out[5]:

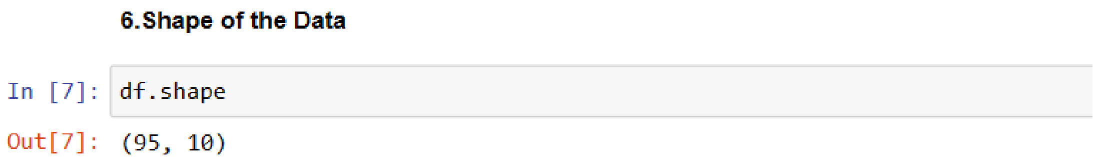
	Date	Rajasthan	Gujarat	Madhya Pradesh	Maharashtra	Andhra Pradesh	Telangana	Karnataka	Kerala	Tamil Nadu
count	95	95.000000	95.000000	95.000000	95.000000	95.000000	95.000000	95.000000	95.000000	95.000000
mean	2020-03-01 13:23:22.105263104	524.017474	1214.599474	360.272105	611.482000	582.905474	20.849053	767.438421	11.114737	1209.626842
min	2016-04-01 00:00:00	151.300000	255.020000	78.710000	163.780000	82.180000	0.000000	240.670000	2.100000	210.250000
25%	2018-03-16 12:00:00	331.750000	741.645000	227.445000	293.965000	323.365000	14.805000	463.715000	5.525000	408.490000
50%	2020-03-01 00:00:00	443.600000	1074.570000	322.810000	394.210000	404.730000	18.500000	591.260000	7.900000	682.140000
75%	2022-02-15 00:00:00	684.965000	1483.865000	493.655000	866.070000	728.870000	22.880000	1008.650000	14.320000	2125.870000
max	2024-02-01 00:00:00	1461.500000	3804.740000	723.570000	1702.390000	1928.400000	54.960000	2009.220000	97.580000	3666.700000
std	NaN	269.385749	680.165255	168.951252	421.253838	419.733413	9.655341	437.729087	11.481840	959.623792



5.Checking for Null values

```
In [6]: print("\nNull Values:")
        print(df.isnull().sum())
```

```
Null Values:
Date              0
Rajasthan         0
Gujarat           0
Madhya Pradesh    0
Maharashtra       0
Andhra Pradesh    0
Telangana         0
Karnataka         0
Kerala            0
Tamil Nadu        0
dtype: int64
```



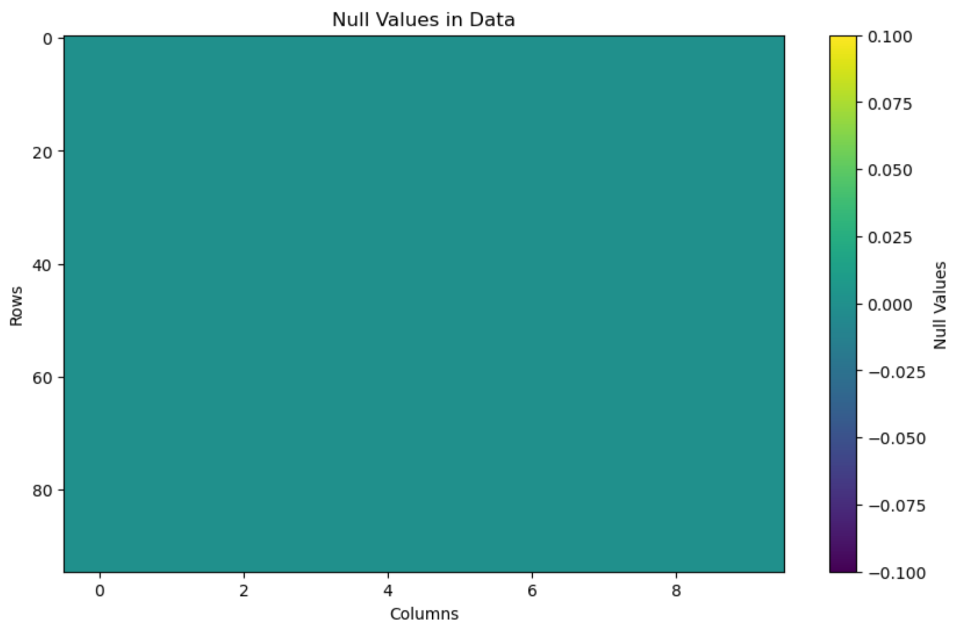
6.Shape of the Data

```
In [7]: df.shape
Out[7]: (95, 10)
```

These steps involve checking for the null values in the data, shape of the data and describing the statistical measures of the data.

7.Visualize null values

```
In [8]: plt.figure(figsize=(10, 6))
        plt.title('Null Values in Data')
        plt.imshow(df.isnull(), cmap='viridis', aspect='auto')
        plt.xlabel('Columns')
        plt.ylabel('Rows')
        plt.colorbar(label='Null Values')
        plt.show()
```



8.Setting "Date" column as index

```
In [9]: df.set_index('Date', inplace=True)
```

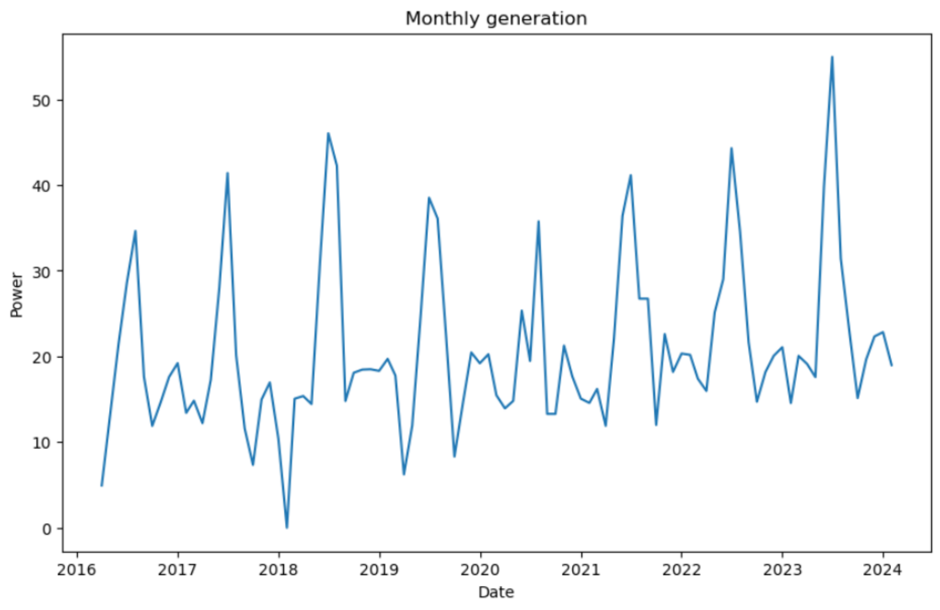
9.Select the target variable ¶

```
In [159]: y = df['Telangana']
```

These steps involve setting the date column of the data as index and selecting a state as target variable. We can select any of the states or variables as target variable.

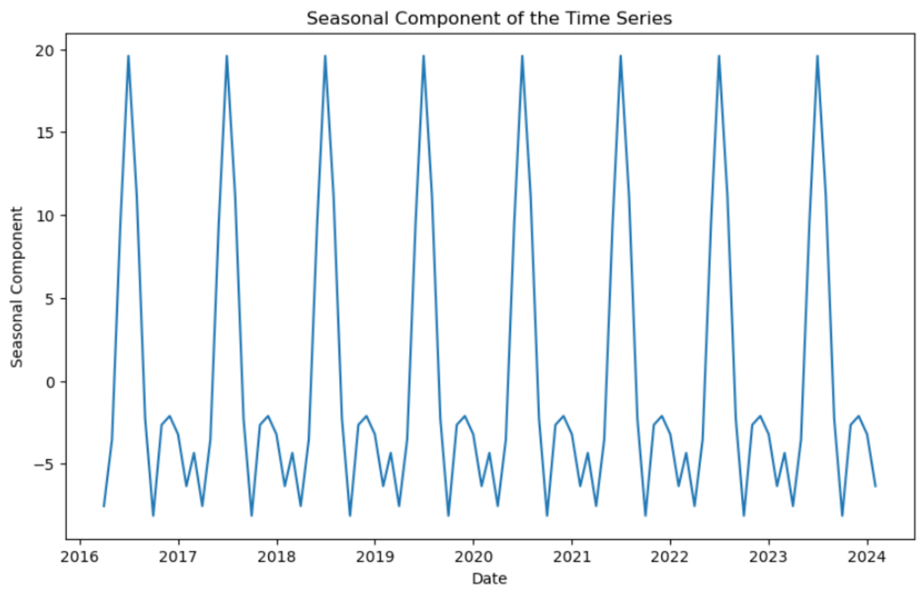
10.Plot the time series data

```
In [160]: plt.figure(figsize=(10, 6))
          plt.plot(y)
          plt.title('Monthly generation')
          plt.xlabel('Date')
          plt.ylabel('Power')
          plt.show()
```



11.Decomposition of the time series data and plotting

```
In [161]: decomposition = seasonal_decompose(y)
          seasonal_component = decomposition.seasonal
          plt.figure(figsize=(10, 6))
          plt.plot(seasonal_component)
          plt.title('Seasonal Component of the Time Series')
          plt.xlabel('Date')
          plt.ylabel('Seasonal Component')
          plt.show()
```



12.Stationarity Check

In [162]:
```
def check_stationarity(y):
    # Perform Dickey-Fuller test
    result = adfuller(y)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

    if result[1] > 0.05:
        print('The series is likely non-stationary.')
    else:
        print('The series is likely stationary.')

# Check stationarity
print('Original Data:')
check_stationarity(y)
```

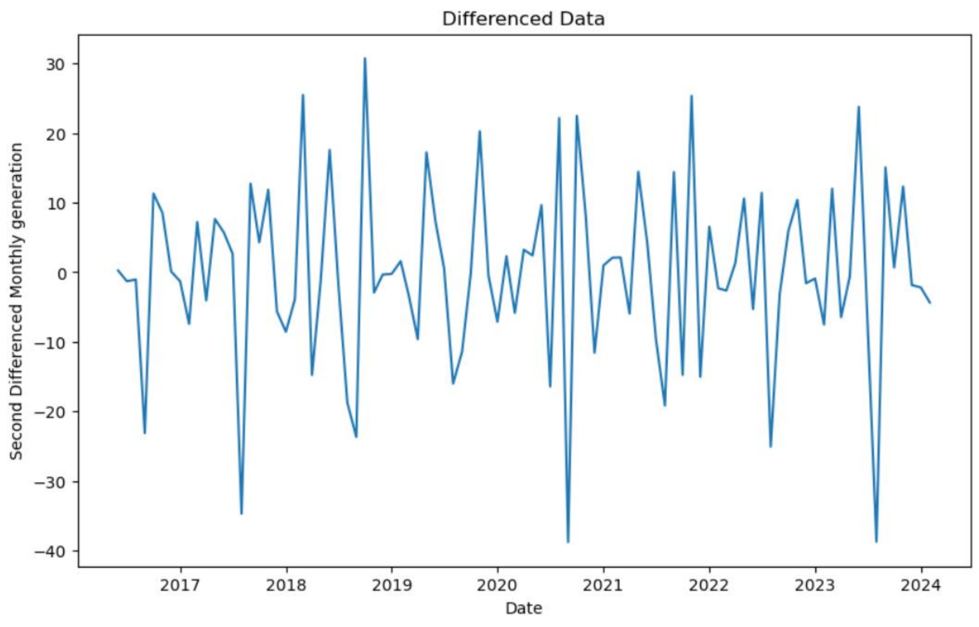
```
Original Data:
ADF Statistic: -1.1044422607890627
p-value: 0.7132971354394794
Critical Values:
	1%: -3.512
	5%: -2.897
	10%: -2.586
The series is likely non-stationary.
```

13.Differencing the data

In [163]:
```
# First Differencing
y_diff = y.diff().dropna()

# Second Differencing
y_diff = y_diff.diff().dropna()
```

14.Plotting the differenced data

In [164]:
```
plt.figure(figsize=(10, 6))
plt.plot(y_diff)
plt.title('Differenced Data')
plt.xlabel('Date')
plt.ylabel('Second Differenced Monthly generation')
plt.show()
```



15.Checking Stationarity of differenced data

In [165]:
```
print('Differenced Data:')
check_stationarity(y_diff)
```

```
Differenced Data:
ADF Statistic: -7.671457418206155
p-value: 1.5895632640255983e-11
Critical Values:
        1%: -3.515
        5%: -2.898
        10%: -2.586
The series is likely stationary.
```

Autoregressive Integrated Moving Average (ARIMA)

16.Building ARIMA model

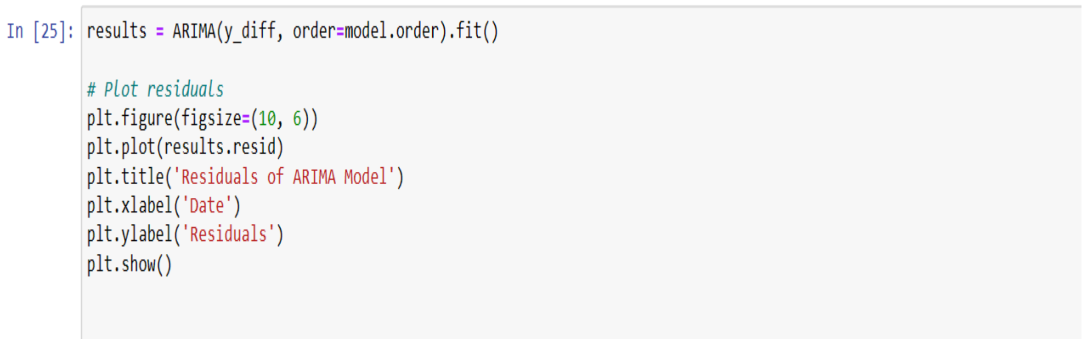
In [166]:
```
# Fit ARIMA model using auto_arima
model = auto_arima(y_diff, seasonal=False, trace=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=inf, Time=0.10 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=744.432, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=734.140, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=inf, Time=0.04 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=729.097, Time=0.03 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=715.497, Time=0.04 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=711.743, Time=0.06 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=703.630, Time=0.06 sec
 ARIMA(5,0,1)(0,0,0)[0]             : AIC=inf, Time=0.25 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=inf, Time=0.17 sec
 ARIMA(5,0,0)(0,0,0)[0] intercept   : AIC=705.590, Time=0.13 sec

Best model:  ARIMA(5,0,0)(0,0,0)[0]
Total fit time: 0.907 seconds
```

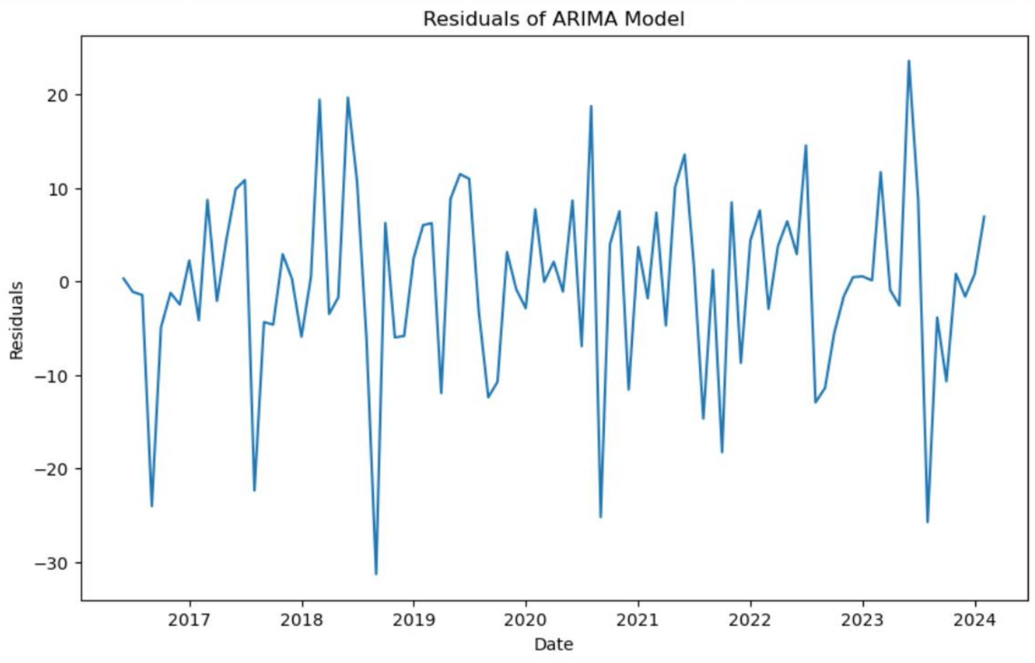
In [167]:
```
# Summary of the model
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   93
Model:               SARIMAX(5, 0, 0)   Log Likelihood                -345.815
Date:                Mon, 22 Apr 2024   AIC                            703.630
Time:                        13:54:30   BIC                            718.826
Sample:                    06-01-2016   HQIC                           709.766
                         - 02-01-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7384      0.104     -7.077      0.000      -0.943      -0.534
ar.L2         -0.7378      0.121     -6.106      0.000      -0.975      -0.501
ar.L3         -0.7159      0.142     -5.026      0.000      -0.995      -0.437
ar.L4         -0.4676      0.167     -2.795      0.005      -0.795      -0.140
ar.L5         -0.3233      0.124     -2.611      0.009      -0.566      -0.081
sigma2        97.6820     12.792      7.636      0.000      72.610     122.754
===================================================================================
Ljung-Box (L1) (Q):                   0.57   Jarque-Bera (JB):                 8.35
Prob(Q):                              0.45   Prob(JB):                         0.02
Heteroskedasticity (H):               0.87   Skew:                            -0.55
Prob(H) (two-sided):                  0.70   Kurtosis:                         3.97
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

17.Fitting the ARIMA model



```
In [25]: results = ARIMA(y_diff, order=model.order).fit()

         # Plot residuals
         plt.figure(figsize=(10, 6))
         plt.plot(results.resid)
         plt.title('Residuals of ARIMA Model')
         plt.xlabel('Date')
         plt.ylabel('Residuals')
         plt.show()
```

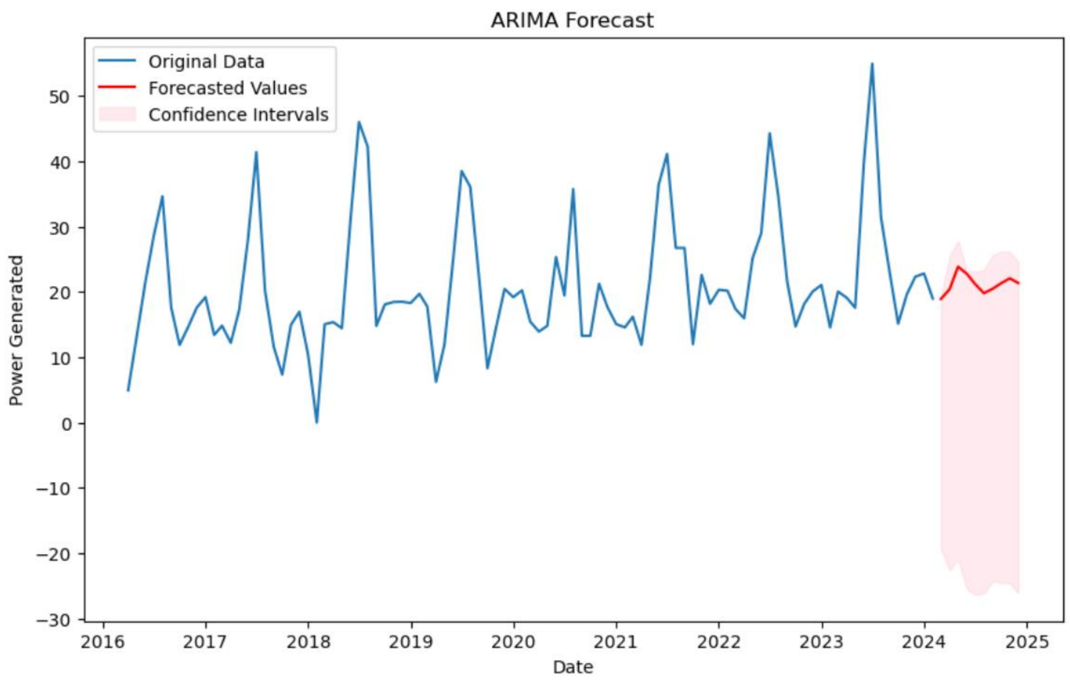


After checking the stationarity of the data, through auto ARIMA function the order of the model will be decided. In the summary of the model, we can see various values like log likelihood and information criterion values like AIC, BIC etc. The lesser these values are, the better the model is in Time series.

18.Forecasting the results

```
In [26]:
forecast_start = pd.Timestamp('2023-04-01')
forecast_end = pd.Timestamp('2024-02-01')
forecast_period = (forecast_end - forecast_start).days // 30  # Convert to number of months
forecast = results.get_forecast(steps=forecast_period)

# Get the forecasted values and confidence intervals
forecast_values_diff = forecast.predicted_mean
confidence_intervals_diff = forecast.conf_int()

# Cumulative sum of forecasted differences
forecast_cumsum = np.cumsum(forecast_values_diff)

# Add the cumulative sum to the last observed value to get positive forecasted values
last_observed_value = y.iloc[-1]
forecast_positive = forecast_cumsum + last_observed_value
```

```
In [27]:
# Plot the original data and forecasted values
plt.figure(figsize=(10, 6))
plt.plot(y, label='Original Data')
plt.plot(forecast_positive, color='red', label='Forecasted Values')
plt.fill_between(confidence_intervals_diff.index, confidence_intervals_diff.iloc[:, 0], confidence_intervals_diff.iloc[:, 1], col
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Power Generated')
plt.legend()
plt.show()
```

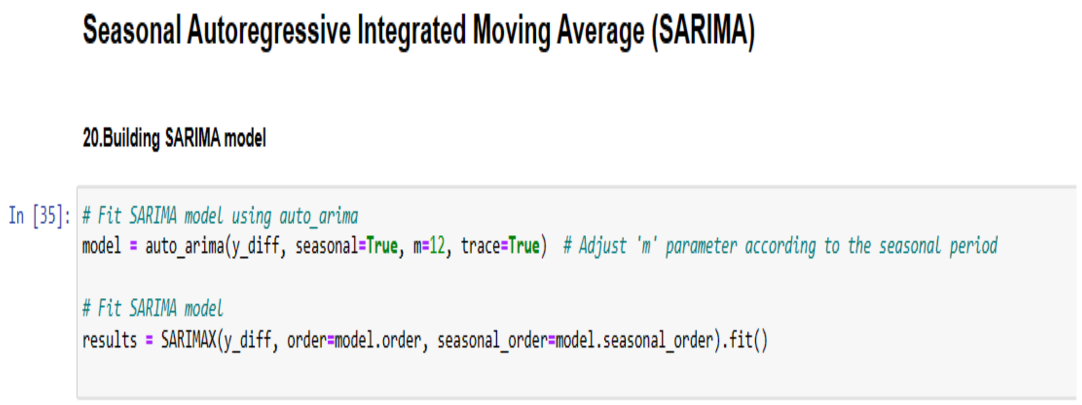


```
In [171]: # Print the forecasted values along with the dates
          forecast_dates = pd.date_range(start=forecast_start, periods=len(forecast_positive), freq='MS')
          forecast_with_dates = pd.Series(forecast_positive.values, index=forecast_dates)
          print("Forecasted Values (Positive):")
          print(forecast_with_dates)

          Forecasted Values (Positive):
          2023-04-01    18.930652
          2023-05-01    20.404659
          2023-06-01    23.870990
          2023-07-01    22.789182
          2023-08-01    21.195242
          2023-09-01    19.808433
          2023-10-01    20.475682
          2023-11-01    21.322137
          2023-12-01    22.082090
          2024-01-01    21.371706
          Freq: MS, dtype: float64
```

19. Calculation of the Metrics

```
In [29]: # Calculate RMSE for the forecasted values
         actual_values = y.loc['2023-04-01':'2024-01-01']
         rmse = np.sqrt(mean_squared_error(actual_values, forecast_positive))
         print("RMSE:", rmse)

         # Calculate R-squared and adjusted R-squared scores
         def calculate_r_squared(y_true, y_pred, p):
             # Calculate R-squared
             SSR = np.sum((y_pred - y_true.mean())**2)
             SST = np.sum((y_true - y_true.mean())**2)
             R_squared = SSR / SST

             # Calculate adjusted R-squared
             n = len(y_true)
             adj_R_squared = 1 - ((1 - R_squared) * (n - 1) / (n - p - 1))

             return R_squared, adj_R_squared

         # Get fitted values
         fitted_values = results.fittedvalues

         # Calculate R-squared and adjusted R-squared
         R_squared, adj_R_squared = calculate_r_squared(y_diff, fitted_values, len(model.order))

         # Print scores
         print("R-squared:", R_squared)
         print("Adjusted R-squared:", adj_R_squared)

         from sklearn.metrics import mean_absolute_error, mean_squared_error

         # Calculate MAE
         mae = mean_absolute_error(y.iloc[-11:-1], forecast_positive)
         print("MAE:", mae)

         # Calculate MAPE
         def calculate_mape(y_true, y_pred):
             return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

         mape = calculate_mape(y.iloc[-11:-1], forecast_positive)
         print("MAPE:", mape)
```

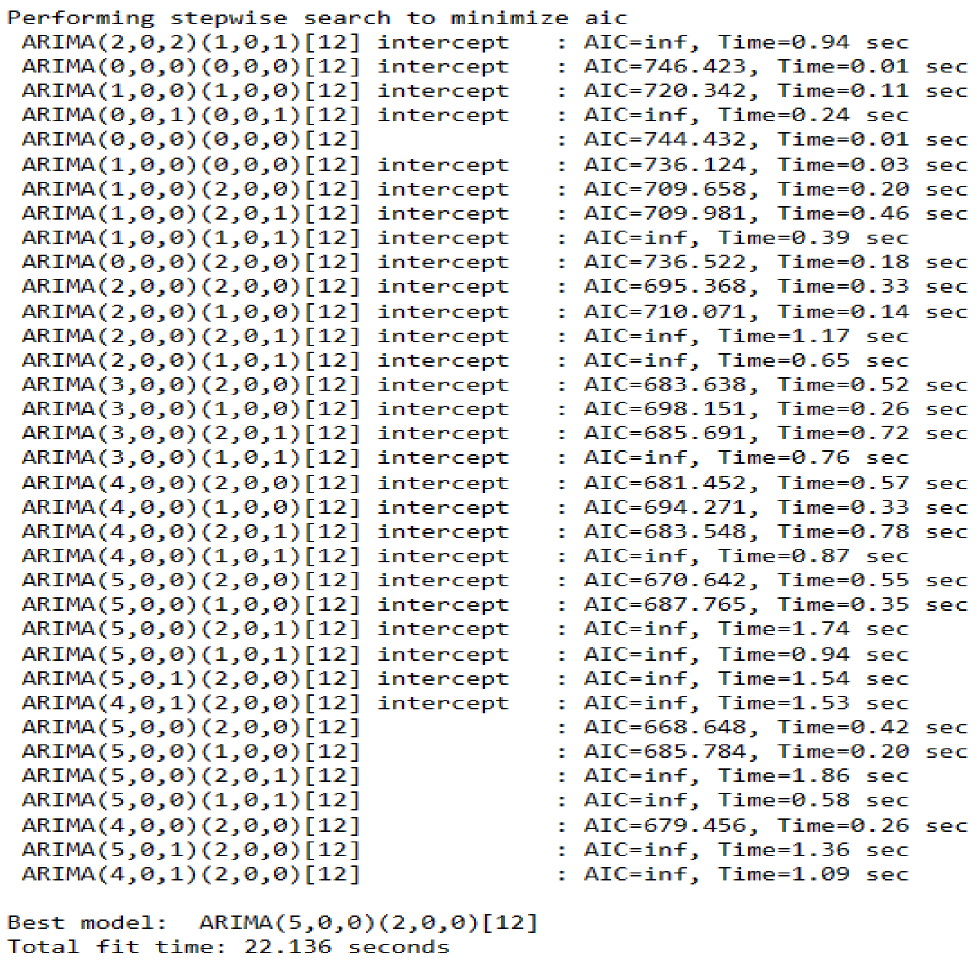
```
RMSE: 12.01944405338248
R-squared: 0.42172105121565673
Adjusted R-squared: 0.40222850238022945
MAE: 7.317418375427588
MAPE: nan
```



Seasonal Autoregressive Integrated Moving Average (SARIMA)

20.Building SARIMA model

```
In [35]: # Fit SARIMA model using auto_arima
         model = auto_arima(y_diff, seasonal=True, m=12, trace=True)  # Adjust 'm' parameter according to the seasonal period

         # Fit SARIMA model
         results = SARIMAX(y_diff, order=model.order, seasonal_order=model.seasonal_order).fit()
```



```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=0.94 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=746.423, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=720.342, Time=0.11 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=inf, Time=0.24 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=744.432, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=736.124, Time=0.03 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=709.658, Time=0.20 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=709.981, Time=0.46 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.39 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=736.522, Time=0.18 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=695.368, Time=0.33 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=710.071, Time=0.14 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=inf, Time=1.17 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.65 sec
 ARIMA(3,0,0)(2,0,0)[12] intercept   : AIC=683.638, Time=0.52 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=698.151, Time=0.26 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=685.691, Time=0.72 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.76 sec
 ARIMA(4,0,0)(2,0,0)[12] intercept   : AIC=681.452, Time=0.57 sec
 ARIMA(4,0,0)(1,0,0)[12] intercept   : AIC=694.271, Time=0.33 sec
 ARIMA(4,0,0)(2,0,1)[12] intercept   : AIC=683.548, Time=0.78 sec
 ARIMA(4,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.87 sec
 ARIMA(5,0,0)(2,0,0)[12] intercept   : AIC=670.642, Time=0.55 sec
 ARIMA(5,0,0)(1,0,0)[12] intercept   : AIC=687.765, Time=0.35 sec
 ARIMA(5,0,0)(2,0,1)[12] intercept   : AIC=inf, Time=1.74 sec
 ARIMA(5,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.94 sec
 ARIMA(5,0,1)(2,0,0)[12] intercept   : AIC=inf, Time=1.54 sec
 ARIMA(4,0,1)(2,0,0)[12] intercept   : AIC=inf, Time=1.53 sec
 ARIMA(5,0,0)(2,0,0)[12]             : AIC=668.648, Time=0.42 sec
 ARIMA(5,0,0)(1,0,0)[12]             : AIC=685.784, Time=0.20 sec
 ARIMA(5,0,0)(2,0,1)[12]             : AIC=inf, Time=1.86 sec
 ARIMA(5,0,0)(1,0,1)[12]             : AIC=inf, Time=0.58 sec
 ARIMA(4,0,0)(2,0,0)[12]             : AIC=679.456, Time=0.26 sec
 ARIMA(5,0,1)(2,0,0)[12]             : AIC=inf, Time=1.36 sec
 ARIMA(4,0,1)(2,0,0)[12]             : AIC=inf, Time=1.09 sec

Best model:  ARIMA(5,0,0)(2,0,0)[12]
Total fit time: 22.136 seconds
```

In this step we have fitted a SARIMA model by taking the seasonality into account and gave the value of seasonal period as 12 months.

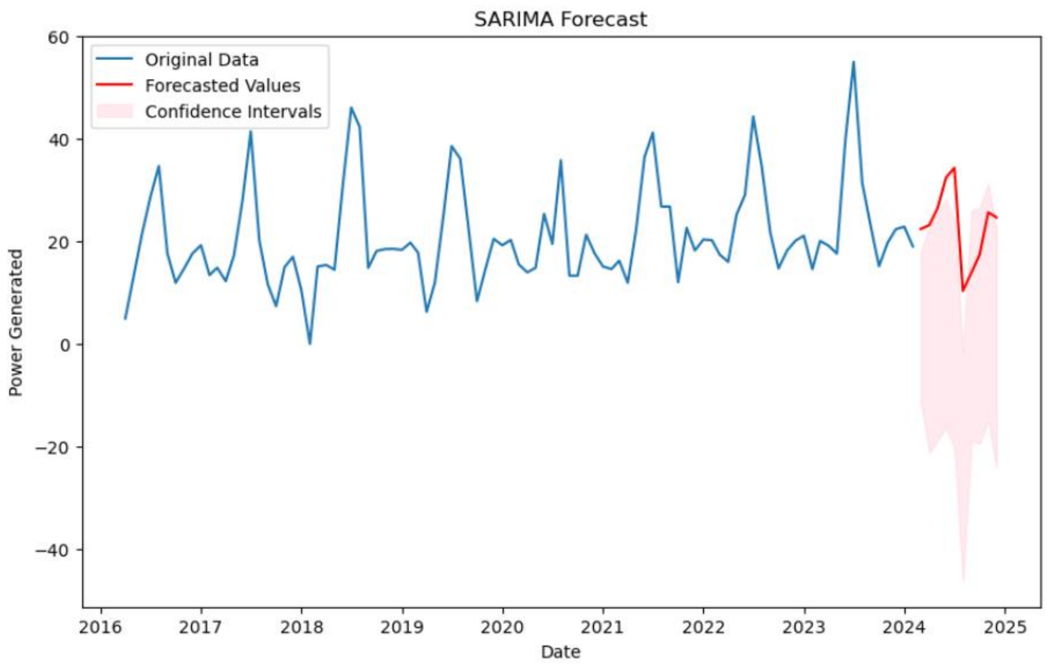
```
In [174]: # Summary of the model
          print(model.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                   93
Model:             SARIMAX(5, 0, 0)x(2, 0, 0, 12)   Log Likelihood                -326.324
Date:                            Mon, 22 Apr 2024   AIC                            668.648
Time:                                    13:55:19   BIC                            688.909
Sample:                                06-01-2016   HQIC                           676.829
                                     - 02-01-2024
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.1168      0.101    -11.019      0.000      -1.315      -0.918
ar.L2         -1.0769      0.170     -6.334      0.000      -1.410      -0.744
ar.L3         -0.8843      0.192     -4.595      0.000      -1.261      -0.507
ar.L4         -0.5878      0.176     -3.346      0.001      -0.932      -0.243
ar.L5         -0.3595      0.122     -2.940      0.003      -0.599      -0.120
ar.S.L12       0.3099      0.087      3.554      0.000       0.139       0.481
ar.S.L24       0.4947      0.093      5.346      0.000       0.313       0.676
sigma2        55.8445      9.085      6.147      0.000      38.038      73.651
===================================================================================
Ljung-Box (L1) (Q):                   0.65   Jarque-Bera (JB):                 6.87
Prob(Q):                              0.42   Prob(JB):                         0.03
Heteroskedasticity (H):               0.97   Skew:                            -0.62
Prob(H) (two-sided):                  0.94   Kurtosis:                         3.51
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

21.Forecasting the results

```
In [37]: forecast_start = pd.Timestamp('2023-04-01')
         forecast_end = pd.Timestamp('2024-02-01')
         forecast_period = (forecast_end - forecast_start).days // 30  # Convert to number of months
         forecast = results.get_forecast(steps=forecast_period)

         # Get the forecasted values and confidence intervals
         forecast_values_diff = forecast.predicted_mean
         confidence_intervals_diff = forecast.conf_int()

         # Cumulative sum of forecasted differences
         forecast_cumsum = np.cumsum(forecast_values_diff)

         # Add the cumulative sum to the last observed value to get positive forecasted values
         last_observed_value = y.iloc[-1]
         forecast_positive = forecast_cumsum + last_observed_value
```

```
In [38]: # Plot the original data and forecasted values
         plt.figure(figsize=(10, 6))
         plt.plot(y, label='Original Data')
         plt.plot(forecast_positive, color='red', label='Forecasted Values')
         plt.fill_between(confidence_intervals_diff.index, confidence_intervals_diff.iloc[:, 0], confidence_intervals_diff.iloc[:, 1], col
         plt.title('SARIMA Forecast')
         plt.xlabel('Date')
         plt.ylabel('Power Generated')
         plt.legend()
         plt.show()
```

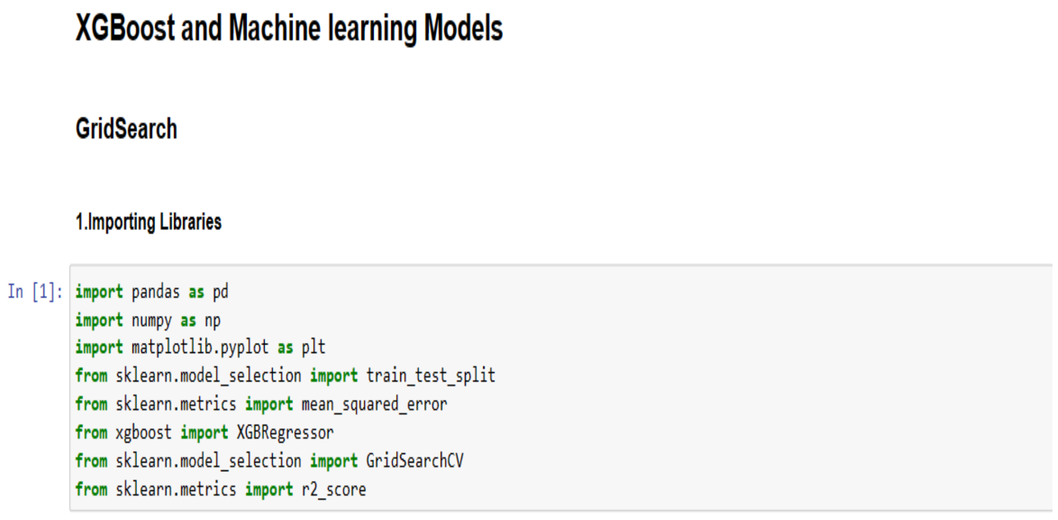


```
In [177]: # Print the forecasted values along with the dates
          forecast_dates = pd.date_range(start=forecast_start, periods=len(forecast_positive), freq='MS')
          forecast_with_dates = pd.Series(forecast_positive.values, index=forecast_dates)
          print("Forecasted Values (Positive):")
          print(forecast_with_dates)
```

```
Forecasted Values (Positive):
2023-04-01    22.381396
2023-05-01    23.129350
2023-06-01    26.438964
2023-07-01    32.414045
2023-08-01    34.288955
2023-09-01    10.321114
2023-10-01    13.872131
2023-11-01    17.400271
2023-12-01    25.605203
2024-01-01    24.644040
Freq: MS, dtype: float64
```

Metrics of the SARIMA model

```
RMSE: 9.573586484099591
R-squared: 0.7506854366715472
Adjusted R-squared: 0.7422815749863184
MAE: 6.882241862106701
MAPE: nan
```

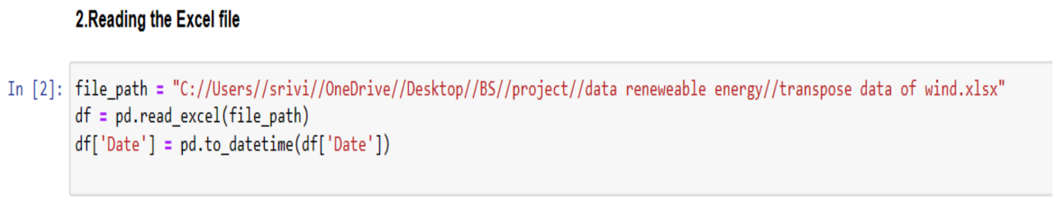


XGBoost and Machine learning Models

GridSearch

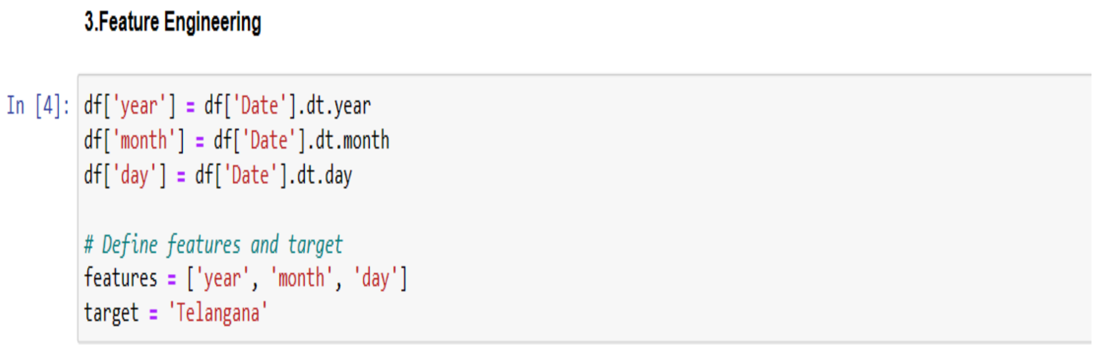
1.Importing Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error
        from xgboost import XGBRegressor
        from sklearn.model_selection import GridSearchCV
        from sklearn.metrics import r2_score
```



2.Reading the Excel file

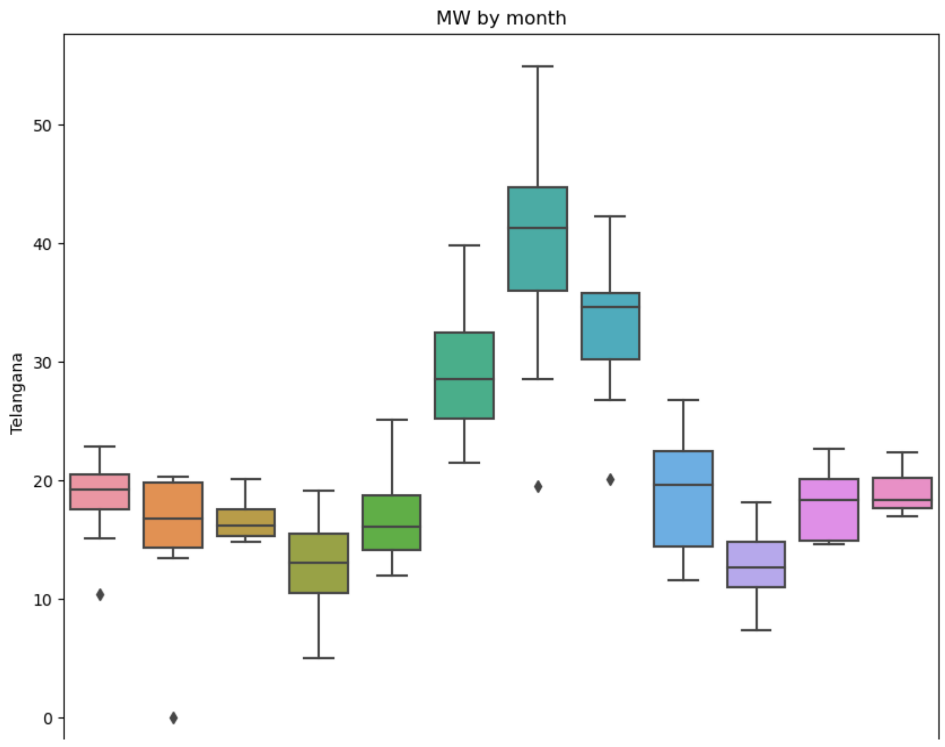
```
In [2]: file_path = "C://Users//srivi//OneDrive//Desktop//BS//project//data reneweable energy//transpose data of wind.xlsx"
        df = pd.read_excel(file_path)
        df['Date'] = pd.to_datetime(df['Date'])
```



3.Feature Engineering

```
In [4]: df['year'] = df['Date'].dt.year
        df['month'] = df['Date'].dt.month
        df['day'] = df['Date'].dt.day

        # Define features and target
        features = ['year', 'month', 'day']
        target = 'Telangana'
```

```
In [20]: import matplotlib.pyplot as plt
         import seaborn as sns
         fig, ax = plt.subplots(figsize=(10, 8))
         sns.boxplot(data=df, x='month', y='Telangana')
         ax.set_title('MW by month')
         plt.show()
```



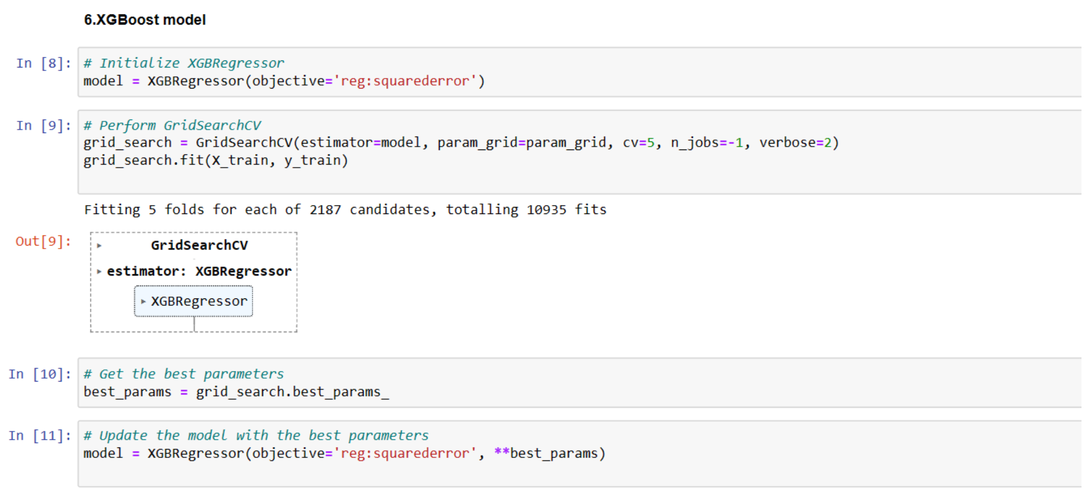
4.Data Split

```
In [6]: X = df[features]
        y = df[target]

        # Split data into training and testing sets (80:20 ratio)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5.HyperParameter Tuning

```
In [7]: # Define the parameter grid for GridSearchCV
        param_grid = {
            'n_estimators': [100, 500, 1000],
            'max_depth': [3, 5, 7],
            'learning_rate': [0.05, 0.1, 0.2],
            'subsample': [0.6, 0.8, 1.0],
            'colsample_bytree': [0.6, 0.8, 1.0],
            'reg_alpha': [0.05, 0.1, 0.5],
            'reg_lambda': [0.05, 0.1, 0.5]
        }
```

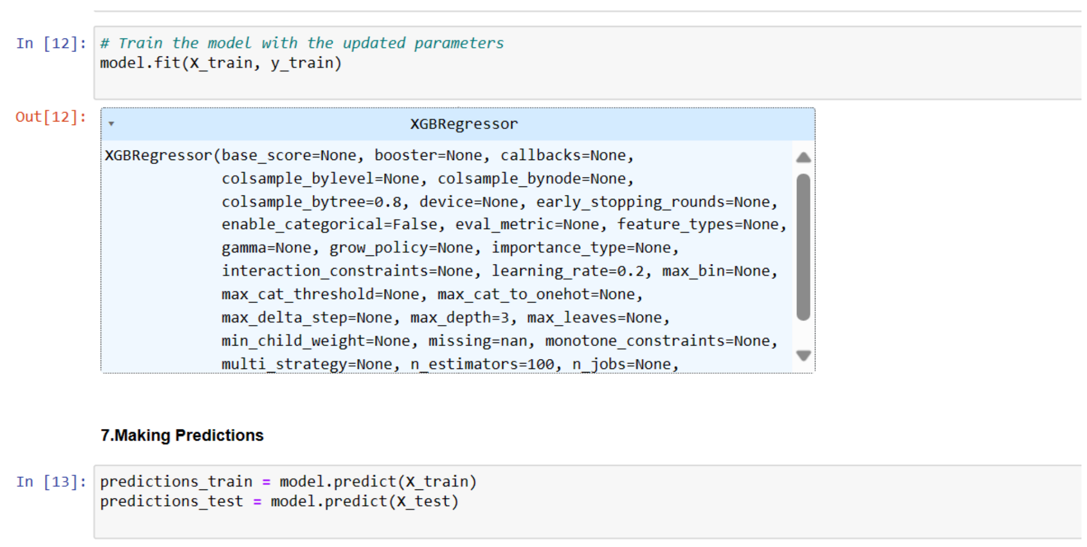
GridSearch:



6.XGBoost model

```
In [8]: # Initialize XGBRegressor
        model = XGBRegressor(objective='reg:squarederror')

In [9]: # Perform GridSearchCV
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
        grid_search.fit(X_train, y_train)

        Fitting 5 folds for each of 2187 candidates, totalling 10935 fits

Out[9]: ▸ GridSearchCV
        ▸ estimator: XGBRegressor
            ▸ XGBRegressor

In [10]: # Get the best parameters
         best_params = grid_search.best_params_

In [11]: # Update the model with the best parameters
         model = XGBRegressor(objective='reg:squarederror', **best_params)
```

The hyperparameter tuning method GridSearch has been used here where we define a grid with parameter values and the function searches for the best set of parameters through k-fold cross validation technique.

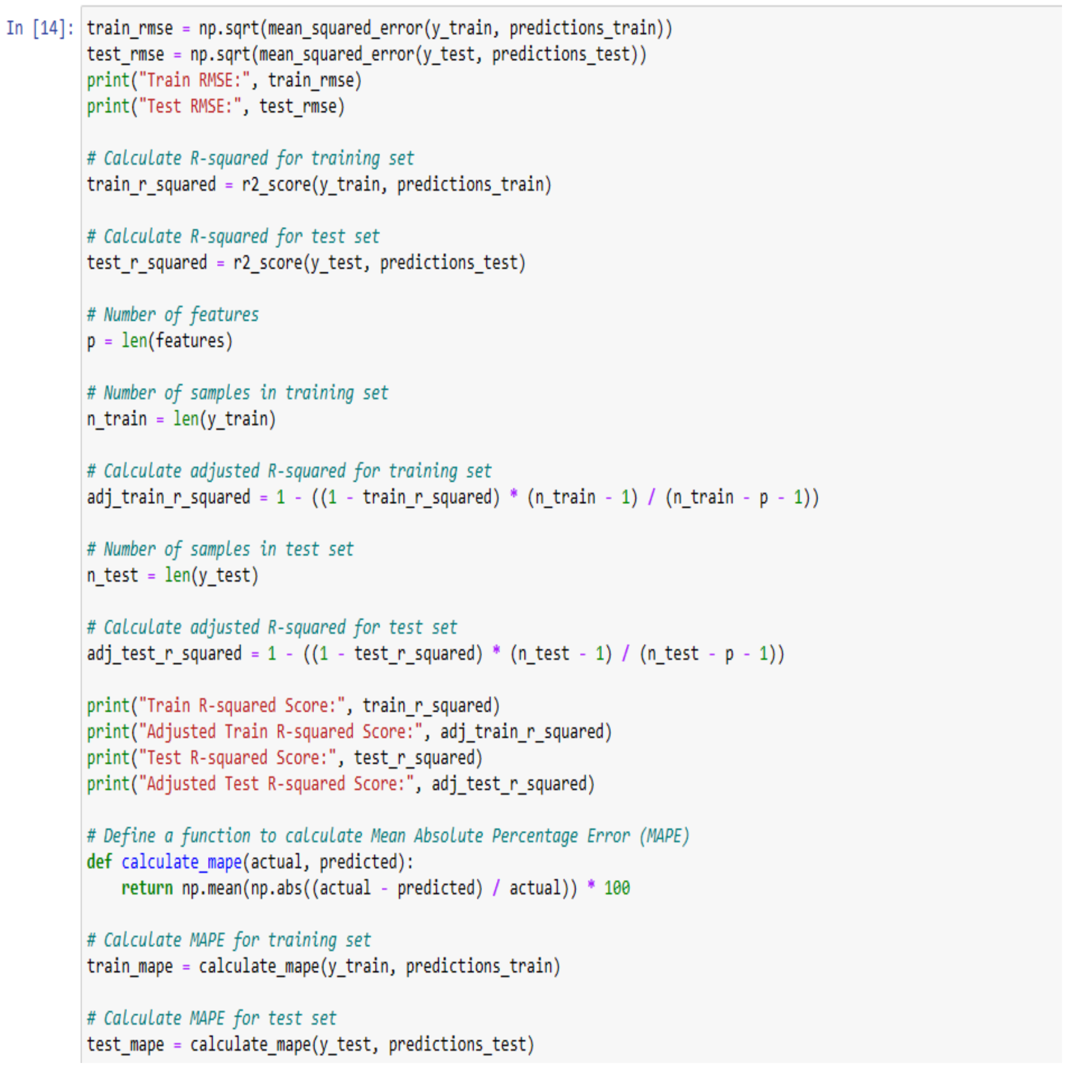


```
In [12]: # Train the model with the updated parameters
         model.fit(X_train, y_train)

Out[12]: ▾ XGBRegressor
         XGBRegressor(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=0.8, device=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.2, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=3, max_leaves=None,
                      min_child_weight=None, missing=nan, monotone_constraints=None,
                      multi_strategy=None, n_estimators=100, n_jobs=None,
```

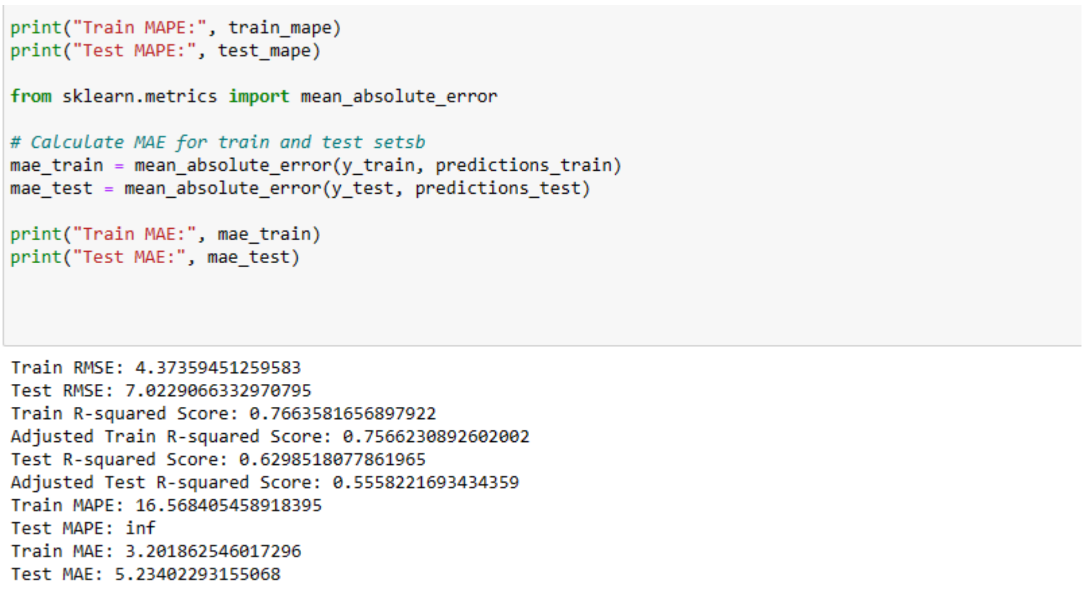
7.Making Predictions

```
In [13]: predictions_train = model.predict(X_train)
         predictions_test = model.predict(X_test)
```

8.Evaluating the Model



```
In [14]: train_rmse = np.sqrt(mean_squared_error(y_train, predictions_train))
         test_rmse = np.sqrt(mean_squared_error(y_test, predictions_test))
         print("Train RMSE:", train_rmse)
         print("Test RMSE:", test_rmse)

         # Calculate R-squared for training set
         train_r_squared = r2_score(y_train, predictions_train)

         # Calculate R-squared for test set
         test_r_squared = r2_score(y_test, predictions_test)

         # Number of features
         p = len(features)

         # Number of samples in training set
         n_train = len(y_train)

         # Calculate adjusted R-squared for training set
         adj_train_r_squared = 1 - ((1 - train_r_squared) * (n_train - 1) / (n_train - p - 1))

         # Number of samples in test set
         n_test = len(y_test)

         # Calculate adjusted R-squared for test set
         adj_test_r_squared = 1 - ((1 - test_r_squared) * (n_test - 1) / (n_test - p - 1))

         print("Train R-squared Score:", train_r_squared)
         print("Adjusted Train R-squared Score:", adj_train_r_squared)
         print("Test R-squared Score:", test_r_squared)
         print("Adjusted Test R-squared Score:", adj_test_r_squared)

         # Define a function to calculate Mean Absolute Percentage Error (MAPE)
         def calculate_mape(actual, predicted):
             return np.mean(np.abs((actual - predicted) / actual)) * 100

         # Calculate MAPE for training set
         train_mape = calculate_mape(y_train, predictions_train)

         # Calculate MAPE for test set
         test_mape = calculate_mape(y_test, predictions_test)
```

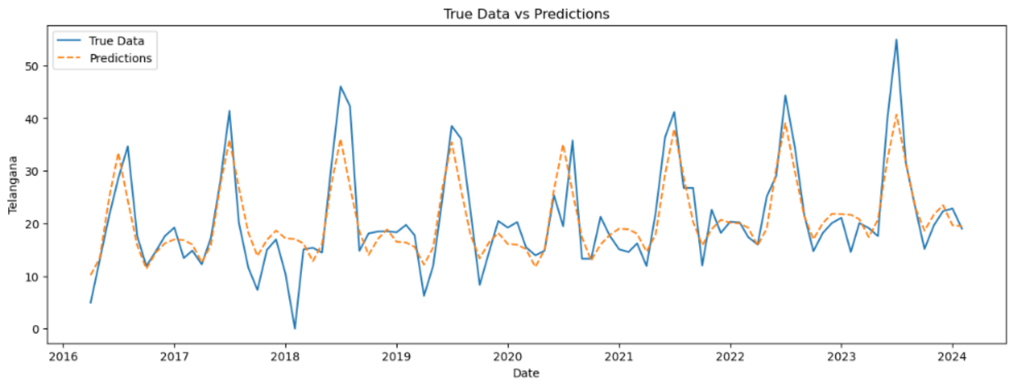


```
print("Train MAPE:", train_mape)
print("Test MAPE:", test_mape)

from sklearn.metrics import mean_absolute_error

# Calculate MAE for train and test setsb
mae_train = mean_absolute_error(y_train, predictions_train)
mae_test = mean_absolute_error(y_test, predictions_test)

print("Train MAE:", mae_train)
print("Test MAE:", mae_test)
```

```
Train RMSE: 4.37359451259583
Test RMSE: 7.0229066332970795
Train R-squared Score: 0.7663581656897922
Adjusted Train R-squared Score: 0.7566230892602002
Test R-squared Score: 0.6298518077861965
Adjusted Test R-squared Score: 0.5558221693434359
Train MAPE: 16.568405458918395
Test MAPE: inf
Train MAE: 3.201862546017296
Test MAE: 5.23402293155068
```

9.Plotting the Predictions

```
In [15]: plt.figure(figsize=(15, 5))
         plt.plot(df['Date'], df[target], label='True Data')
         plt.plot(df['Date'], model.predict(df[features]), label='Predictions', linestyle='--')
         plt.xlabel('Date')
         plt.ylabel(target)
         plt.title('True Data vs Predictions')
         plt.legend()
         plt.show()
```

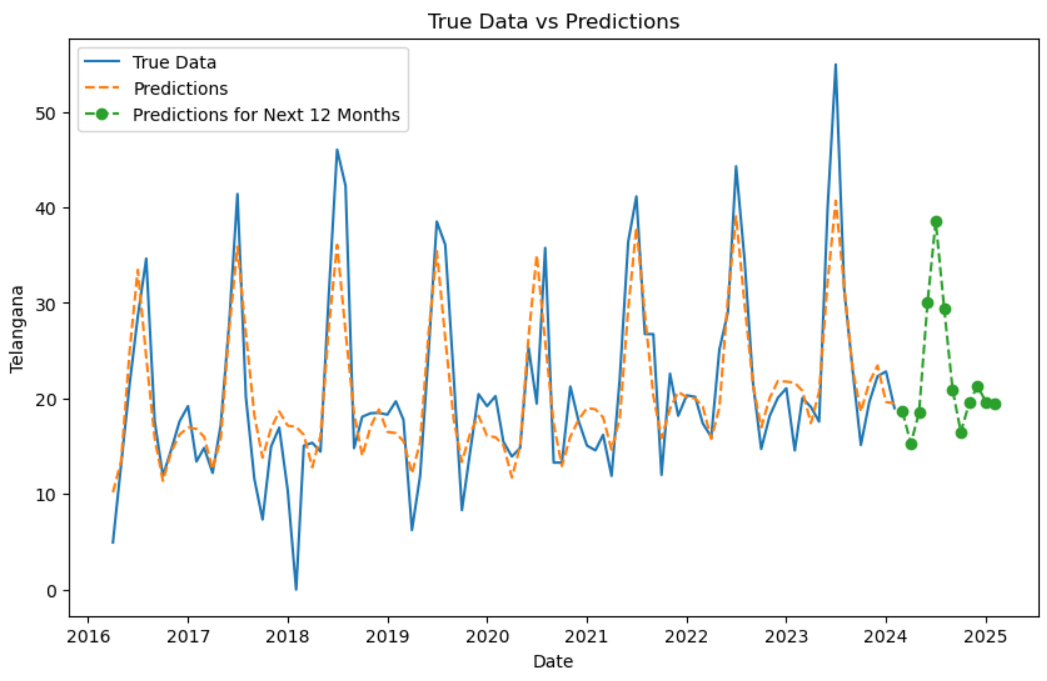


```
In [16]: # Select the last 12 months from the dataset
         existing_dates = df['Date'].tail(12)

         # Create a DataFrame for existing dates
         existing_df = pd.DataFrame({'Date': existing_dates})
         existing_df['year'] = existing_df['Date'].dt.year
         existing_df['month'] = existing_df['Date'].dt.month
         existing_df['day'] = existing_df['Date'].dt.day

         # Make predictions for existing dates
         existing_predictions = model.predict(existing_df[features])

         # Print the existing predictions
         print("Predictions for the existing 12 months:")
         for date, prediction in zip(existing_df['Date'], existing_predictions):
             print(f"{date}: {prediction}")
```

```
Predictions for the existing 12 months:
2023-03-01 00:00:00: 20.801586151123047
2023-04-01 00:00:00: 17.40984535217285
2023-05-01 00:00:00: 20.68121337890625
2023-06-01 00:00:00: 32.14493942260742
2023-07-01 00:00:00: 40.69474411010742
2023-08-01 00:00:00: 31.58974838256836
2023-09-01 00:00:00: 23.05022621154785
2023-10-01 00:00:00: 18.620975494384766
2023-11-01 00:00:00: 21.6868953704834
2023-12-01 00:00:00: 23.458681106567383
2024-01-01 00:00:00: 19.63335418701172
2024-02-01 00:00:00: 19.500045776367188
```

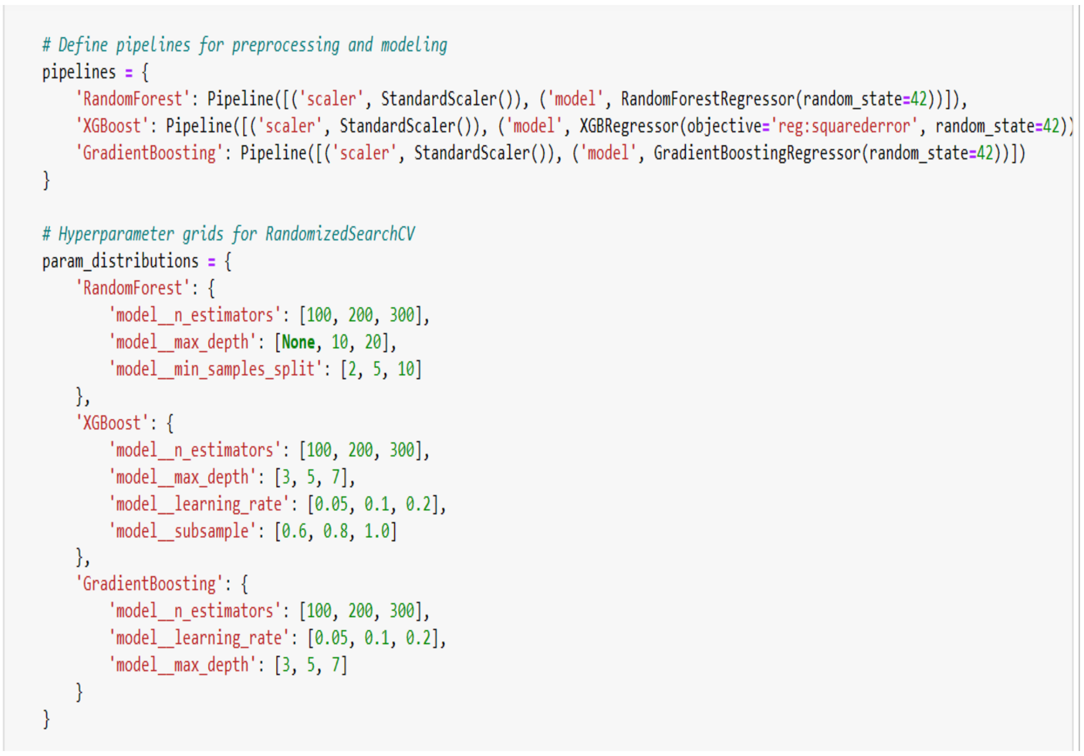
```
In [17]: # Generate dates for the next 12 months
         last_date = df['Date'].max()
         next_dates = pd.date_range(start=last_date + pd.DateOffset(days=1), periods=12, freq='MS')

         # Extract year, month, and day features
         next_dates_df = pd.DataFrame({
             'Date': next_dates,
             'year': next_dates.year,
             'month': next_dates.month,
             'day': next_dates.day
         })

         # Predict the target variable for the next 12 months
         predictions_next_12_months = model.predict(next_dates_df[['year', 'month', 'day']])
```

```
In [18]: # Visualize the predictions for the next 12 months
         plt.figure(figsize=(10, 6))
         plt.plot(df['Date'], df[target], label='True Data')
         plt.plot(df['Date'], model.predict(df[features]), label='Predictions', linestyle='--')
         plt.plot(next_dates, predictions_next_12_months, label='Predictions for Next 12 Months', linestyle='--', marker='o')
         plt.xlabel('Date')
         plt.ylabel(target)
         plt.title('True Data vs Predictions')
         plt.legend()
         plt.show()
```

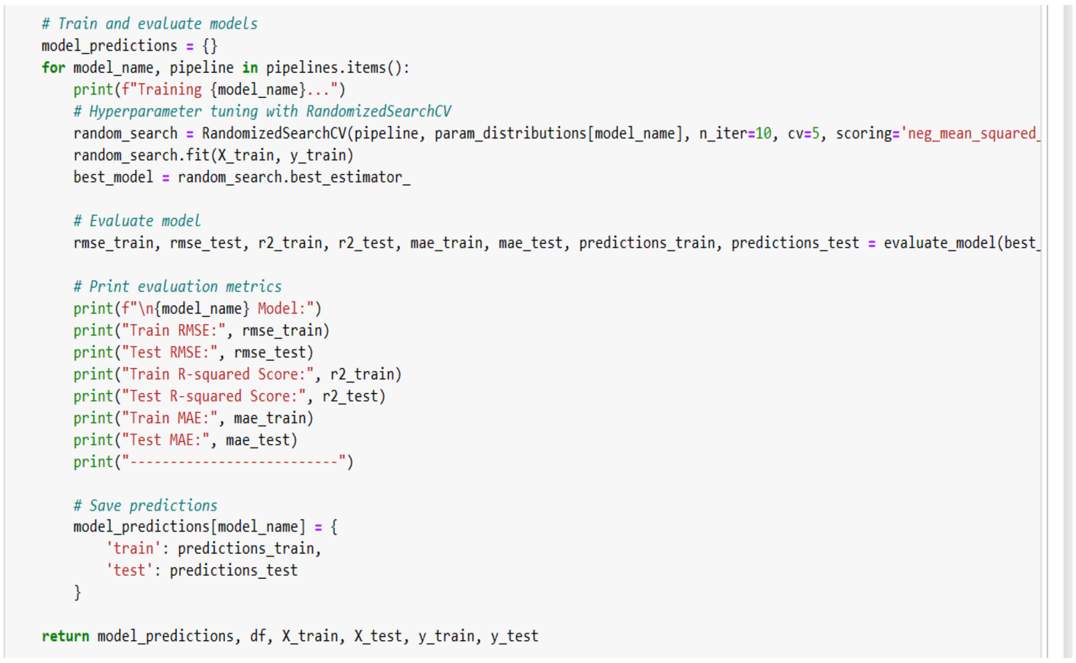


The above graph shows the predictions for the coming period, 2025. In the next steps we can see the performance of the Randomized search hyperparameter tuning technique so as to compare the forecasting accuracy.

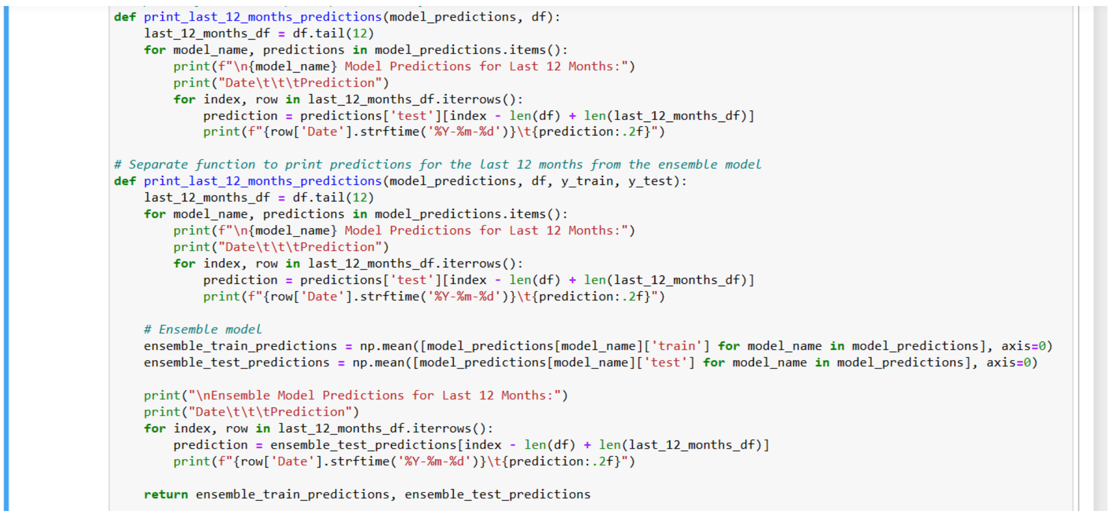
Randomized Search:



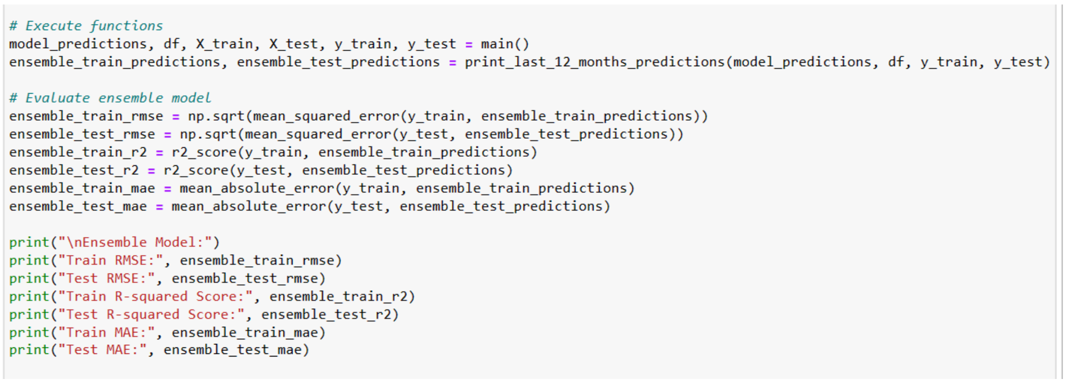
```
# Define pipelines for preprocessing and modeling
pipelines = {
    'RandomForest': Pipeline([('scaler', StandardScaler()), ('model', RandomForestRegressor(random_state=42))]),
    'XGBoost': Pipeline([('scaler', StandardScaler()), ('model', XGBRegressor(objective='reg:squarederror', random_state=42))
    'GradientBoosting': Pipeline([('scaler', StandardScaler()), ('model', GradientBoostingRegressor(random_state=42))])
}

# Hyperparameter grids for RandomizedSearchCV
param_distributions = {
    'RandomForest': {
        'model__n_estimators': [100, 200, 300],
        'model__max_depth': [None, 10, 20],
        'model__min_samples_split': [2, 5, 10]
    },
    'XGBoost': {
        'model__n_estimators': [100, 200, 300],
        'model__max_depth': [3, 5, 7],
        'model__learning_rate': [0.05, 0.1, 0.2],
        'model__subsample': [0.6, 0.8, 1.0]
    },
    'GradientBoosting': {
        'model__n_estimators': [100, 200, 300],
        'model__learning_rate': [0.05, 0.1, 0.2],
        'model__max_depth': [3, 5, 7]
    }
}
```



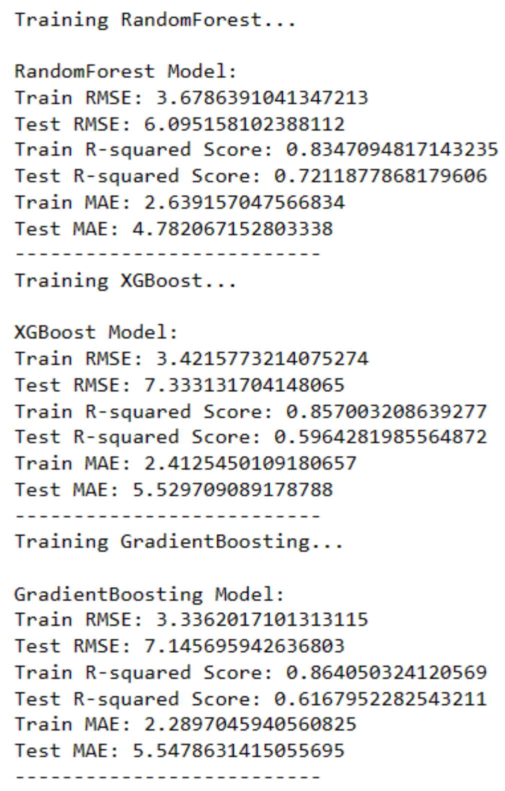
```
# Train and evaluate models
model_predictions = {}
for model_name, pipeline in pipelines.items():
    print(f"Training {model_name}...")
    # Hyperparameter tuning with RandomizedSearchCV
    random_search = RandomizedSearchCV(pipeline, param_distributions[model_name], n_iter=10, cv=5, scoring='neg_mean_squared_
    random_search.fit(X_train, y_train)
    best_model = random_search.best_estimator_

    # Evaluate model
    rmse_train, rmse_test, r2_train, r2_test, mae_train, mae_test, predictions_train, predictions_test = evaluate_model(best_

    # Print evaluation metrics
    print(f"\n{model_name} Model:")
    print("Train RMSE:", rmse_train)
    print("Test RMSE:", rmse_test)
    print("Train R-squared Score:", r2_train)
    print("Test R-squared Score:", r2_test)
    print("Train MAE:", mae_train)
    print("Test MAE:", mae_test)
    print("-------------------------")

    # Save predictions
    model_predictions[model_name] = {
        'train': predictions_train,
        'test': predictions_test
    }

return model_predictions, df, X_train, X_test, y_train, y_test
```



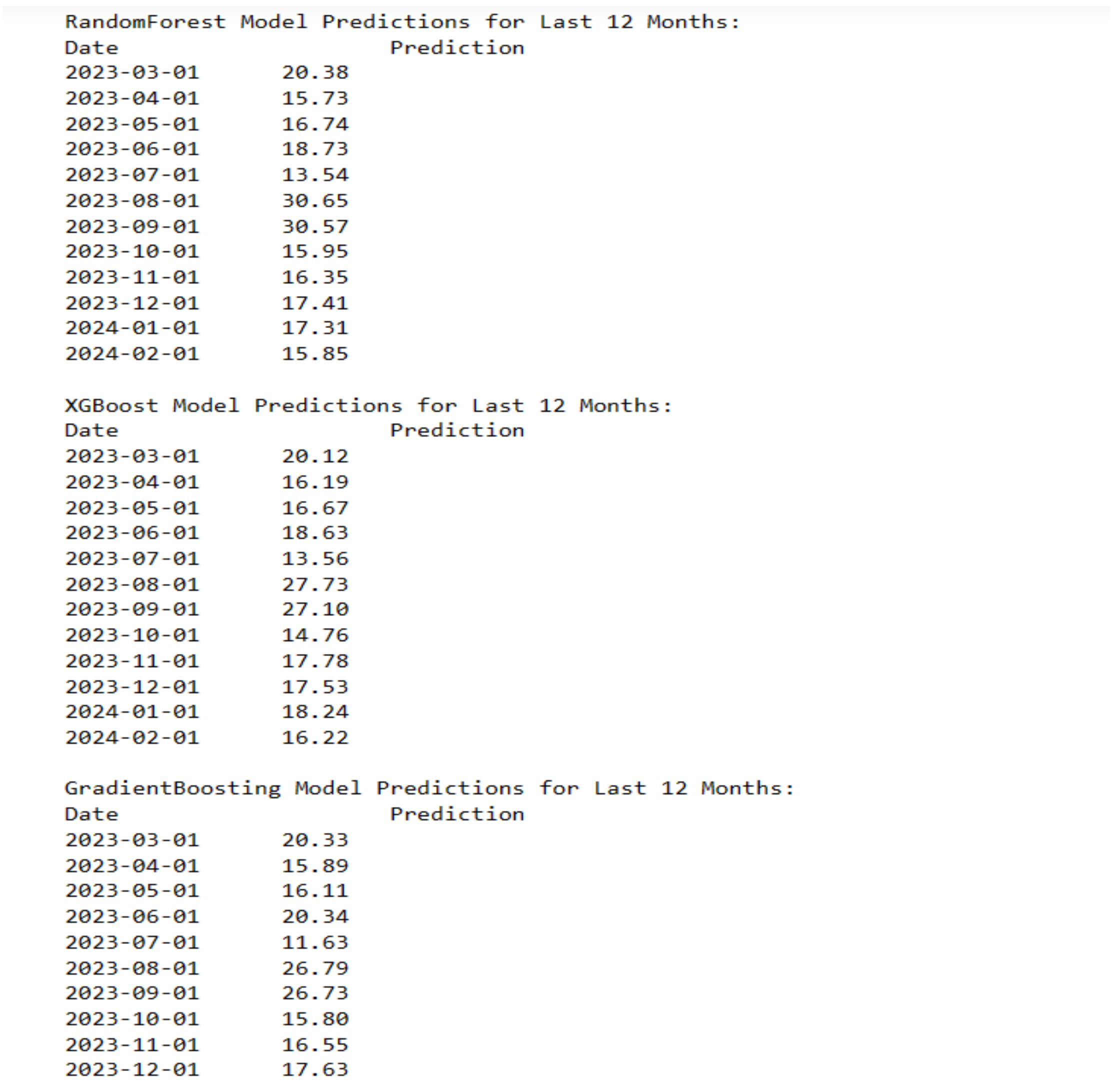
```
def print_last_12_months_predictions(model_predictions, df):
    last_12_months_df = df.tail(12)
    for model_name, predictions in model_predictions.items():
        print(f"\n{model_name} Model Predictions for Last 12 Months:")
        print("Date\t\t\tPrediction")
        for index, row in last_12_months_df.iterrows():
            prediction = predictions['test'][index - len(df) + len(last_12_months_df)]
            print(f"{row['Date'].strftime('%Y-%m-%d')}\t{prediction:.2f}")

# Separate function to print predictions for the last 12 months from the ensemble model
def print_last_12_months_predictions(model_predictions, df, y_train, y_test):
    last_12_months_df = df.tail(12)
    for model_name, predictions in model_predictions.items():
        print(f"\n{model_name} Model Predictions for Last 12 Months:")
        print("Date\t\t\tPrediction")
        for index, row in last_12_months_df.iterrows():
            prediction = predictions['test'][index - len(df) + len(last_12_months_df)]
            print(f"{row['Date'].strftime('%Y-%m-%d')}\t{prediction:.2f}")

    # Ensemble model
    ensemble_train_predictions = np.mean([model_predictions[model_name]['train'] for model_name in model_predictions], axis=0)
    ensemble_test_predictions = np.mean([model_predictions[model_name]['test'] for model_name in model_predictions], axis=0)

    print("\nEnsemble Model Predictions for Last 12 Months:")
    print("Date\t\t\tPrediction")
    for index, row in last_12_months_df.iterrows():
        prediction = ensemble_test_predictions[index - len(df) + len(last_12_months_df)]
        print(f"{row['Date'].strftime('%Y-%m-%d')}\t{prediction:.2f}")

    return ensemble_train_predictions, ensemble_test_predictions
```



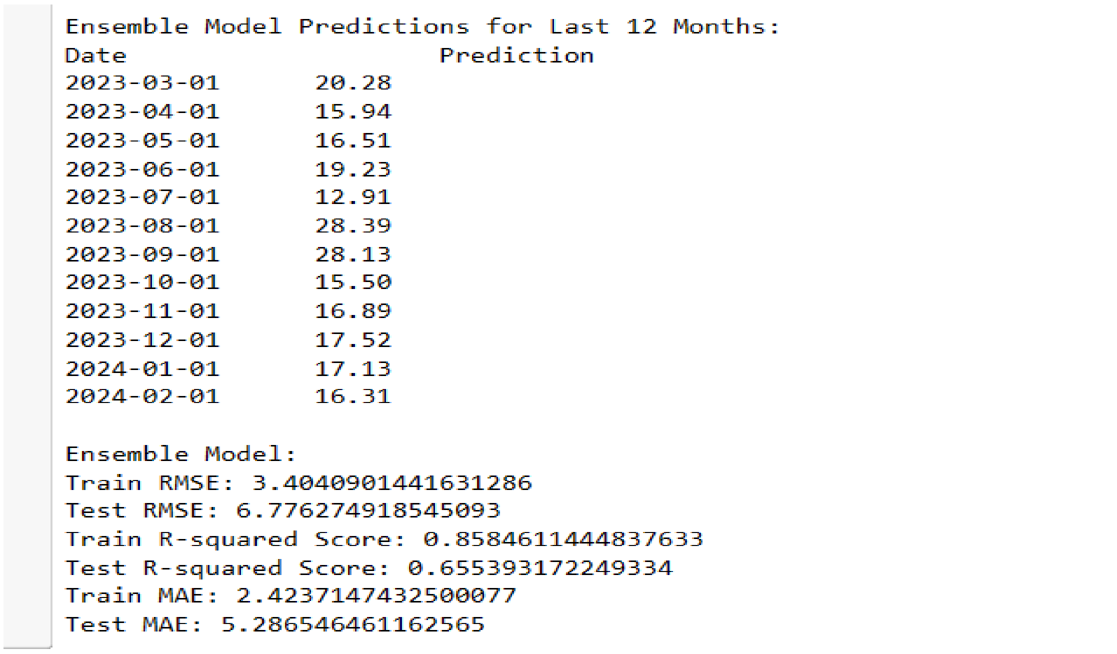
```
# Execute functions
model_predictions, df, X_train, X_test, y_train, y_test = main()
ensemble_train_predictions, ensemble_test_predictions = print_last_12_months_predictions(model_predictions, df, y_train, y_test)

# Evaluate ensemble model
ensemble_train_rmse = np.sqrt(mean_squared_error(y_train, ensemble_train_predictions))
ensemble_test_rmse = np.sqrt(mean_squared_error(y_test, ensemble_test_predictions))
ensemble_train_r2 = r2_score(y_train, ensemble_train_predictions)
ensemble_test_r2 = r2_score(y_test, ensemble_test_predictions)
ensemble_train_mae = mean_absolute_error(y_train, ensemble_train_predictions)
ensemble_test_mae = mean_absolute_error(y_test, ensemble_test_predictions)

print("\nEnsemble Model:")
print("Train RMSE:", ensemble_train_rmse)
print("Test RMSE:", ensemble_test_rmse)
print("Train R-squared Score:", ensemble_train_r2)
print("Test R-squared Score:", ensemble_test_r2)
print("Train MAE:", ensemble_train_mae)
print("Test MAE:", ensemble_test_mae)
```



```
Training RandomForest...

RandomForest Model:
Train RMSE: 3.6786391041347213
Test RMSE: 6.095158102388112
Train R-squared Score: 0.8347094817143235
Test R-squared Score: 0.7211877868179606
Train MAE: 2.639157047566834
Test MAE: 4.782067152803338
--------------------------
Training XGBoost...

XGBoost Model:
Train RMSE: 3.4215773214075274
Test RMSE: 7.333131704148065
Train R-squared Score: 0.857003208639277
Test R-squared Score: 0.5964281985564872
Train MAE: 2.4125450109180657
Test MAE: 5.529709089178788
--------------------------
Training GradientBoosting...

GradientBoosting Model:
Train RMSE: 3.3362017101313115
Test RMSE: 7.145695942636803
Train R-squared Score: 0.864050324120569
Test R-squared Score: 0.6167952282543211
Train MAE: 2.2897045940560825
Test MAE: 5.5478631415055695
--------------------------
```



```
RandomForest Model Predictions for Last 12 Months:
Date                     Prediction
2023-03-01       20.38
2023-04-01       15.73
2023-05-01       16.74
2023-06-01       18.73
2023-07-01       13.54
2023-08-01       30.65
2023-09-01       30.57
2023-10-01       15.95
2023-11-01       16.35
2023-12-01       17.41
2024-01-01       17.31
2024-02-01       15.85

XGBoost Model Predictions for Last 12 Months:
Date                     Prediction
2023-03-01       20.12
2023-04-01       16.19
2023-05-01       16.67
2023-06-01       18.63
2023-07-01       13.56
2023-08-01       27.73
2023-09-01       27.10
2023-10-01       14.76
2023-11-01       17.78
2023-12-01       17.53
2024-01-01       18.24
2024-02-01       16.22

GradientBoosting Model Predictions for Last 12 Months:
Date                     Prediction
2023-03-01       20.33
2023-04-01       15.89
2023-05-01       16.11
2023-06-01       20.34
2023-07-01       11.63
2023-08-01       26.79
2023-09-01       26.73
2023-10-01       15.80
2023-11-01       16.55
2023-12-01       17.63
```



```
Ensemble Model Predictions for Last 12 Months:
Date                     Prediction
2023-03-01       20.28
2023-04-01       15.94
2023-05-01       16.51
2023-06-01       19.23
2023-07-01       12.91
2023-08-01       28.39
2023-09-01       28.13
2023-10-01       15.50
2023-11-01       16.89
2023-12-01       17.52
2024-01-01       17.13
2024-02-01       16.31

Ensemble Model:
Train RMSE: 3.4040901441631286
Test RMSE: 6.776274918545093
Train R-squared Score: 0.8584611444837633
Test R-squared Score: 0.655393172249334
Train MAE: 2.4237147432500077
Test MAE: 5.286546461162565
```

5.2. Visualizations:

Installed Capacities:

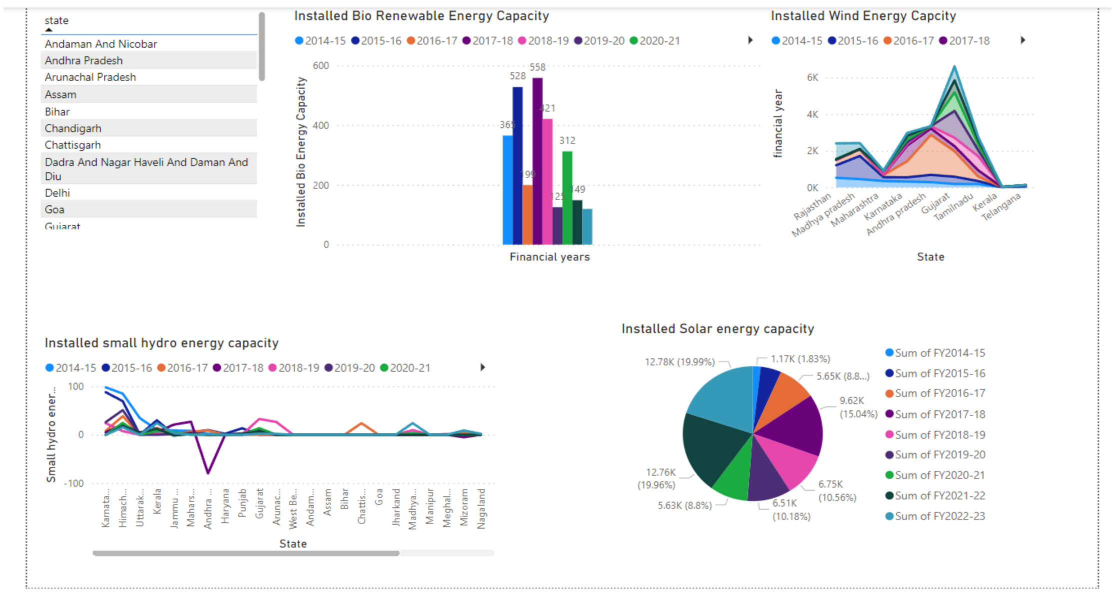


Fig 2. Visualization of Installed various power capacities in all states

Solar Energy:

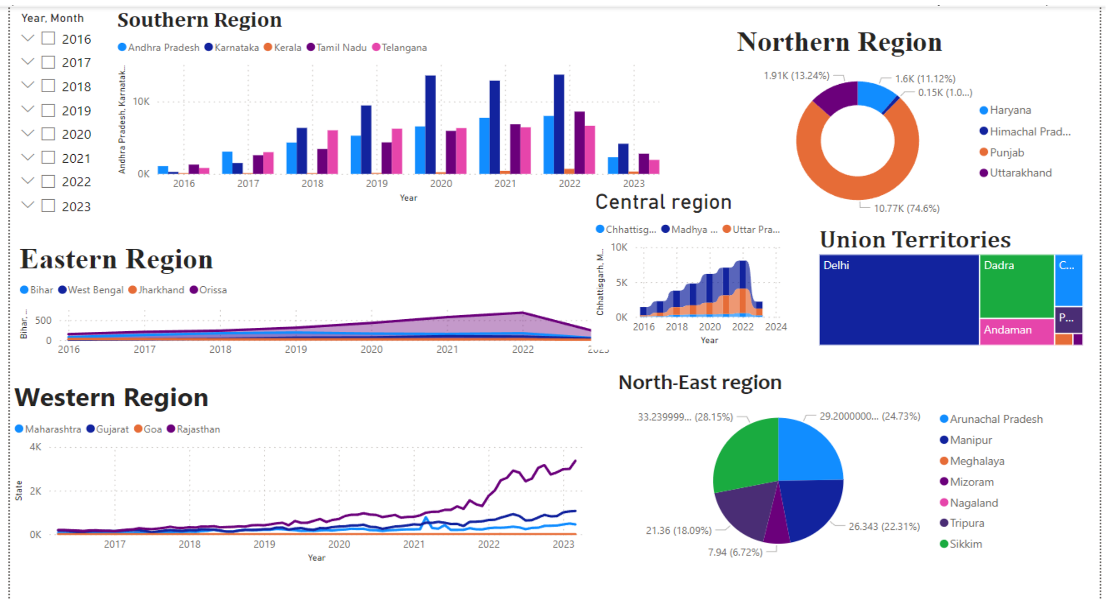


Fig 3. Visualization of Solar Energy across various regions

In solar energy, the states Karnataka, Punjab, Orissa, Madhya Pradesh, Rajasthan, Delhi, and Manipur have been constantly on rise across all the years.

Wind Energy:

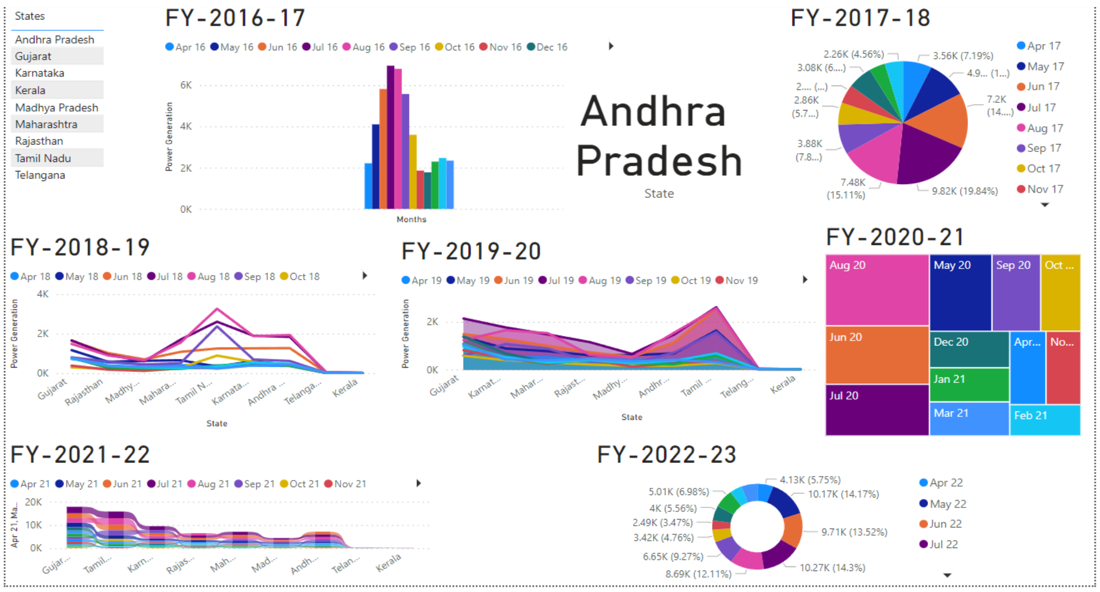


Fig 4. Visualization of Wind Energy in India

Wind energy is being produced constantly by 9 states which are Andhra Pradesh, Karnataka, Kerala, Tamil Nadu, Telangana, Gujarat, Rajasthan, Madhya Pradesh and Maharashtra in India.

Bio Energy:

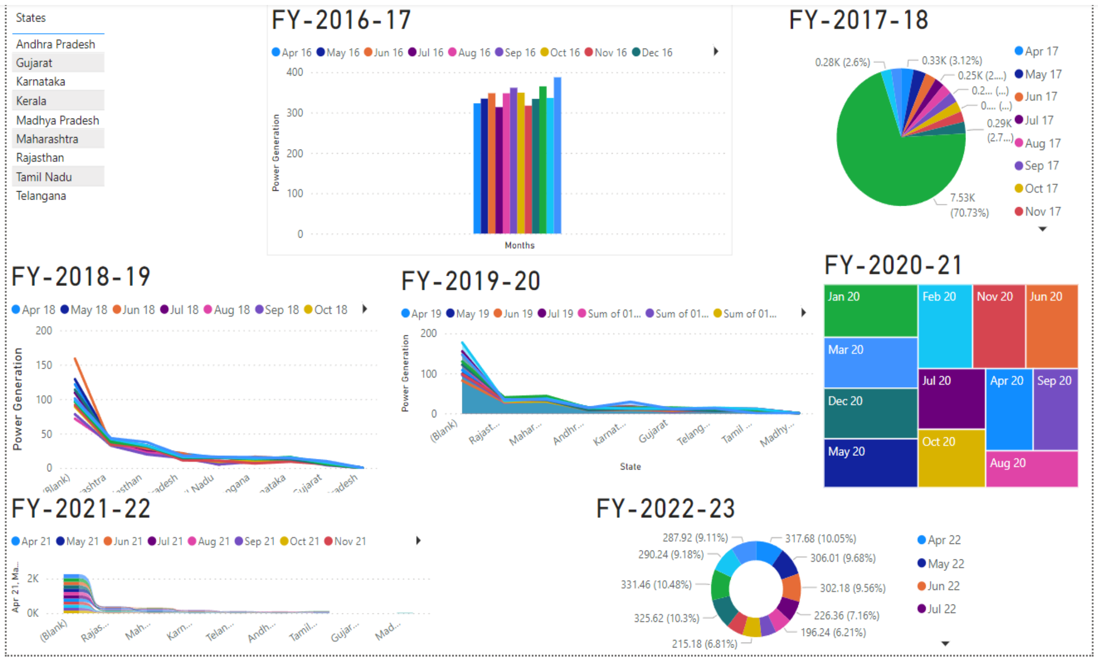


Fig 5. Visualization of Bio Energy in India

Bio energy is also produced by around 15 states in India including Southern states (Except Kerala) and few other states.

Hydro Energy:

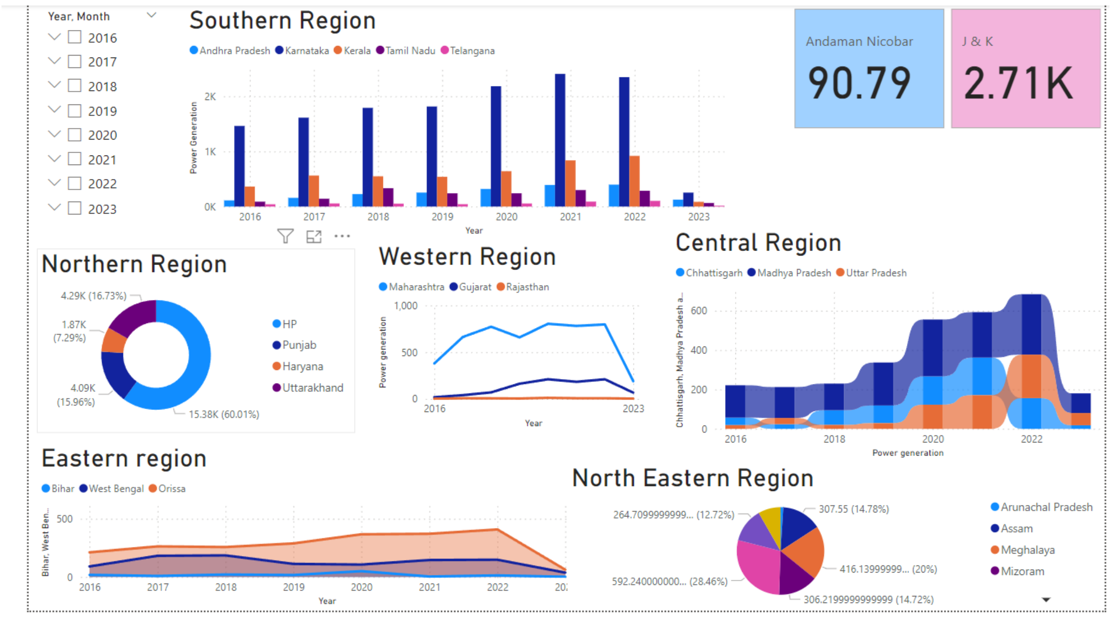


Fig 6. Visualization of Small Hydro energy

In Small hydro energy production, the states Karnataka, Himachal Pradesh, Maharashtra, Orissa, Nagaland are leading across all the years.

6. RESULT AND DISCUSSION

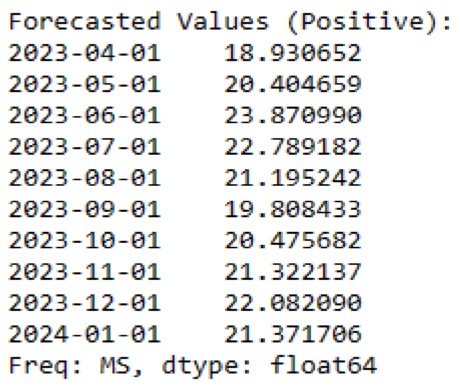
After executing different forecasting models on the data of the monthly power generation of different renewable energies. The models tested are Autoregressive Integrated Moving average (ARIMA) and Seasonal Integrated Moving Average (SARIMA) in the traditional time series models. Along with these Machine learning models like Extreme gradient boosting with Grid search hyper parameter tuning and an ensemble model of Random Forest, Gradient boost and Extreme gradient boost (XGBoost) with Random search hyperparameter tuning.

6.1. ARIMA:

After executing the ARIMA model for the monthly wind power generation data of the state Telangana, it was observed that the data is not stationary. This stationarity check has been done by using Augmented Dickey Fuller (ADF) test and it has given a p- value of 0.71 which is greater than 0.05. this means that the data is not stationary. In order to convert the data into stationary second differencing has been done and again the stationarity test has been conducted.

After converting the data into stationary using second differencing, Auto ARIMA function has been applied to check the best suited order for the data. The ARIMA model with order (5,0,0) gave the lesser AIC value, so it has been considered for model fitting.

After fitting the model, forecasting has been done for the data. And the forecasting is as follows



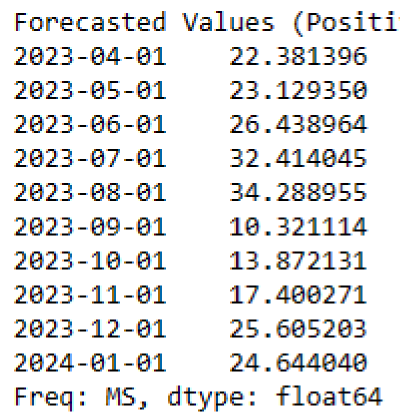
```
Forecasted Values (Positive):
2023-04-01    18.930652
2023-05-01    20.404659
2023-06-01    23.870990
2023-07-01    22.789182
2023-08-01    21.195242
2023-09-01    19.808433
2023-10-01    20.475682
2023-11-01    21.322137
2023-12-01    22.082090
2024-01-01    21.371706
Freq: MS, dtype: float64
```

And after calculating the RMSE value determined was 12.02 for this model.

6.2. SARIMA:

Seasonal Autoregressive Integrated Moving average (SARIMA) is useful when the data is having seasonality component. Since the monthly power generation is having seasonality, we have applied a SARIMA model set for a period of 12 months. The best order for this model is (5,0,0) (2,0,0) where the latter value represents the seasonality parameters.

After fitting the SARIMA model, the following is the forecast from the model



```
Forecasted Values (Positive):
2023-04-01    22.381396
2023-05-01    23.129350
2023-06-01    26.438964
2023-07-01    32.414045
2023-08-01    34.288955
2023-09-01    10.321114
2023-10-01    13.872131
2023-11-01    17.400271
2023-12-01    25.605203
2024-01-01    24.644040
Freq: MS, dtype: float64
```

And after calculating the metrics the RMSE value was determined as 9.57 for this model.

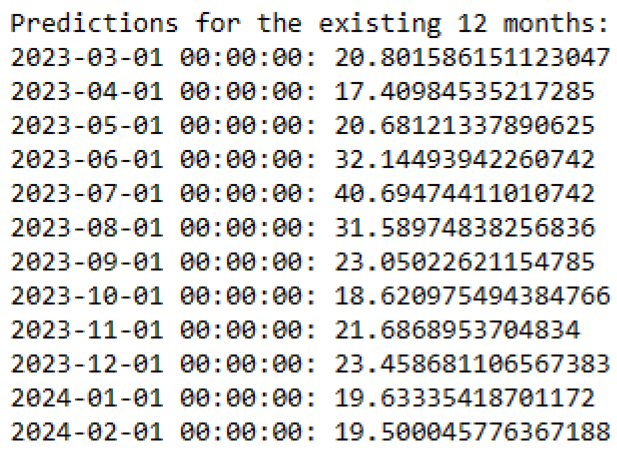
6.3. Machine Learning Models:

6.3.1. XGBoost:

Machine learning models have also been proved good for forecasting time series data. An Extreme Gradient boosting (XGBoost) model has been developed by taking the year, month and day as features and the state as the target variable. The data has been split into 80:20 ratio which means 80% of the data is taken as training set and 20% as testing or validation set.

To make the model more efficient hyperparameter tuning has been performed on the data and GridSearchCV method has been used in this model for the tuning. We define the parameters with various values while defining the grid. The training dataset is split into k subsets when a model is trained and evaluated k times, using one subset as the validation set and the remaining subsets as the training set. We call this method k-fold cross-validation. The optimal set of parameters is selected and applied to the training dataset based on the validation set's performance.

After executing the model, the forecast of the model has been follows.



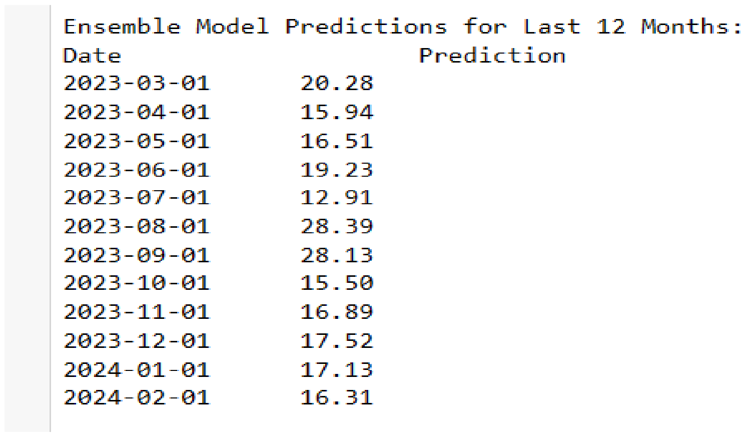
```
Predictions for the existing 12 months:
2023-03-01 00:00:00: 20.801586151123047
2023-04-01 00:00:00: 17.40984535217285
2023-05-01 00:00:00: 20.68121337890625
2023-06-01 00:00:00: 32.14493942260742
2023-07-01 00:00:00: 40.69474411010742
2023-08-01 00:00:00: 31.58974838256836
2023-09-01 00:00:00: 23.05022621154785
2023-10-01 00:00:00: 18.620975494384766
2023-11-01 00:00:00: 21.6868953704834
2023-12-01 00:00:00: 23.458681106567383
2024-01-01 00:00:00: 19.63335418701172
2024-02-01 00:00:00: 19.500045776367188
```

The RMSE value of the model on the testing set is 7.02 and it has given 0.76 and 0.63 as train and testing R-Squared scores.

6.3.2. Ensemble model:

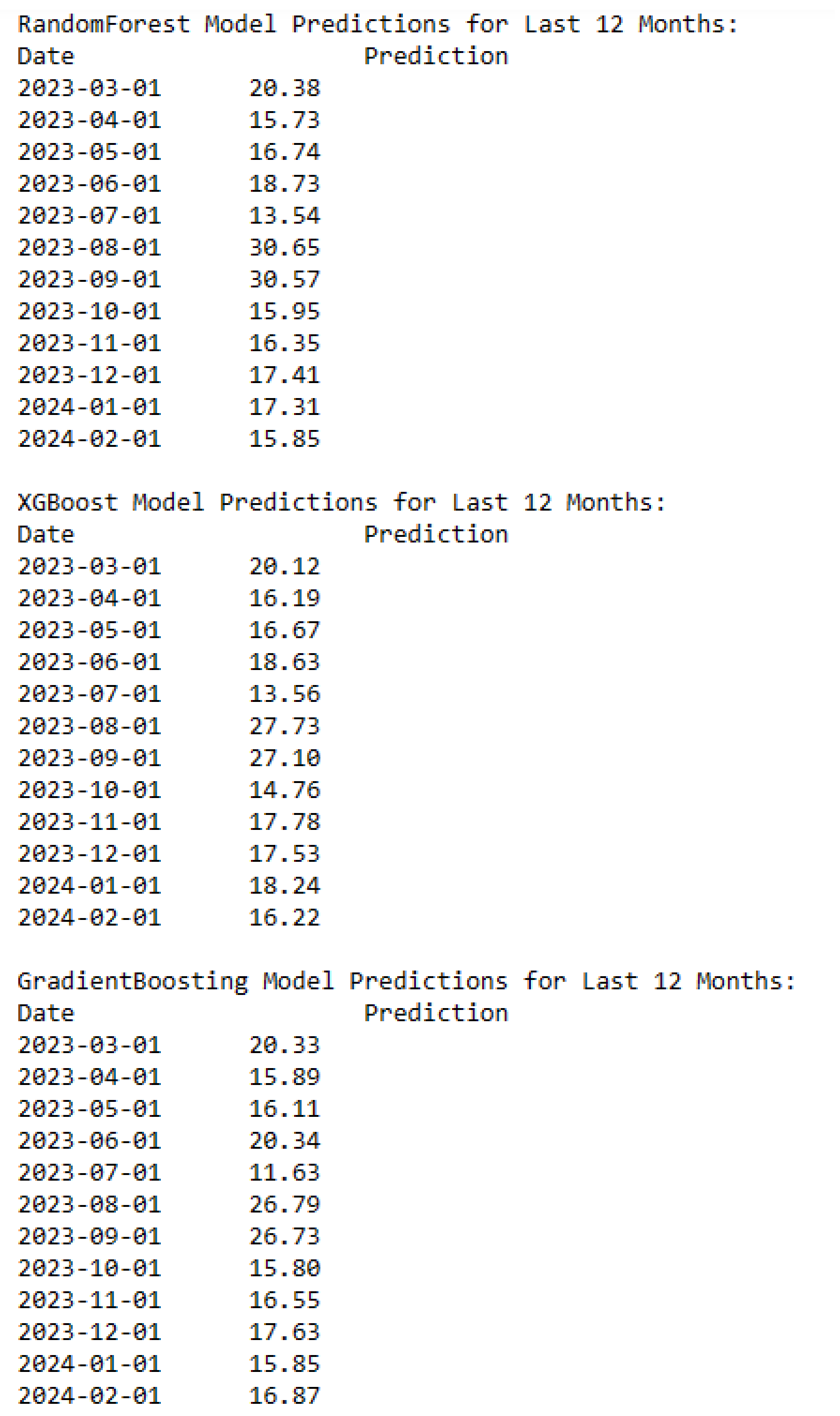
An ensemble model of Random Forest, Gradient boost, XGBoost has also been developed using Randomized search for hyper parameter tuning. Random search evaluates the combinations randomly in the specified range and selects the best set of parameters among them.

The individual models of Random Forest, Gradient Boost, XGBoost and ensemble model gave the following predictions:



```
Ensemble Model Predictions for Last 12 Months:
Date                           Prediction
2023-03-01      20.28
2023-04-01      15.94
2023-05-01      16.51
2023-06-01      19.23
2023-07-01      12.91
2023-08-01      28.39
2023-09-01      28.13
2023-10-01      15.50
2023-11-01      16.89
2023-12-01      17.52
2024-01-01      17.13
2024-02-01      16.31
```

.



```
RandomForest Model Predictions for Last 12 Months:
Date                        Prediction
2023-03-01      20.38
2023-04-01      15.73
2023-05-01      16.74
2023-06-01      18.73
2023-07-01      13.54
2023-08-01      30.65
2023-09-01      30.57
2023-10-01      15.95
2023-11-01      16.35
2023-12-01      17.41
2024-01-01      17.31
2024-02-01      15.85

XGBoost Model Predictions for Last 12 Months:
Date                        Prediction
2023-03-01      20.12
2023-04-01      16.19
2023-05-01      16.67
2023-06-01      18.63
2023-07-01      13.56
2023-08-01      27.73
2023-09-01      27.10
2023-10-01      14.76
2023-11-01      17.78
2023-12-01      17.53
2024-01-01      18.24
2024-02-01      16.22

GradientBoosting Model Predictions for Last 12 Months:
Date                        Prediction
2023-03-01      20.33
2023-04-01      15.89
2023-05-01      16.11
2023-06-01      20.34
2023-07-01      11.63
2023-08-01      26.79
2023-09-01      26.73
2023-10-01      15.80
2023-11-01      16.55
2023-12-01      17.63
2024-01-01      15.85
2024-02-01      16.87
```

The RMSE values of the models are 6.09 for Random Forest, 7.33 for the XGboost model and 7.14 for Gradient boost model. And the RMSE value of the ensemble model of all three models is 6.77.

7. CONCLUSION AND FUTURE SCOPE

7.1. Conclusion:

Through this immense data collection and visualization analysis we can observe the upward trajectory of the growth in renewable energy in many states. This usage of renewable energy has become a very viable and better alternative for the non renewable energy. This is also motivating the policy makers to initiate and design the plans accordingly for a better and healthy environment.

After comparing all the above models for forecasting this renewable energy generation based on the past monthly data of Solar, Wind, Small hydro and Bio mass energies, It has been observed that the ensemble model of the Random Forest, Gradient Boost and XGBoost gave a better result in the overall machine learning models and time series models where as Seasonal Autoregressive Integrated Moving average (SARIMA) gave better results compared to ARIMA in the traditional time series models alone. So, the forecast obtained through these models can be used for further analysis and actions.

7.2. Future Scope:

Deep learning models like Recurrent Neural Networks (RNN) namely Long Short-Term Memory (LSTM) and advanced models like FB Prophet etc can also be used for further analysis or comparison with the above tested models. But the main result or focus of all these models is to forecast the demand of the power generation through renewable energies. This demand forecast can be used to develop plans to increase the capacities of the renewable energy plants in the respective states. And also the effect of various factors in the energy production from these renewable energy sources can also be studied and analysed as that also plays a major role in the power generation and its demand.

8.REFERENCES

1. Aowabin Rahmana, V. S. (2017). Predicting electricity consumption for commercial and residential buildings using deep recurrent neural networks. *Applied energy*.
2. Bessa, J. R. (2017). Improving Renewable Energy Forecasting with a grid of numerical weather predictions. *IEEE TRANSACTIONS ON SUSTAINABLE ENERGY*.
3. Carla Gonc¸alves, P. P. (2020). Towards Data Markets in Renewable Energy Forecasting. *IEEE TRANSACTIONS ON SUSTAINABLE ENERGY*.
4. Conor Sweeney, R. J. (2019). The future of forecasting for renweable energy. *WIREs Energy and Environment*.
5. D.W. van der Meer, M. S. (2018). Probabilistic forecasting of electricity consumption, photovoltaic power generation and net demand of an individual building using Gaussian Processes. *Aoplied Energy* .
6. Huai-zhi Wang, G.-q. L.-b.-c.-t. (2016). Deep learning based ensemble approach for probabilistic wind power forecasting. *Applied Energy*.
7. Huaizhi Wanga, Z. L. (2019). A review of deep learning for renewable energy forecasting. *Elsevier*.
8. Huiting Zheng, J. Y. (2017). Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation. *Energies*.
9. J. Antonanzas, N. O.-d.-P.-T. (2016). Review of photovoltaic power forecasting. *Solar Energy*.
10. Jakob W. Messner, P. P. (2018). Online adaptive lasso estimation in vector autoregressive models for high dimensional wind power forecasting. *International Journal of Forecasting*.
11. KiJeon Nam, S. H. (2020). A deep learning-based forecasting model for renewable energy scenarios to guide sustainable energy policy : A case study of Korea. *Renewable and Sustainable Energy Reviews*.

12. Laura Cavalcante, R. J. (2016). LASSO vector autoregression structures for very short-term wind power forecasting. *Wind Energy*.
13. Md. Siddikur Rahman, A. H. (2022). Accuracy comparison of ARIMA and XGBoost forecasting models in predicting the incidence of COVID-19 in Bangladesh. *PLOS GLOBAL PUBLIC HEALTH*.
14. Michelangelo Ceci, R. C. (2019). Spatial autocorrelation and entropy for renewable energy forecasting. *Data Mining and Knowledge Discovery*.
15. Ricardo J. Bessa, C. ,.-M. (2017). Towards Improved Understanding of the applicability of uncertainity forecasts in the electric power industry. *energies*.
16. Roberto Corizzo, M. C.-T. (2020). Multi-aspect renewable energy forecasting. *Information sciences*.
17. Singh, B. R. (2016). Future Scope of Solar Energy in India. *S-JPSET*.
18. Sultan Al-Yahyai, Y. C. (2010). Review of the use of Numerical Weather Prediction (NWP) Models for wind energy assessment. *Renewable and Sustainable Energy Reviews*.
19. Tanveer Ahmad, H. C. (2019). Deep learning for multi-scale smart energy forecasting. *Energy*.
20. William Y.Y. Cheng, Y. L. (2017). Short-term wind forecast of a data assimilation/weather forecasting system with wind turbine anemometer measurement assimilation. *Renewable Energy*.
21. Wu, H. Z. (2019). A XGBoost Model withWeather Similarity Analysis and Feature Engineering for Short-Term Wind Power Forecasting. *Applied Sciences*.
22. Xie dairu, Z. S. (2021). Machine Learning Model for Sales Forecasting by using XGBoost. *IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE 2021).*
23. Ye Ren, P. ,. (2015). Ensemble methods for wind and solar power forecasting - A state-of-the-art review. *Renewable and Sustainable Energy Reviews*.

24. Yuanyuan Wang, S. S. (2021). Short-term load forecasting of industrial customers based on SVMD and XGBoost. *International Journal of Electrical Power and Energy Systems*.

25. Yun Wang, H. W. (2018). Robust functional regression for wind speed forecasting based on Sparse Bayesian learning. *Renewable Energy*.