A
PROJECT REPORT
ON

EV RANGE PREDICTOR
USING MACHINE LEARNING

Submitted in Partial fulfillment of the Requirement for award the degree of

BACHELOR OF TECHNOLOGY
In
ELECTRONICS ENGINEERING

Submitted by:
ARCHI JAISWAL (19415)
ARUSHI TRIVEDI (19417)
RITIKA JAIN (19447)

Guided by:
Prof. Y.K. Mishra
H.O.D
Department of Electronics

DEPARTMENT OF ELECTRONICS ENGINEERING
KAMLA NEHRU INSTITUTE OF TECHNOLOGY
SULTANPUR (U.P), 228118

SESSION 2022-23

DEPARTMENT OF ELECTRONICS ENGINEERING

Kamla Nehru Institute of Technology, Sultanpur

CERTIFICATE

Certified that **Archi Jaiswal** (RN. 19415), **Arushi Trivedi** (RN. 19417), **Ritika Jain** (RN. 19448) has carried out the project work presented in this report entitled **"EV RANGE PREDICTOR USING MACHINE LEARNING"** for award the degree of **Bachelor of Technology** in **Electronics Engineering** at **Kamla Nehru Institute of Technology, Sultanpur** (U.P.), affiliated to Dr. A.P.J Abdul Kalam Institute of Technology, Lucknow under my supervision. The project embodies results of original work, and studies are carried out by the students themselves and the contents of the report do not used for the award of any other degree to the candidates or anybody else from this or any other University/Institution.

(Prof. Y.K. Mishra)
Head of Department

(Prof. Y.K. Mishra)
Project Guide

Date: May, 2023
Place: Sultanpur

ACKNOWLEDGEMENT

We are highly obliged to Kamla Nehru Institute of Technology, Sultanpur for allowing us to pursue a Bachelor of Technology in Electronics Engineering and providing facilities to pursue this project.

We express our deepest sense of gratitude towards our Head of Department Prof. Y.K. Mishra Sir, Department of Electronics Engineering of Kamla Nehru Institute of Technology, Sultanpur, for his patience, inspiration, guidance, constant encouragement, moral support, keen interest, and valuable suggestion during preparation of this project report.

We are thankful to our Supervisor and Project guide Prof. Y.K Mishra Sir, of Electronics Engineering, for his support and valuable suggestions during preparation of this project report. The work would not have been successfully accomplished without his inspiration and generous guidance.

Our heartfelt gratitude goes to Prof. S.P. Gangwar, Prof. Harsh Vikram Singh and all other faculty members of the Electronics Engineering department who, with their encouraging and caring words and most valuable suggestions have contributed directly or indirectly in a significant way towards completion of this project report.

We are indebted to all our classmates from Electronics Engineering, for taking interest in discussing our problems and encouraging us.

We owe a debt of gratitude to our family for their consistent support, sacrifice, candid views and meaningful suggestions given to us at the different stages of our work.

Last but not the least, I am thankful to the almighty who gave us the strength and health for completing our project.

(Candidate Signature)

DECLARATION

We, the students of 8th semester of Electronics Engineering, Kamla Nehru Institute of Technology, Sultanpur declare that the work entitled " **EV Range Predictor using Machine Learning**" has been successfully completed under the guidance of Prof. Y.K. Mishra Sir, Electronics Engineering Department, Kamla Nehru Institute Of Technology, Sultanpur. This dissertation work is submitted in partial fulfillment requirements for the award of Degree of Bachelor of Technology in Electronics Engineering during the academic year 2022-2023.

Further the matter embodied in the project report has not been submitted previously by anybody for the award of any degree or diploma to any university.

Place:

Date:

Team Members:

Archi Jaiswal (19415)

Arushi Trivedi (19417)

Ritika Jain (19447)

TABLE OF CONTENTS

ABSTRACT

Electric vehicles (EVs) have gained immense popularity in recent years, with the global push towards cleaner and more sustainable energy sources. However, a major challenge in widespread adoption of EVs is range anxiety, i.e., the fear of running out of battery charge before reaching the destination. In order to alleviate this concern, we propose the development of an EV range predictor using machine learning (ML) algorithms.

The EV range predictor will utilize a variety of inputs, including battery charge level, driving speed, road elevation, temperature, and weather conditions, to accurately predict the remaining driving range of an EV. The model will be trained using historical data from a fleet of EVs, collected over a variety of driving conditions and scenarios.

The proposed EV range predictor will not only provide drivers with accurate range estimates, but will also help optimize driving behavior.

CHAPTER 1:INTRODUCTION TO ELECTRIC VEHICLE AND ELECTRIC VEHICLE RANGE PREDICTOR

1.1 Introduction

Electric vehicles (EVs) are an attractive alternative to conventional vehicles powered by internal combustion engines due to their low carbon footprint, low running cost, and higher energy efficiency. These vehicles use electric motors and rechargeable batteries to power the vehicle, which produces zero emissions and can significantly reduce the carbon footprint associated with transportation.

This project aims to overcome the limitations of current range predictors by incorporating latest ML techniques to improve the accuracy of range predictions. ML algorithms are particularly effective for handling large amounts of data and identifying complex patterns, which is crucial for accurately predicting EV range. The proposed EV range predictor will also provide valuable insights into driving behavior and help drivers make more informed decisions about their driving habits and charging practices.

1.2 WHAT IS A RANGE PREDICTOR?

A range predictor in electric vehicles is a system that uses a variety of factors to estimate how far the vehicle can travel on a single charge. The factors that are typically considered include the vehicle's battery capacity, the current state of charge of the battery,auxiliary power, and the driver's behavior.

Range predictors are used to help drivers plan their trips and to avoid running out of power before reaching their destination. They can also be used to help improve the efficiency of electric vehicles by providing feedback to drivers about how they can reduce their energy consumption.

The accuracy of range predictors can vary depending on a number of factors, including the accuracy of the data that is used to train the algorithm, and the driver's behavior. Here are some of the factors that can affect the accuracy of range predictors:

- The vehicle's battery capacity. The larger the battery capacity, the longer the vehicle's range.
- The current state of charge of the battery. The lower the state of charge, the shorter the vehicle's range.
- The driving conditions. The driving conditions can have a significant impact on the vehicle's range. For example, driving at high speeds or in cold weather can reduce the range.
- The driver's behavior. The driver's behavior can also have a significant impact on the vehicle's range. For example, hard acceleration and braking can reduce the range.

Range predictors can be a valuable tool for drivers of electric vehicles. They can help drivers plan their trips and avoid running out of power before reaching their destination. They can also help improve the efficiency of electric vehicles by providing feedback to drivers about how they can reduce their energy consumption.

1.3 ACCURACY CONCERNS OF RANGE PREDICTOR

1.4 EXISTING METHODS FOR RANGE PREDICTION

There are few existing methods used in electric vehicle(EV) range prediction. These methods utilize various techniques and data sources to estimate the remaining range of an EV based on different parameters. Generally, the remaining driving range of EV is estimated by employing various **simple physics-based models** and **NREL's FASTSim** model.

1.4.1 PHYSICS BASED MODEL

The Physics-based vehicle energy consumption model utilizes energy recuperation model by regenerative braking, energy consumption by auxiliary electric devices and dual automatic temperature control, battery charging and discharging efficiency map, etc. However, these physics-based models do not consider nonlinear and uncertain state information, and moreover a way to simply combine the multiple physics-based models may frequently overlook complex dynamic states that cannot be ignored. The physics-based model works by using equations to estimate the power required to move the vehicle under different driving conditions. The model then calculates the amount of energy required to move the vehicle over a specific distance, taking into account factors such as energy losses due to friction, aerodynamic drag, and battery efficiency. Using this information, the model can estimate the remaining range of the vehicle based on its current state of charge and the driving conditions. A physics-based model was implemented in EVPRE using the following formula:

$$P_{\text{wheels}}(T) = \frac{v(t)}{P_{\text{motor}} P_{\text{driveline}}} \left(ma(t) + mg\cos(\theta)\frac{C_r}{1000}(C_1 v(t) + C_2) + 0.5\rho_{\text{air}} A_f C_D V_w(t)^2 + mg\sin(\theta) \right)$$

The physics-based model takes the power cost at the wheels so the true power cost must be translated up to the motor which is done by the division of *P*motor and *P*driveline.

Definition and value of variables used in the physics model.

Variable	Definition	Value
m	mass of vehicle	vehicle specific
a	acceleration of the vehicle	trip specific
g	gravitational acceleration	9.81
θ	road grade	trip specific
C_r	rolling resistance surface type	1.75
C_1	rolling resistance road condition	0.0328
C_2	rolling resistance tire type	4.575
ρ_{air}	air mass density	1.2256
A_f	frontal area of the vehicle	vehicle specific
C_D	drag coefficient	vehicle specific
V_w	wind speed	trip specific
$v(t)$	vehicle speed	trip specific
P_{motor}	electric motor efficiency	0.91
$P_{driveline}$	driveline efficiency	0.92

In general, simple physics-based models can have an accuracy within the range of 10-20% compared to the actual range achieved by an electric vehicle.

1.4.2 FASTSIM MODEL

FASTSim (Fleet Analysis and Simulation Tool) is a widely used model for electric vehicle (EV) range prediction and energy consumption analysis. It is developed by the National Renewable Energy Laboratory (NREL) in the United States and is specifically designed for fleet-level analysis.

FASTSim utilizes a combination of empirical and physics-based modeling techniques to estimate the energy consumption and range of EVs. It takes into account various factors such as vehicle characteristics, driving conditions, and energy usage patterns.

The model uses a large database of vehicle dynamometer test data, which includes information on vehicle powertrain efficiency, energy consumption, and performance under different driving cycles. By comparing the real-time driving conditions to the available data, FASTSim estimates the energy consumption and range for a given EV. The FASTSim model utilizes the FASTSim energy algorithm to calculate node edge power cost. It does this by creating a small simulation (cycle in FASTSim terminology), and within that simulation

creating a small slice emulating the edge's length (i.e., road section length) and one way height differential.Typically, these models can achieve an accuracy within the range of 5-10% compared to the actual range achieved by an electric vehicle.

1.5 PROPOSED SOLUTION

- We propose to design and develop a machine learning model that accurately predicts the remaining range of an electric vehicle based on various parameters such as battery charge level, driving speed, road elevation, temperature, weather conditions, etc.

- We wish to use machine learning models to predict the range of the Electric Vehicle based on the current charge state and the exogenous factors.

- While also using conventional machine learning models to perform our predictions. We also wish to propose and implement a novel approach that involves ensembling multiple machine learning models to further improve the accuracy of our predictions

CHAPTER 2: DETAILED DESCRIPTION OF THE PARAMETERS WHICH AFFECT THE RANGE OF AN ELECTRIC VEHICLE

MOTOR POWER

The motor power of an electric vehicle (EV) can impact its range in several ways:

Energy consumption: The motor power directly affects the energy consumption of the EV. A more powerful motor generally requires more energy to operate, especially during acceleration and maintaining higher speeds. As a result, a higher motor power can lead to increased energy consumption, which reduces the overall range of the EV.

Efficiency: The efficiency of the motor plays a significant role in determining the impact of motor power on range. While a more powerful motor may consume more energy, it doesn't necessarily mean it is less efficient. Modern electric motors are designed to be highly efficient across a range of power outputs. Therefore, a higher power motor may still have good efficiency, allowing for an acceptable range even with increased power.

Weight: A more powerful motor may be larger and heavier, which can increase the overall weight of the vehicle. The added weight can negatively impact the EV's energy efficiency and range. The vehicle will require more energy to move the increased weight, resulting in decreased range.

Driving style and acceleration: A higher motor power can enable faster acceleration and improved performance. However, aggressive driving and frequent rapid acceleration can significantly increase energy consumption and reduce the EV's range. Drivers who frequently utilize the full power of a high-power motor may experience a decrease in range compared to those who adopt a more energy-conscious driving style.

Regenerative braking: EVs often utilize regenerative braking, which converts some of the kinetic energy during braking into electrical energy to recharge the battery. A more powerful motor can potentially generate more regenerative energy during braking, which helps extend the range of the vehicle.

ODOMETER

The odometer reading, which indicates the total distance traveled by an electric vehicle (EV), does not directly affect its range. The range of an EV primarily depends on factors such as battery capacity, driving conditions, driving style, and energy efficiency. However, the odometer reading indirectly affects the range in the following ways:

Battery degradation: As an EV accumulates more miles, the battery may experience gradual capacity degradation. Over time, the battery's ability to hold a charge and provide the same range it did when new can diminish. So, while the odometer reading itself doesn't impact range, the age and usage of the battery, which are correlated with mileage, can affect its health and, consequently, its range.

Battery management system (BMS): The BMS of an EV monitors various factors, including the odometer reading, to estimate and manage battery performance. The BMS uses historical data to predict range, estimate state of charge, and ensure the battery operates within safe limits. The accumulated mileage may be taken into account by the BMS when providing range estimates or triggering maintenance reminders.

ESTIMATED CABIN PTC HEATER POWER()to update

The Estimated Cabin PTC (Positive Temperature Coefficient) Heater power in an electric vehicle (EV) can have an impact on its range. Here's how:

Energy consumption: The PTC heater is responsible for heating the cabin of the EV, providing comfort to the occupants during cold weather.

The power rating of the PTC heater indicates how much energy it consumes to generate heat. A higher power rating means the PTC heater draws more power from the battery, which increases energy consumption and reduces the overall range of the EV.

Battery drain: The power required to operate the PTC heater is supplied by the EV's battery. When the PTC heater is running at a higher power level, it draws more current from the

battery, resulting in increased battery drain. This can significantly impact the range of the EV, as a larger portion of the available energy is utilized for heating instead of propelling the vehicle.

Efficiency considerations: PTC heaters are known for their relatively lower efficiency compared to other heating technologies. They convert electrical energy into heat, and some energy is inevitably lost during the conversion process. Consequently, higher power PTC heaters may have lower overall efficiency, meaning more energy is consumed for the same amount of heat output. This lowers the range of the EV, as more energy is required to maintain cabin temperature.

Temperature management: The power rating of the PTC heater is designed to meet the heating demands of the cabin under various external temperature conditions. In colder climates, where higher heating power is required to maintain a comfortable temperature, the range of the EV will be affected more compared to milder climates with lower heating demands.

Efficient use of the PTC heater can help mitigate its impact on range. For example, preheating the cabin while the vehicle is still plugged into a charger can utilize grid power instead of draining the battery. Additionally, optimizing the use of seat heaters and cabin zoning can reduce the reliance on the PTC heater and minimize its energy consumption.

ELEVATION

The elevation of a place can indeed affect the range of an electric vehicle (EV). Here's how:

Energy consumption: When driving uphill, an EV needs to exert more power to overcome the force of gravity. This results in increased energy consumption and can reduce the overall range of the vehicle.

The steeper the incline, the more energy is required to climb it, leading to a greater impact on the range.

Regenerative braking: On the flip side, when driving downhill or descending an incline, EVs can take advantage of regenerative braking. This technology allows the vehicle to recover

some of the energy expended during acceleration by converting the kinetic energy back into electrical energy and storing it in the battery. Descending downhill or driving on a downward slope can help recharge the battery to some extent, thus extending the range.

Aerodynamic resistance: Elevation does not directly influence aerodynamic resistance, but higher altitudes may have thinner air, which can result in slightly reduced air density. Lower air density can have a minor positive effect on reducing aerodynamic drag, leading to slightly improved efficiency and range. However, the impact of altitude on aerodynamics is generally negligible compared to other factors.

Temperature variations: Elevation often correlates with changes in temperature. Cold weather can negatively affect an EV's battery performance and reduce its range. However, this impact is due to temperature rather than elevation itself. In colder climates, it's advisable to pre-condition the battery before driving to mitigate the adverse effects of low temperatures.

CONTROL PILOT

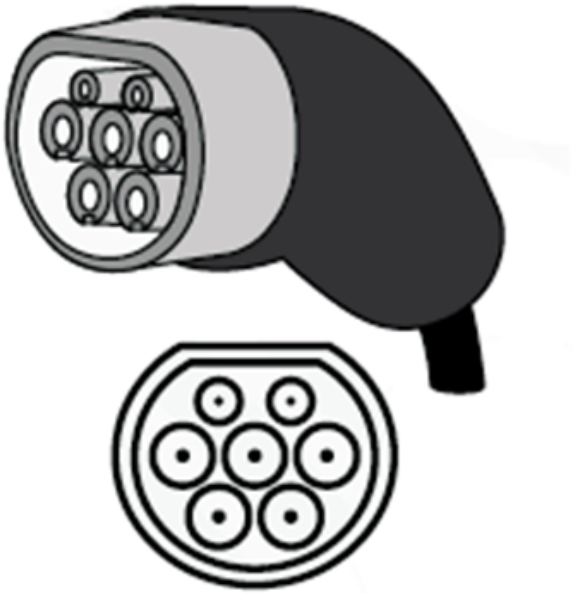
The control pilot, also known as the pilot signal or pilot communication, is an essential component of the charging infrastructure for electric vehicles (EVs). It enables communication between the charging station and the EV's onboard charger, allowing for proper coordination and control of the charging process.

In an electric vehicle, a control pilot (also known as a charge control pilot or CCP) is a communication signal that is sent between the charging station and the vehicle to ensure safe and efficient charging.

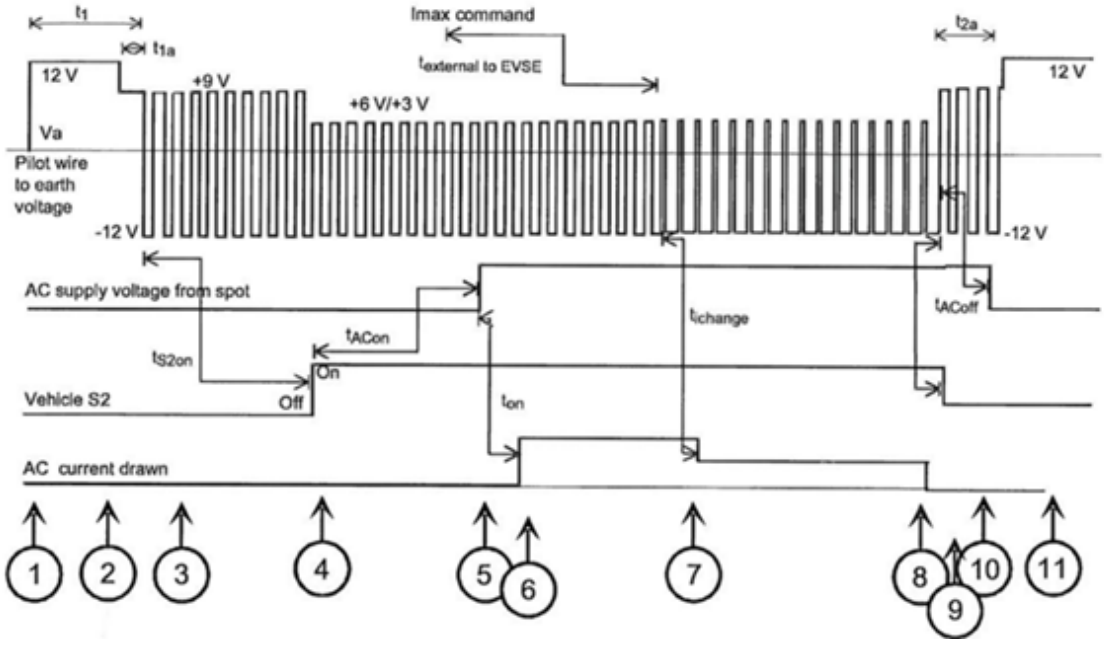
The control pilot serves several important functions. First, it verifies that the vehicle is properly connected to the charging station, and that the station is capable of providing the necessary amount of power to charge the vehicle's battery.

Second, the control pilot ensures that the vehicle and charging station are communicating effectively and that they are compatible. It does this by establishing a common language between the two devices, and by verifying that the charging station is configured correctly to provide the appropriate voltage and current to the vehicle.

Finally, the control pilot helps to manage the charging process itself, by controlling the rate of charge and monitoring the battery's state of charge. This helps to prevent overcharging or overheating of the battery, which can be dangerous or damaging to the vehicle.



Different charging states of an EV:



It helps determine the charging rate, manage the charging process, and ensure compatibility between the vehicle and the charging infrastructure.

Typical charging cycle under normal operating conditions

While the control pilot signal does not have a direct impact on the EV's range, it plays a role in the charging process, which indirectly affects the range. Here's how it can influence range:

Charging speed: The control pilot signal is used to negotiate and determine the charging rate between the vehicle and the charging station. Different control pilot signals correspond to different charging rates (e.g., Level 1, Level 2, DC fast charging). A higher charging rate allows the battery to be replenished more quickly, reducing the time spent charging and potentially increasing the available range for the EV.

Charging infrastructure compatibility: The control pilot signal ensures compatibility between the vehicle and the charging infrastructure. If there is a mismatch or compatibility issue between the control pilot signals, the charging session may not initiate or may operate at a lower charging rate. In such cases, the charging time may be prolonged, which indirectly affects the overall range of the EV.

STATE OF CHARGE

The state of charge (SOC) of an electric vehicle (EV) battery directly affects its range. Here's how:

Available energy: The SOC of the battery represents the amount of energy stored in it. A higher SOC means more energy is available for the vehicle to use, resulting in a greater range. Conversely, a lower SOC means less energy is available, leading to a reduced range.

Energy density: Battery systems typically have a higher energy density towards their nominal capacity. This means that as the battery SOC decreases, the available energy per unit of weight or volume decreases as well. Consequently, the range of an EV will decrease as the SOC decreases due to the diminishing energy density of the battery.

Battery management system: EVs are equipped with a battery management system (BMS) that monitors the SOC and ensures the battery operates within safe limits. The BMS takes into account the SOC when estimating the range and provides accurate information to the driver or vehicle's control system.

Performance degradation: The range of an EV can be affected by the battery's performance degradation over time. The battery's capacity may decrease gradually as it undergoes charge and discharge cycles. As a result, the actual range achieved at a specific SOC may be lower compared to when the battery was new.

Range estimation: The range estimation displayed by an EV often takes into account the current SOC of the battery. The estimation considers various factors such as driving conditions, historical energy consumption, and the battery's characteristics to provide a projected range based on the available SOC.

A/C POWER

The range of an electric vehicle (EV) can be affected by several factors, including the use of air conditioning (A/C) while driving. A/C systems in EVs consume electrical power, and this power comes from the vehicle's battery pack. Therefore, using A/C can have an impact on the overall range of an electric vehicle. Here's how A/C power usage affects EV range:

Energy Consumption: A/C systems require electricity to operate the compressor, fans, and other components. This draws power from the battery, reducing the amount of energy available for driving the vehicle. The more energy the A/C system consumes, the less energy remains for propelling the vehicle, which can reduce the overall range.

Efficiency: A/C systems can vary in terms of efficiency. Some newer EV models incorporate more energy-efficient A/C systems, which help minimize the impact on range. However, older or less efficient A/C systems may consume more power, leading to a greater reduction in range.

Driving Conditions: The impact of A/C on range can be more pronounced in certain driving conditions. For example, if you are driving in extreme heat or cold, the A/C system may need to work harder to maintain a comfortable temperature, resulting in increased power consumption and reduced range.

Battery Capacity: The size and capacity of the EV's battery pack also play a role. If the battery pack has a larger capacity, it can provide more energy for both driving and operating the A/C system. In such cases, the impact of A/C power usage on range may be relatively less significant compared to EVs with smaller battery packs.

CHAPTER 3: THEORETICAL DESCRIPTION

3.1 APPROACH

Our approach focuses on data-driven ML modeling of the impacts of EV characteristics, battery, and performance specifications on vehicle range. The data used for regression model training and testing includes the specifications of EVs collected from various sources .

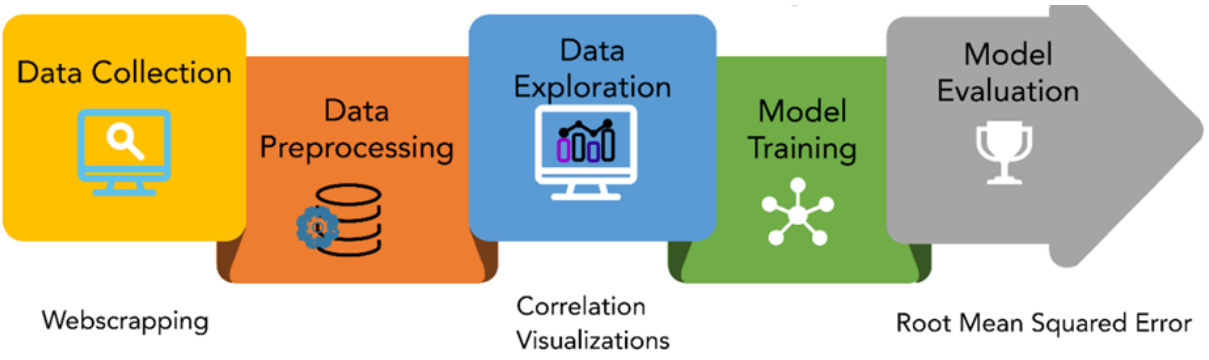
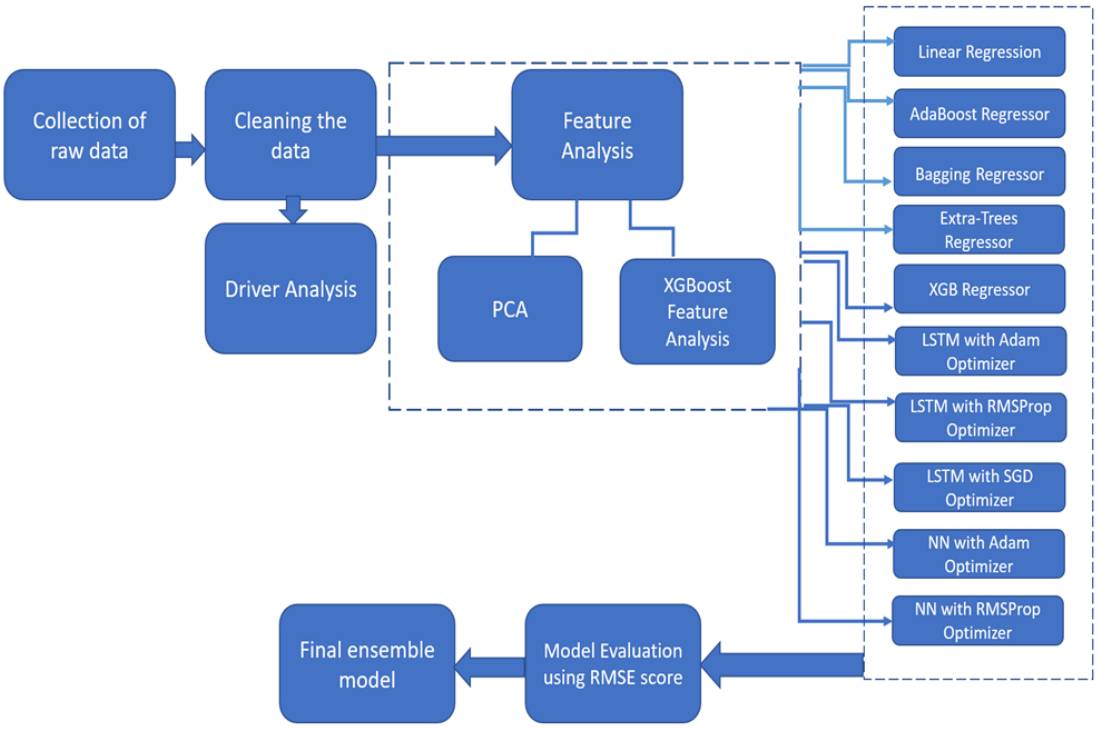


fig flowchart of the solution approach

3.2 BLOCK DIAGRAM

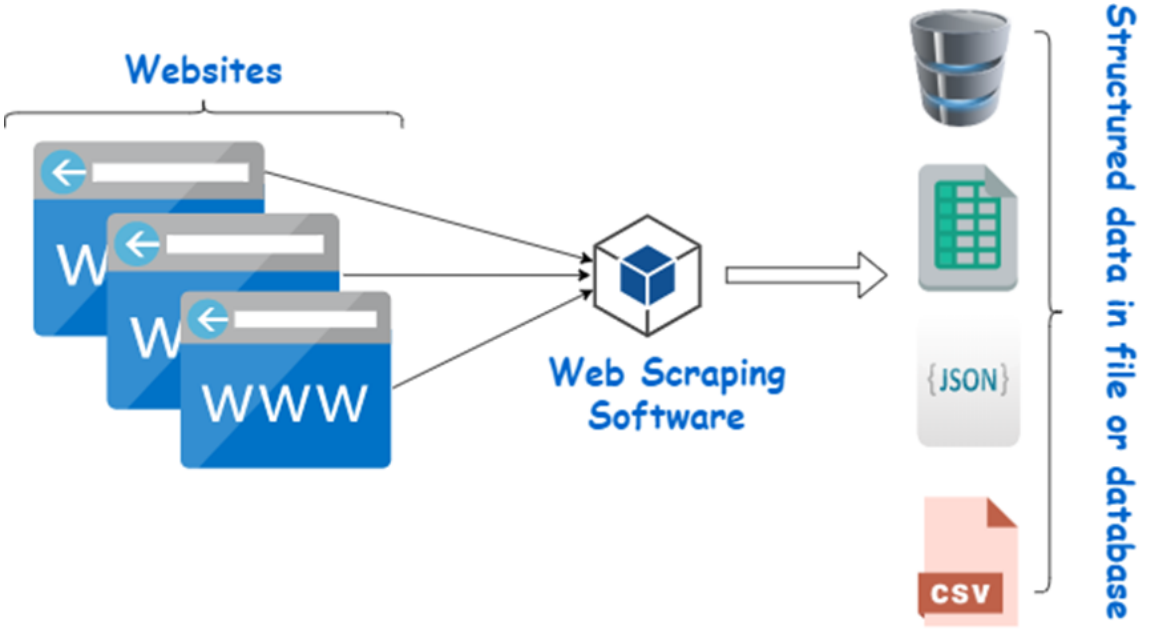


3.3 STEP BY STEP MODEL ANALYSIS

3.3.1 DATA COLLECTION

A detailed dataset of the technical specifications of commercial EV models will be collected through web mining. Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API's or even creating your code for web scraping from scratch.

We will be extracting data using **Beautiful Soup**. Beautiful Soup is an open-source Python library designed for web-scraping HTML and XML files. It is the top Python parser that has been widely used.



Raw data from a CSV file

```
Date/Time,Lat,Long,Elv,Speed,Gids,SOC,AHr,Pack Volts,Pack Amps,Max CP mV,Min CP mV,Avg CP mV,CP mV Diff,Judgment Value,Pack T1 F,Pack T1 C,Pack T2 F,Pack T2 C,Pack T3 F,Pack T3 C,Pack T4 F,Pack T4
C,CP1,CP2,CP3,CP4,CP5,CP6,CP7,CP8,CP9,CP10,CP11,CP12,CP13,CP14,CP15,CP16,CP17,CP18,CP19,CP20,CP21,CP22,CP23,CP24,CP25,CP26,CP27,CP28,CP29,CP30,CP31,CP32,CP33,CP34,CP35,CP36,CP37,CP38,CP39,CP40,CP41,
44,CP45,CP46,CP47,CP48,CP49,CP50,CP51,CP52,CP53,CP54,CP55,CP56,CP57,CP58,CP59,CP60,CP61,CP62,CP63,CP64,CP65,CP66,CP67,CP68,CP69,CP70,CP71,CP72,CP73,CP74,CP75,CP76,CP77,CP78,CP79,CP80,CP81,CP82,CP83,
86,CP87,CP88,CP89,CP90,CP91,CP92,CP93,CP94,CP95,CP96,12v Bat Amps,VIN,Hx,12v Bat Volts,Odo(km),QC,L1/L2,TP-FL,TP-FR,TP-RR,TP-RL,Ambient,SOH,Regenkh,BLevel,epoch time,Motor Pwr(w),Aux Pwr(100w),A/C P
Comp(0.1MPa),Est Pwr A/C(50w),Est Pwr Htr(250w),Plug State,Charge Mode,OBC Out Pwr,Gear,HVolt1,HVolt2,GPS Status,Power SW,BMS,OBC,Debug,Motor Temp,Inverter 2 Temp,Inverter 4 Temp, Speed1, Speed2,Wip
Status,Torque Nm
11/3/2018 20:06,54 58.5341,-1
31.10285,301,0,0,1.20E+07,9895953,401.18,0.709,4186,4176,4179,10,0,47.1,8.4,47.8,8.8,none,none,46.9,8.3,4184,4186,4179,4180,4183,4181,4179,4178,4180,4180,4180,4177,4179,4176,4177,4176,4178,4181,4180
177,4177,4177,4178,4183,4180,4182,4180,4183,4178,4177,4182,4179,4178,4179,4178,4177,4180,4180,4177,4179,4186,4184,4182,4182,4182,4178,-4184,-4177,-4183,-4182,-4182,-4180,-4186,-4178,-4177,-4178,-418
1,-4179,-4177,-4178,-4183,-4186,-4178,-4177,-4177,-4178,-4179,-4177,-4183,-4181,-4181,-4179,-4177,-4179,-4177,-4178,-4178,-4176,-4178,-4177,-4178,-4177,-4177,-4178,-4183,-4184,-4179,-4178,-4182,-418
0,-0.48,SJNFAAZE1U0005576,280,13.04,0,17,23,0,0,0,0,43.7,99.53,0,73,1520798802,0,3,0,0,0,0,0,0,1,401.25,401.25,67F,1,1,1,80,55,40,40,255,0,0D,0
11/3/2018 20:06,54 58.53192,-1
31.09946,298,0,0,1.20E+07,9699345,401.09,1.929,4185,4175,4178,10,0,47.1,8.4,47.8,8.8,none,none,46.9,8.3,4183,4184,4178,4179,4182,4180,4178,4177,4178,4178,4178,4175,4178,4176,4176,4176,4176,4179,4179
176,4176,4175,4176,4183,4179,4180,4180,4182,4176,4176,4179,4178,4177,4177,4177,4176,4179,4178,4176,4178,4185,4183,4180,4180,4181,4176,-4183,-4176,-4182,-4181,-4180,-4179,-4184,-4177,-4175,-4176,-418
9,-4178,-4176,-4177,-4183,-4183,-4177,-4177,-4176,-4177,-4177,-4175,-4182,-4178,-4179,-4178,-4176,-4177,-4177,-4177,-4176,-4176,-4175,-4177,-4175,-4176,-4177,-4176,-4175,-4182,-4183,-4178,-4176,-4180,-417
9,-0.5,SJNFAAZE1U0005576,280,13.04,0,17,23,0,0,0,0,43.7,99.53,0,73,1520798819,0,3,0,0,0,0,0,0,1,401.13,401.13,47F,1,1,1,80,55,40,40,255,0,0D,40.5
11/3/2018 20:07,54 58.53761,-1
```

3.3.2 DATA PREPROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

We will resample the data from second to minute intervals and to keep the time-related gaps in the dataset. We used the Python Pandas module to read our data and to resample our data from second timesteps to minute timesteps.

3.3.3 DATA EXPLORATION

DATA EXPLORATION

Data exploration is a process to analyze data to understand and summarize its main characteristics using statistical and visualization methods To analyse and visualize the data, we will use techniques like Principal Component Analysis(PCA), and various statistical techniques like regressors to establish correlation between various parameters.

Principal Component Analysis of Features:

PCA is used for dimension reduction i.e. it helps in understanding the most important features from the dataset that best explains the variance of the whole dataset. It projects the values of features on some fixed number of dimensions which is also known as PCA Components. Then on the basis of these projected values on the components, it compares the variability of the features in the dataset and computes it as PCA loadings. On the basis of the loadings score, we can estimate the most important features.

3.3.4 MODEL TRAINING AND EVALUATION

The models will be trained using the Linear Regression and SVC classes in Python's SciKit Learn package.

Model Evaluation

The root-mean-squared error (RMSE) was used to evaluate the model and is expressed by

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

where yˆi and yi denote the model predicted and actual values, respectively, and n is the total number of observations/data points.

CHAPTER 4: DETAILED DESCRIPTION OF THE MACHINE LEARNING ALGORITHMS USED

LINEAR REGRESSION

Linear regression is a statistical method that is used to model the relationship between a dependent variable and one or more independent variables. The dependent variable is the variable that is being predicted, and the independent variables are the variables that are used to predict the dependent variable.

Linear regression assumes that the relationship between the dependent variable and the independent variables is linear. This means that the dependent variable can be represented as a linear combination of the independent variables.

The mathematical model for linear regression is a linear equation that predicts the value of a dependent variable, y, from the values of one or more independent variables, x. The equation is:

$y = b0 + b1x1 + b2x2 + ... + bnxn$
where:

- b0 is the y-intercept, which is the value of y when all of the independent variables are equal to 0.
- b1, b2, ..., bn are the slope coefficients, which represent the amount that y changes for a one-unit change in x1, x2, ..., xn.

ADABOOST

AdaBoost, short for Adaptive Boosting, is a machine learning algorithm that combines multiple weak learners to create a strong learner. A weak learner is a classifier that is only

slightly better than random guessing. AdaBoost works by iteratively training weak learners on weighted versions of the training data. The weights are assigned such that instances that are misclassified by previous learners are given more weight. The output of each weak learner is then combined into a weighted sum, with the weights of the learners being adjusted based on their performance. The final output of the AdaBoost classifier is the sign of the weighted sum.

The mathematical model of the AdaBoost algorithm can be described as follows:

1. Let D be the training dataset, and let yi be the label of the ith data point.
2. Initialize the weights of all data points to be equal: wi=n1, where n is the number of data points.
3. For m=1,2,…,M:
    1. Train a weak learner Hm using the weighted training dataset D.
    2. Calculate the error of the weak learner on the weighted training dataset: em=n∑i=1nwiI(yi=Hm(xi)), where I is the indicator function.
    3. Calculate the coefficient of the weak learner: αm=21logem1−em.
    4. Update the weights of the data points: wi=wi⋅exp(−αmyiHm(xi)).
4. The final prediction of the AdaBoost algorithm is given by:

$$f(x) = \sum_{m=1}^{M} \alpha_m H_m(x)$$

The AdaBoost algorithm works by iteratively training a series of weak learners, and then combining them to form a strong learner. The weak learners are trained on a weighted training dataset, where the weights of the data points are inversely proportional to their error. This means that the data points that are most difficult to classify are given more weight. The coefficients of the weak learners are then calculated using a technique called exponential weighting. The final prediction of the AdaBoost algorithm is a weighted sum of the predictions of the weak learners.

AdaBoost is a powerful machine learning algorithm that can be used for a variety of

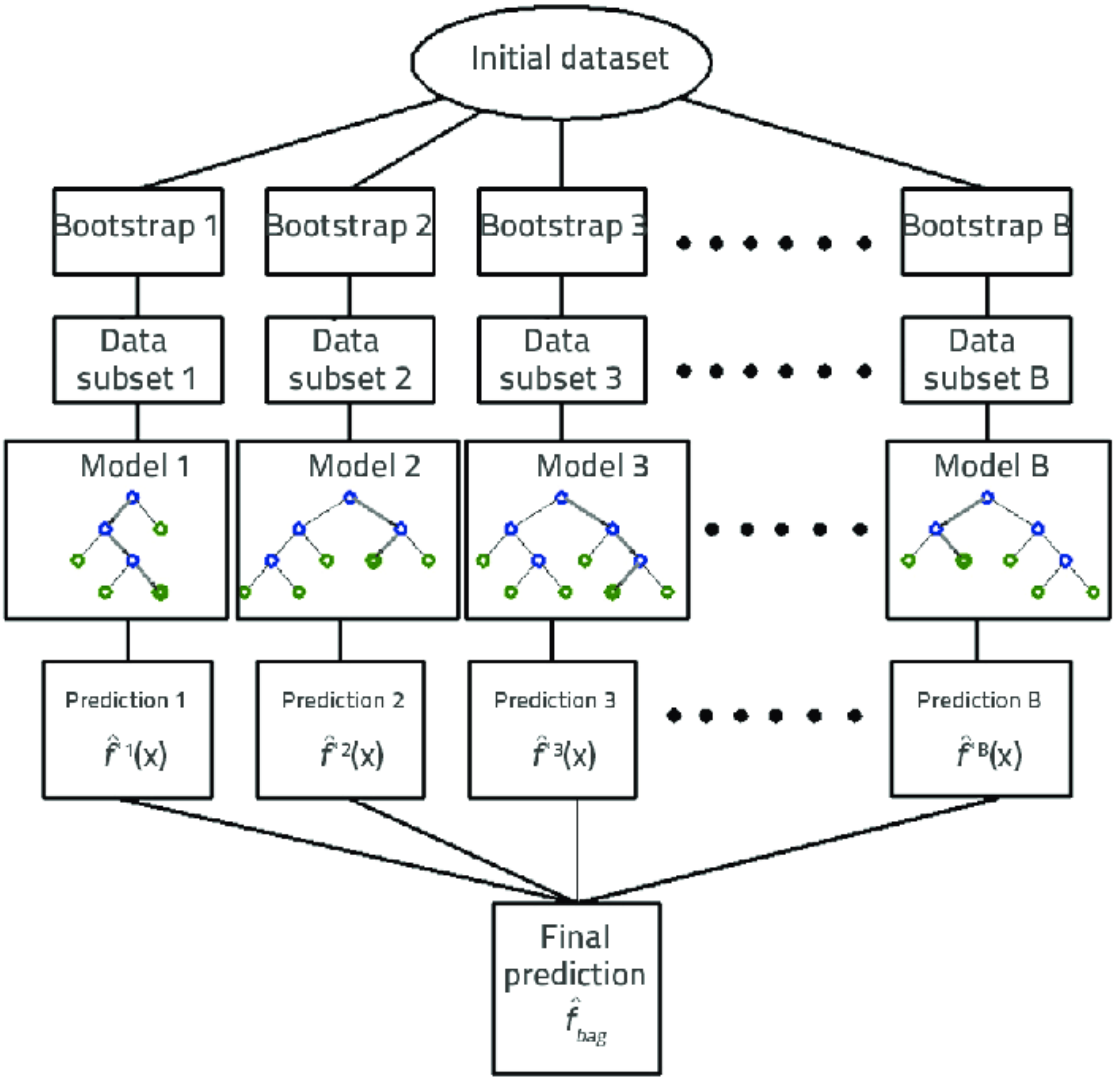
tasks, including classification, regression, and forecasting. It is a relatively simple algorithm to understand and implement, and it has been shown to be effective on a wide range of datasets.

BAGGING REGRESSION

Bagging regression is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.

Bagging works by creating multiple versions of a model, called "models", each trained on a different bootstrap sample of the original dataset. A bootstrap sample is a dataset created by sampling with replacement from the original dataset. This means that some data points may be included in multiple bootstrap samples.

The predictions of the individual models are then averaged to produce a final prediction. This averaging helps to reduce the variance of the final prediction and makes it less likely to overfit the training data.



The mathematical model of bagging regression can be described as follows:

Let D be the training dataset, and let yi be the target value of the ith data point.

Let M be the number of models to be built.

For m=1,2,…,M:

1. Bootstrap a sample Dm from D.
2. Build a model hm using Dm.

The final prediction of the bagging regressor is given by:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} h_m(x)$$

where hm(x) is the prediction of the mth model for the input x.

Bagging regression is a powerful machine learning algorithm that can be used for a variety of regression tasks. It is a relatively simple algorithm to understand and implement, and it has been shown to be effective on a wide range of datasets.

Here are some of the advantages of bagging regression:

- It can reduce variance and help to avoid overfitting.
- It can improve the stability of the model.
- It can be used with a variety of machine learning algorithms.

Here are some of the disadvantages of bagging regression:

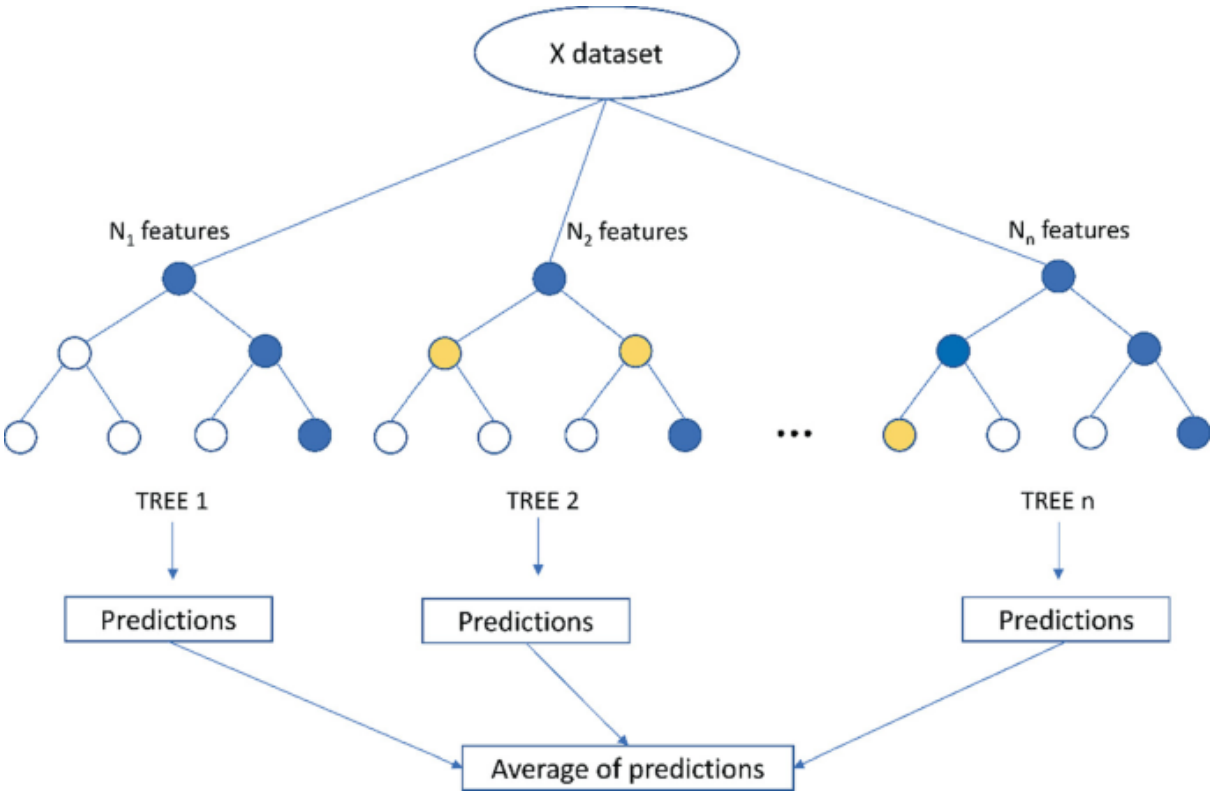
- It can increase the computational complexity of the model.
- It can reduce the accuracy of the model if the individual models are not accurate.
- It can be difficult to tune the hyperparameters of the model.

EXTRA TREES REGRESSOR

Extra-trees is a type of decision tree ensemble model that is built using a bootstrapped sample of the training data. Each tree in the ensemble is trained using a random subset of features and a random subset of the training data. This process of randomly subsampling the data and features helps to reduce the variance of the model and improve its generalization performance.

Extra-trees is a fast and scalable algorithm that can be used to solve a variety of classification

and regression problems. It is particularly well-suited for problems with a large number of features.



The mathematical model of the Extra Trees regressor can be described as follows:

1. Let D be the training dataset, and let yi be the target value of the ith data point.
2. Initialize the number of trees to be built: n estimators.
3. For m=1,2,…,n estimators:
    1. Build a decision tree Tm using the training dataset D.
    2. At each node of the decision tree, randomly select a subset of features to consider.
    3. At each node of the decision tree, randomly select a split point from the selected features.
4. The final prediction of the Extra Trees regressor is given by:

$$f(x) = \frac{1}{n_estimators} \sum_{m=1}^{n_estimators} \hat{y}_m(x)$$

where y^m(x) is the prediction of the mth decision tree for the input x.

The Extra Trees regressor is a type of ensemble learning algorithm that builds a forest of decision trees. The decision trees are built using a random subset of features and a random split point at each node. This randomization helps to prevent the trees from overfitting the training data. The final prediction of the Extra Trees regressor is a weighted average of the predictions of the individual trees.

The Extra Trees regressor is a powerful machine learning algorithm that can be used for a variety of regression tasks. It is a relatively simple algorithm to understand and implement, and it has been shown to be effective on a wide range of datasets.

XGB REGRESSOR

XGBoost is a tree-based ensemble learning algorithm that uses a technique called gradient boosting to build a strong predictive model. Gradient boosting is an iterative process that starts with a weak learner and then adds new learners to the model in a way that minimizes the loss function.

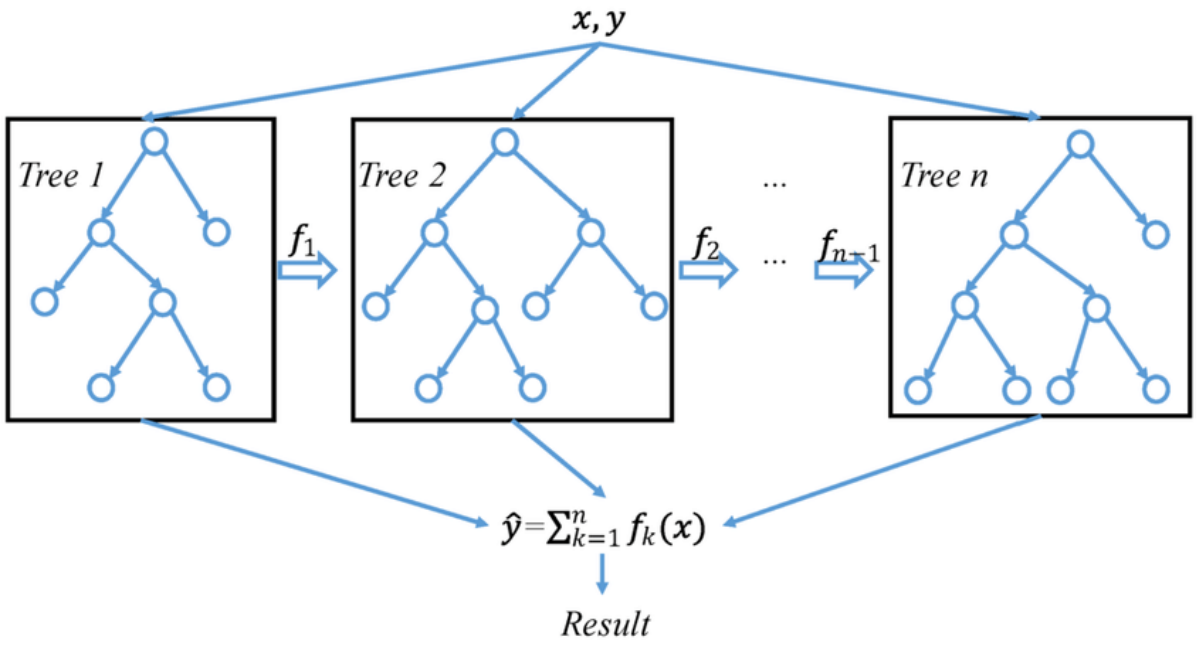
The mathematical model of XGBoost can be described as follows:

Let D be the training dataset, and let yi be the target value of the ith data point.

Let M be the number of trees to be built.

For m=1,2,…,M:

1. Build a decision tree Tm using the training dataset D.
2. Calculate the gradient of the loss function with respect to the prediction of the mth tree.
3. Update the weights of the data points using the gradient.
4. The final prediction of the XGB regressor is given by:



$$\hat{y}=\sum_{k=1}^{n} f_k(x)$$

where y^m(x) is the prediction of the mth decision tree for the input x.

The XGB regressor is a powerful machine learning algorithm that can be used for a variety of regression tasks. It is a relatively simple algorithm to understand and implement, and it has been shown to be effective on a wide range of datasets.

LSTM

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) that is commonly used for natural language processing (NLP) tasks. LSTMs are able to learn long-term dependencies between words in a sentence, which makes them well-suited for tasks such as text classification, sentiment analysis, and machine translation.

The mathematical model of an LSTM can be described as follows:

Let xt be the input at time t, ht be the hidden state at time t, and yt be the output at time t.

The LSTM cell has four gates:

- Forget gate: $\sigma(Wfxt+Ufht-1+bf)$
- Input gate: $\sigma(Wixt+Uiht-1+bi)$
- Output gate: $\sigma(Woxt+Uoht-1+bo)$
- Candidate state: $\tanh(Wcxt+Ucht-1+bc)$

The forget gate controls how much of the previous hidden state is forgotten. The input gate controls how much of the current input is added to the hidden state. The output gate controls how much of the hidden state is outputted. The candidate state is a temporary state that is used to update the hidden state.

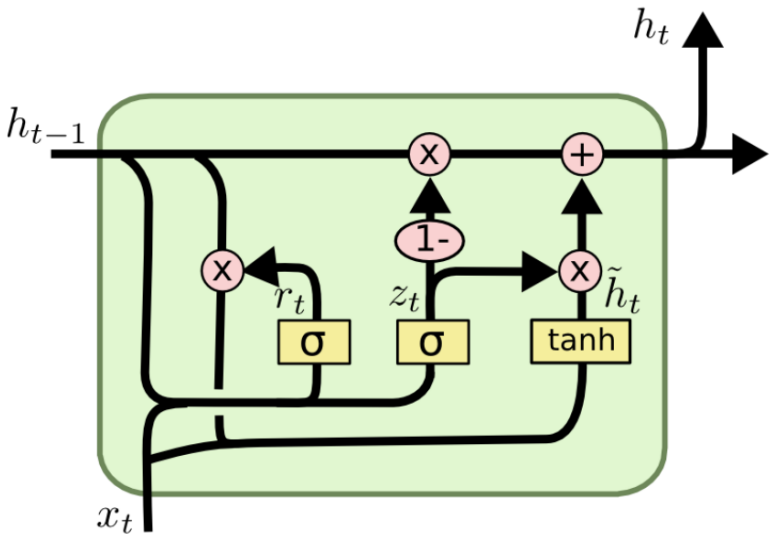
The hidden state at time t is updated as follows:

$ht=LSTM(xt,ht-1)=f(Wfxt+Ufht-1+bf)\odot ht-1+i(Wixt+Uiht-1+bi)\odot ct+o(Woxt+Uoht-1+bo)\odot \tanh(Wcxt+Ucht-1+bc)$

where f is the forget gate, i is the input gate, o is the output gate, $\odot$ is the element-wise product, and tanh is the hyperbolic tangent function.

The output at time t is calculated as follows:

$yt=o(Woxt+Uoht-1+bo)\odot \tanh(ht)$



Structure of LSTM model

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

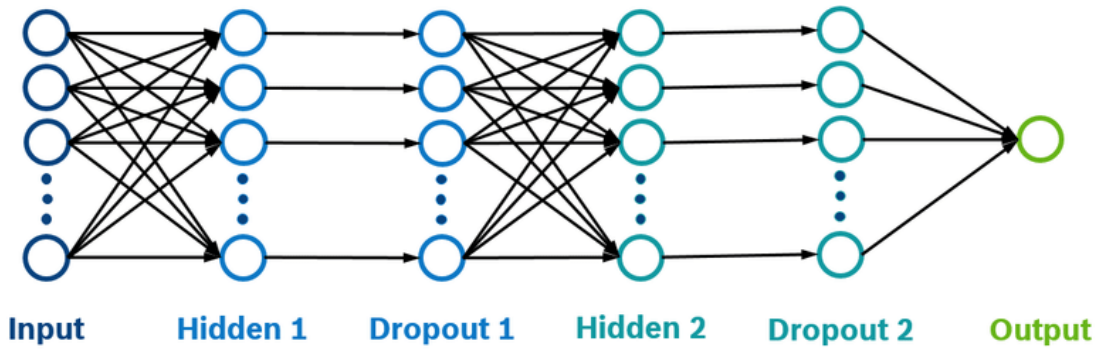
The LSTM cell is repeated for each time step in the input sequence. The final output of the LSTM is a sequence of predictions, one for each time step in the input sequence.

LSTMs are a powerful tool for NLP tasks. They are able to learn long-term dependencies between words in a sentence, which makes them well-suited for tasks such as text classification, sentiment analysis, and machine translation. However, LSTMs can be computationally expensive to train, and they can be sensitive to the quality of the training data.

NEURAL NETWORK

A neural network is a machine learning model that is inspired by the human brain. It is made

up of a network of interconnected nodes, called neurons. Each neuron takes in a number of inputs, performs a mathematical operation on them, and then outputs a single value. The outputs of the neurons in one layer are the inputs to the neurons in the next layer.



The mathematical model of a neural network can be described as follows:

Let $x1,x2,...,xn$ be the inputs to the network,

$w1,w2,...,wn$ be the weights of the connections between the neurons, and

b be the bias of the neuron. The output of the neuron is calculated as follows:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where σ is the activation function.

The activation function is a non-linear function that helps the neural network to learn complex relationships between the inputs and the outputs. Some common activation functions include the sigmoid function, the tanh function, and the rectified linear unit (ReLU) function.

The output of the final layer of the neural network is the prediction of the model. The neural network is trained by adjusting the weights and biases of the neurons so that the model makes accurate predictions on a training dataset.

Neural networks are a powerful tool for machine learning. They can be used to solve a wide variety of problems, including classification, regression, and forecasting. However, neural networks can be computationally expensive to train, and they can be sensitive to the quality of the training data.

CHAPTER 5 : IMPLEMENTATION

Software Specifications

We have used python as our programming language. We have used powershell of the Anaconda.

Following are the libraries used:

Pandas: Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal.

SciKit: Scikit-learn is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem. Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling. Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, and statistical tools for analyzing these models.

Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

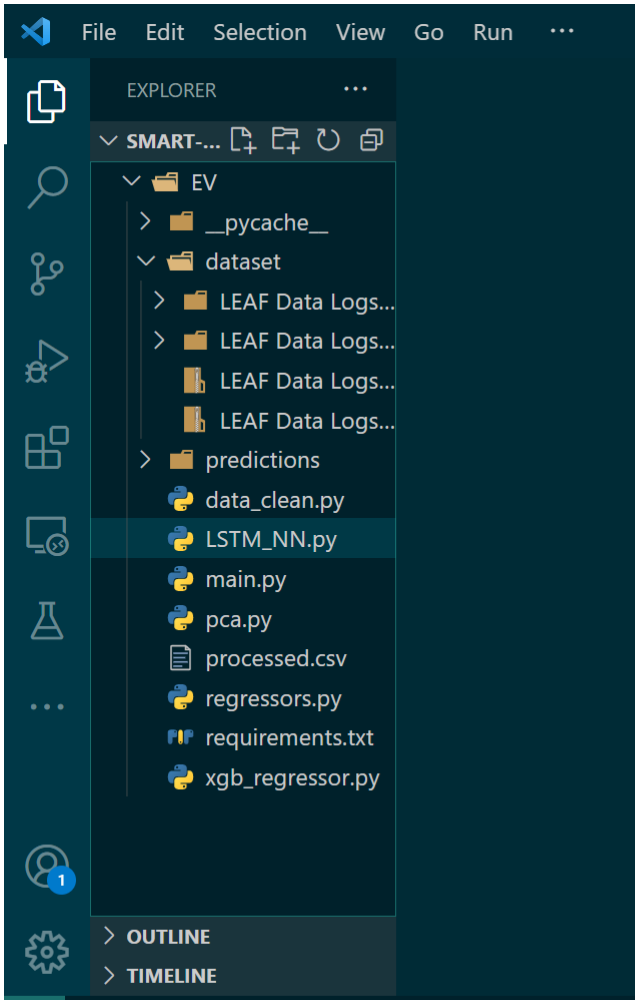
Tensorflow: TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Keras:Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation

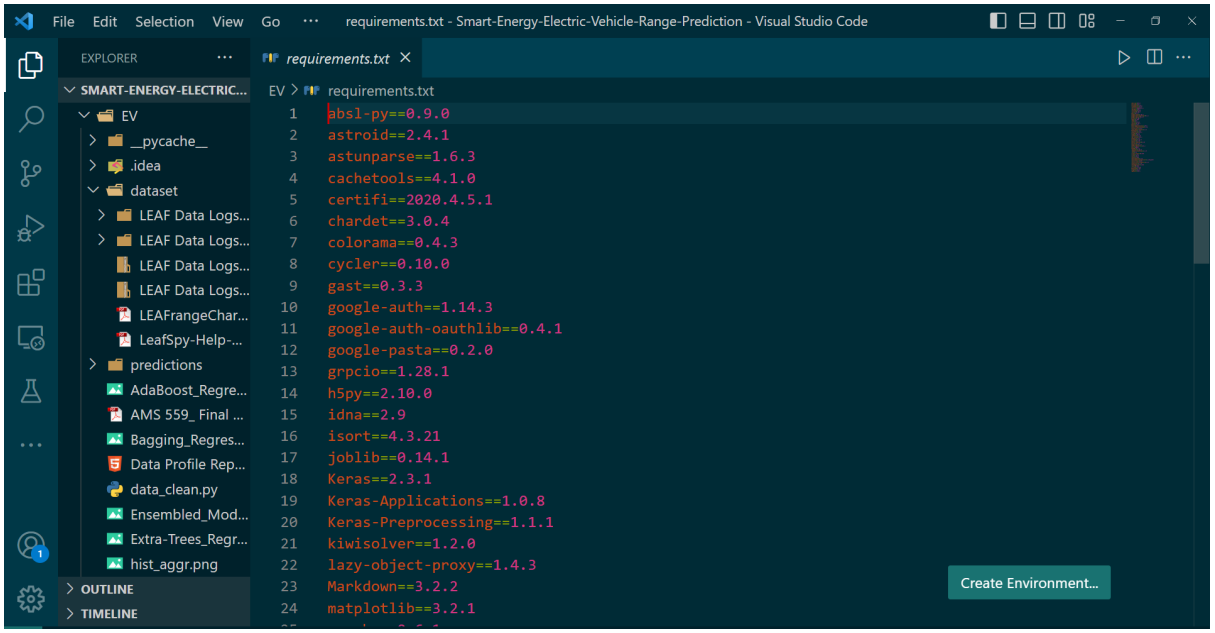
Numpy: NumPy is a Python library used for working with arrays. It also has functions for

working in the domain of linear algebra, fourier transform, and matrices.

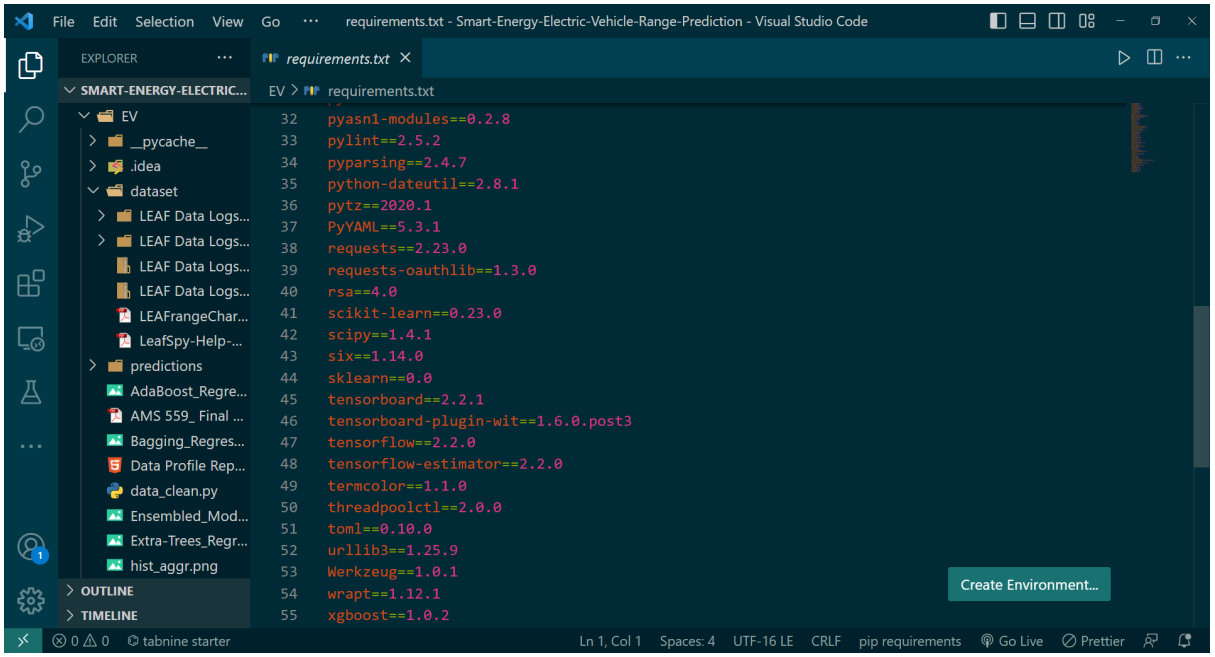
Project Folder Structure



Requirements files



.



DATA_CLEAN.PY

- The important columns are defined in the important_columns variable. These are the columns that will be kept in the final DataFrame.
- The epoch time column is converted to a datetime object using the

datetime.fromtimestamp() function.

- The Gids, SOC, and AHr columns are converted to floats using the float() function.
- The energy level column is calculated by multiplying the Gids, SOC, and AHr columns together.
- The Ambient column is converted to degrees Celsius by subtracting 32 from the value and then multiplying by 5 / 9.
- The Motor Pwr(w), Aux Pwr(100w), A/C Pwr(250w), Est Pwr A/C(50w), and Est Pwr Htr(250w) columns are converted to watts by multiplying the value by the corresponding unit.
- The Motor Temp column is subtracted by 40.
- The Torque Nm column is taken as the absolute value.
- The important columns are kept in the final DataFrame by using the data[important_columns] expression.
- The epoch time column is set as the index of the DataFrame using the data.set_index('epoch time') expression.
- The DataFrame is sorted by the epoch time column using the data.sort_values(by=['epoch time'], ascending=True) expression.
- The DataFrame is resampled to 1-minute intervals using the data.resample("1T",).mean() expression.
- Any missing values are filled in using the data.fillna() expression.
- Any rows where the SOC is greater than 100 are dropped using the data = data[data['SOC'] <= 100] expression.
- The distance traveled and charge drop columns are calculated using the (data["Odo(km)"] - data["Odo(km)"].shift(1)).values and (data["SOC"] - data["SOC"].shift(1)).values expressions, respectively.
- The range column is calculated using the mil_range = [] expression.
- The master DataFrame is created by appending the DataFrame to an empty DataFrame using the dataframe = dataframe.append(data) expression.
- The master DataFrame is sorted by the range column using the data = data[data['range'] <= 500] expression.
- The Percentile_rank column is dropped from the master DataFrame using the data = data.drop(columns=['Percentile_rank']) expression.
- The master DataFrame is written to a CSV file using the

data.loc[:,:].to_csv("processed.csv") expression.

PCA.PY

The pca() function is defined to perform PCA on the dataset. It takes the dataset as input, initializes a PCA object with n_components=26 (the number of principal components to keep), and applies PCA using fit_transform(). The explained variance ratios and cumulative explained variance ratios are calculated using the explained_variance_ratio_ and cumsum() functions, respectively. The transformed principal components are stored in a DataFrame and converted to a list. Finally, the function returns the principal components, explained variance ratios, cumulative explained variance ratios, and the PCA object.

A bar plot of the explained variance ratios is created using plt.bar(). The x-axis represents the PCA components (from 0 to 25) and the y-axis represents the explained variance ratios. The cumulative explained variance ratios are plotted as a line graph using plt.plot()

The top5_features() function is defined to determine the top 5 features with the highest PCA loading scores. PCA loadings are calculated using the formula loadings = pca.components_.T * np.sqrt(pca.explained_variance_). The loadings are stored in a DataFrame with feature names as indices. The sum of loadings for each feature is calculated and added as a new column to the DataFrame. The DataFrame is sorted in descending order based on the sum of loadings. The function returns a list of the top 5 features and the DataFrame.

REGRESSORS.PY

Defining utility functions:

- plot_performance is a function that plots the true values and predicted values against the epoch time for a given model type.
- Calc_RMSE is a function that calculates the root mean squared error between the true values and predicted values.
- split_dataset_y is a function that splits the dataset and target variable into training and testing sets.
- prediction is a function that makes predictions using a given model and returns the root mean squared error and predicted values.

Model-specific functions:

- linear_reg is a function that performs linear regression modeling on the dataset.
- AdaBoostRegressor_model is a function that performs AdaBoost regression modeling on the dataset.
- BaggingRegressor_model is a function that performs Bagging regression modeling on the dataset.
- ExtraTreesRegressor_model is a function that performs Extra Trees regression modeling on the dataset.

Main execution:

- The main code block executes when the script is run directly (not imported as a module).
- _, RMSE_LR, y_pred_LR = linear_reg(dataset) performs linear regression modeling, returning the trained model, RMSE (root mean squared error), and predicted values.
- _, RMSE_AdB, y_pred_AdB = AdaBoostRegressor_model(dataset) performs AdaBoost regression modeling, returning the best model, RMSE, and predicted values.
- _, RMSE_BG, y_pred_BG = BaggingRegressor_model(dataset) performs Bagging regression modeling, returning the best model, RMSE, and predicted values.
- _, RMSE_ET, y_pred_ET = ExtraTreesRegressor_model(dataset) performs Extra Trees regression modeling, returning the best model, RMSE, and predicted values.
- Predicted values are saved as CSV files for each model type.

XGB.PY

`xgb_model` function: This function defines the XGBoost model, trains it, makes predictions, and calculates the RMSE. Let's break it down:

a. Splitting the data: The input data is split into features (`x`) and the target variable (`y`) using `iloc` indexing.

b. Splitting the data into training and testing sets: The `train_test_split` function is used to split the data into 80% training and 20% testing sets. The data is not shuffled (`shuffle=False`) to preserve the temporal order.

c. Creating DMatrix: The training and testing data are converted into `xgb.DMatrix` format, which is required for training the XGBoost model.

d. Defining model parameters: The `params` dictionary contains various parameters for the XGBoost model, such as `colsample_bytree`, `eta` (learning rate), `max_depth`, `min_child_weight`, `objective`, and `subsample`. The evaluation metric is set to RMSE.

e. Training the model: The `xgb.train` function is used to train the XGBoost model. It uses early stopping with a maximum of 10 rounds of no improvement in validation RMSE.

f. Making predictions: The trained model is used to make predictions on the testing set (`X_test`).

LSTM_NN.PY

The `read_csv` function is used to read the 'processed.csv' file into a DataFrame named `dataset`.

The `series_to_supervised` function is defined to transform the time series data into a supervised learning problem. It takes the data, the number of lag observations as input (`n_in`), the number of future observations to predict (`n_out`), and a boolean flag to drop rows with NaN values (`dropnan`). It creates lagged observations of the input data and forms

a new DataFrame with shifted columns. It returns the transformed DataFrame.

The `LSTM_ML` function is defined to build and train an LSTM (Long Short-Term Memory) model for time series prediction. It takes the `dataset` as input. The function performs the following steps:

- The `values` variable is assigned the numpy array representation of the dataset.
- The `MinMaxScaler` is used to scale the values between 0 and 1.
- The `series_to_supervised` function is called to transform the scaled data into a supervised learning problem. The columns representing the current time step are dropped except for the target variable column.
- The transformed data is split into training and testing sets.
- The input sequences are reshaped into the required format for LSTM input.
- An LSTM model is created using the Sequential API from Keras. It consists of an LSTM layer with 50 units and a dense output layer.
- The model is compiled with mean absolute error (MAE) as the loss function and Adam optimizer.
- The model is trained on the training data for 50 epochs with a batch size of 72. Validation data is used for evaluating the performance during training.
- The trained model is used to make predictions on the test data.
- The predictions are inverse transformed to the original scale using the `scaler` object.
- The root mean squared error (RMSE) is calculated between the actual and predicted values.
- The actual and predicted values are plotted using Matplotlib and saved as 'temp2.png'.
- The trained model, predicted values, and RMSE are returned.

The `model`, `inv_yhat`, and `rmse` variables are assigned the return values of the `LSTM_ML` function.

The `neural` function is defined to build and train a neural network model for time series prediction. It takes the `dataset` as input. The function performs the following steps:

- The input features and target variable are separated from the `dataset`.

- The data is split into training and testing sets.
- A neural network model is created using the Sequential API from Keras. It consists of dense layers with 200 units and ReLU activation functions. Dropout regularization is applied to reduce overfitting.
- The model is compiled with mean squared error (MSE) as the loss function and Adam optimizer.
- Early stopping is implemented with a patience of 3 to monitor validation loss and stop training if it doesn't improve.
- The model is trained on the training data for 30 epochs with a validation split of 0.2.
- The trained model is used to make predictions on the test data.
- The actual and predicted values are plotted using Matplotlib and saved as 'temp3.png'.
- The root mean squared error (RMSE) is calculated between the actual and predicted values.
- The trained model, predicted values, and RMSE are returned.

The `model`, `y_pred`, and `rmse` variables are assigned the return

MAIN.PY

The following actions are performed in the main.py file

1. Import necessary libraries and modules:
    - `regressors`: A custom module that contains functions for data preprocessing and performance evaluation of regression models.
    - `xgb_regressor`: A custom module that likely contains the implementation of the XGBoost regressor.
    - `pandas`, `numpy`, `matplotlib.pyplot`: Standard libraries for data manipulation, numerical computations, and data visualization.
    - `mean_squared_error` and `sqrt` from `sklearn.metrics`: Functions for calculating mean squared error and square root, respectively.
2. Read the dataset from a CSV file and preprocess it:

- The dataset is read from the "processed.csv" file, with the index column set as 'epoch time'.
- The index column is converted to datetime format.
- The 'range' column is stored separately in the `y_range` variable.
- The 'range' column is dropped from the dataset.

3. Read the predictions from different models:
   - Predictions from various models (e.g., Linear Regression, AdBoost Regression, Bagging Regression, XGBoost Regression, LSTM with different optimizers, and Neural Network with different optimizers) are read from separate CSV files.
   - The predictions are stored in different variables (`y_pred_LR`, `y_pred_AdB`, etc.).
4. Calculate RMSE for each model:
   - RMSE values are calculated for each model using the `Calc_RMSE` function from the `regressors` module.
   - The RMSE values are stored in a dictionary named `RMSE_ALL`.
5. Print RMSE values for each model:
   - The RMSE values for each model are printed using the `print` function.
6. Define weights for an ensemble model:
   - Weights are defined as an array of values ranging from 0 to 1 with a step of 0.1.
   - These weights will be used to create an ensemble of different models.
7. Iterate over the weights to find the best ensemble model:
   - A nested loop iterates over the weights to find the combination of weights that gives the minimum RMSE.
   - If the sum of weights is less than or equal to 1, a new prediction is calculated by combining the predictions from XGBoost, Bagging, and LSTM models using the defined weights.
   - The RMSE is calculated for the new ensemble prediction.
   - The RMSE and weights for the best ensemble model are printed.
   - The best ensemble prediction is stored in the `best_y_pred` variable.
8. Calculate RMSE for the best ensemble model:

    - The RMSE for the best ensemble model is calculated using the `Calc_RMSE` function from the `regressors` module.
    - The RMSE value is added to the `RMSE_ALL` dictionary with the key "Ensembled".
9. Plot the performance of the best ensemble model:
    - The `plot_performance` function from the `regressors` module is called to plot the performance of the best ensemble model.
    - The test features, true range values, and the predicted range values are passed to the function.
10. Sort the RMSE values in descending order and plot them:

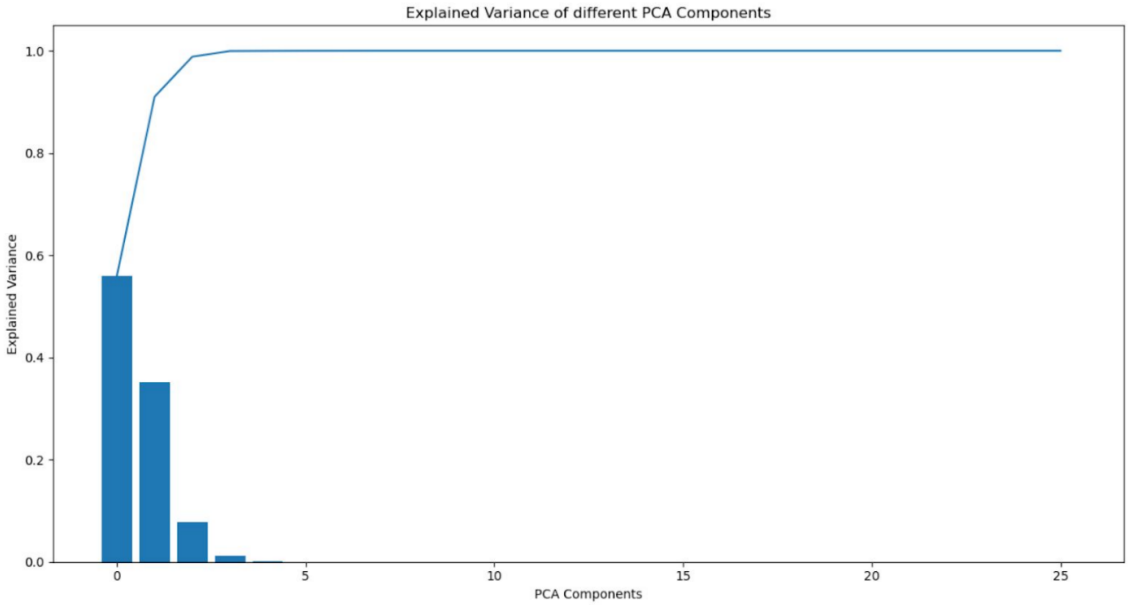
- The `RMSE_ALL` dictionary is sorted in descending order based on the RMSE values using the `sorted` function.
- A bar plot is created using `matplotlib.pyplot` to visualize the RMSE scores of different models.
- The plot is saved as a file named "RMSE_Scores.png" and displayed using `plt.show()`.

CHAPTER 6- ANALYSIS AND PREDICTIONS

PCA

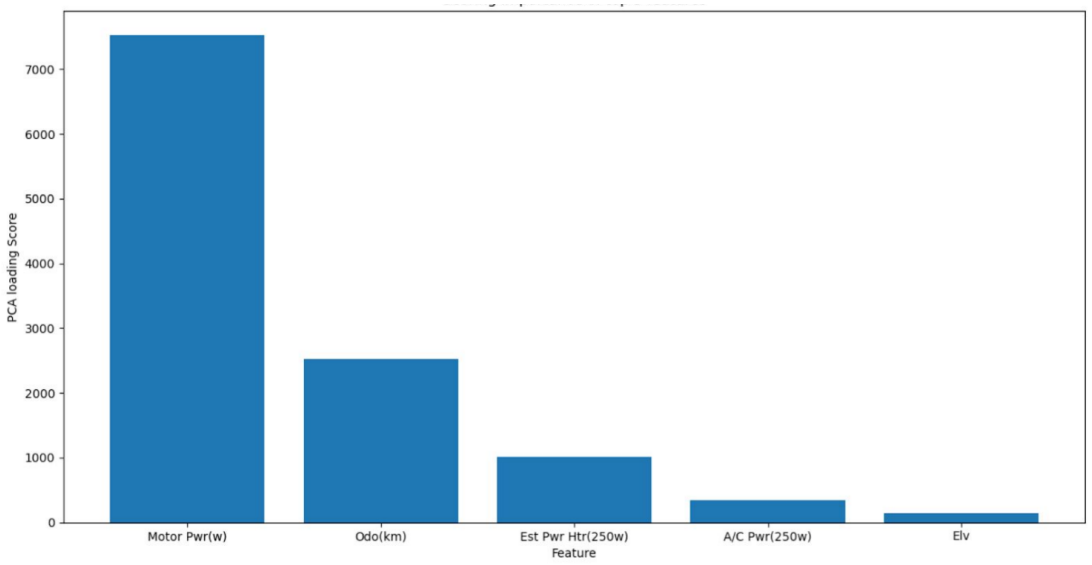
PCA helps in understanding the most important features from the dataset that best explains the variance of the whole dataset. It projects the values of features on some fixed number of dimensions which is also known as PCA Components. Then on the basis of these projected values on the components, it compares the variability of the features in the dataset and computes it as PCA loadings. On the basis of the loadings score, we can estimate the most important features.

We implemented the PCA on the dataset and drew the following scree plot to understand the behavior of the components:



● We can observe from the line plot of cumulative explained variance that 75% of the variability of the dataset can be implemented using the first two to three PCA components. On the basis of these components, we calculated the PCA loadings score of all the features and selected out the top 5 features that explain the variability of the whole dataset.

● Those features or columns are: 'Motor Pwr(w)': Drive motor power, 'Odo(km)': Odometer reading, 'Est Pwr Htr(250w)': Estimated Cabin PTC Heater power in 250-watt units, 'A/C Pwr(250w)': Power used by the Air Conditioning System power in 250-watt units, and 'Elv': Elevation which is received from the GPS hardware in the Android device. The Scoring importance of these top 5 features is shown below:

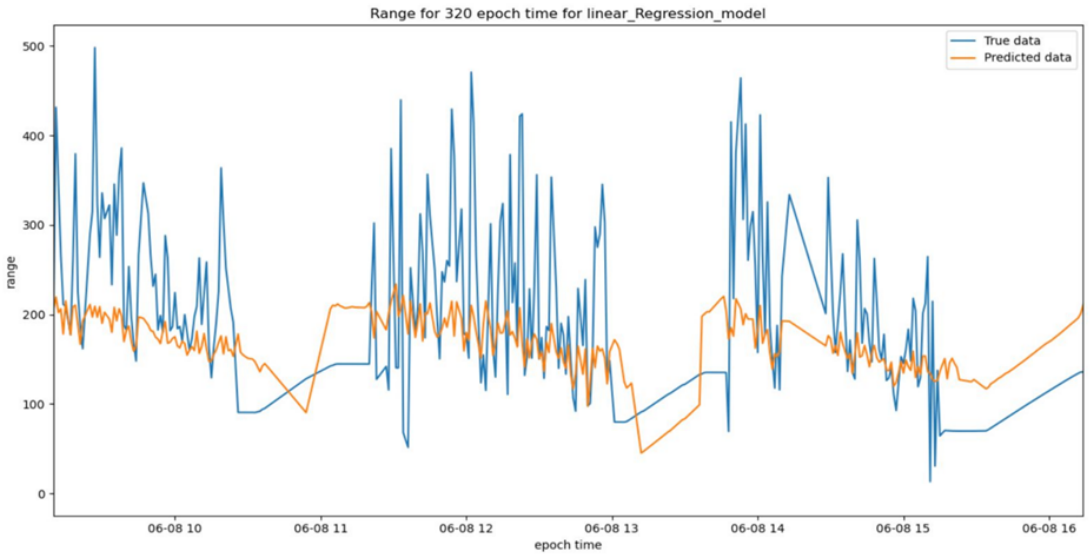


● We can estimate that the first feature(Motor Pwr) is strongly correlated to the rest of the features while showing a PCA loading score of around 7000. After that Odometer reading of the distance traveled is showing good correlation and so on.

● By understanding the Principal Component Analysis of our data, we can presumably say that these features are mostly going to explain the range of the Electric Vehicle.

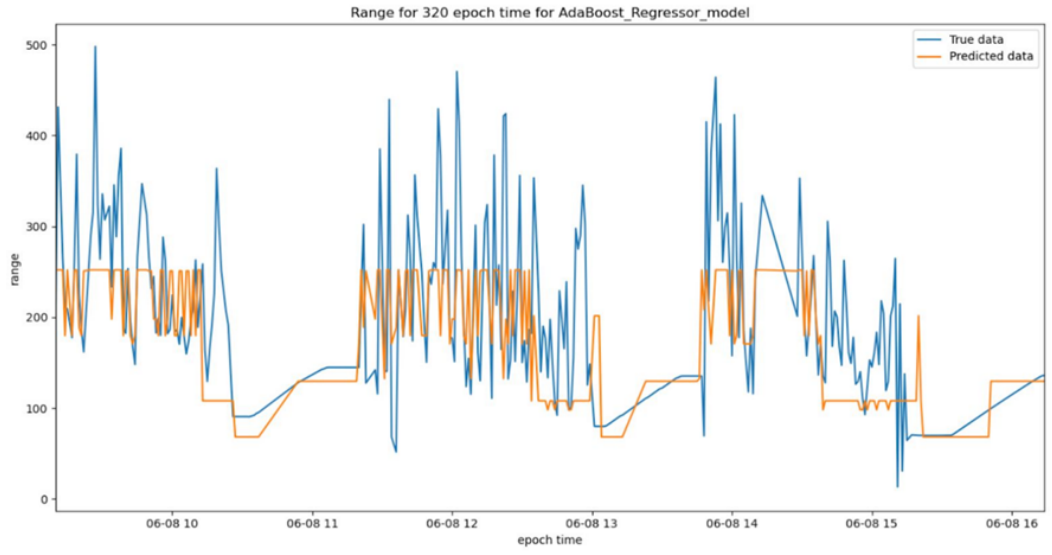
Linear Regression

We started with fitting our dataset on a Linear Regression Model, which tries to fit the data on a line. For linear Regression, we got an RMSE score of 81.78280563366944.



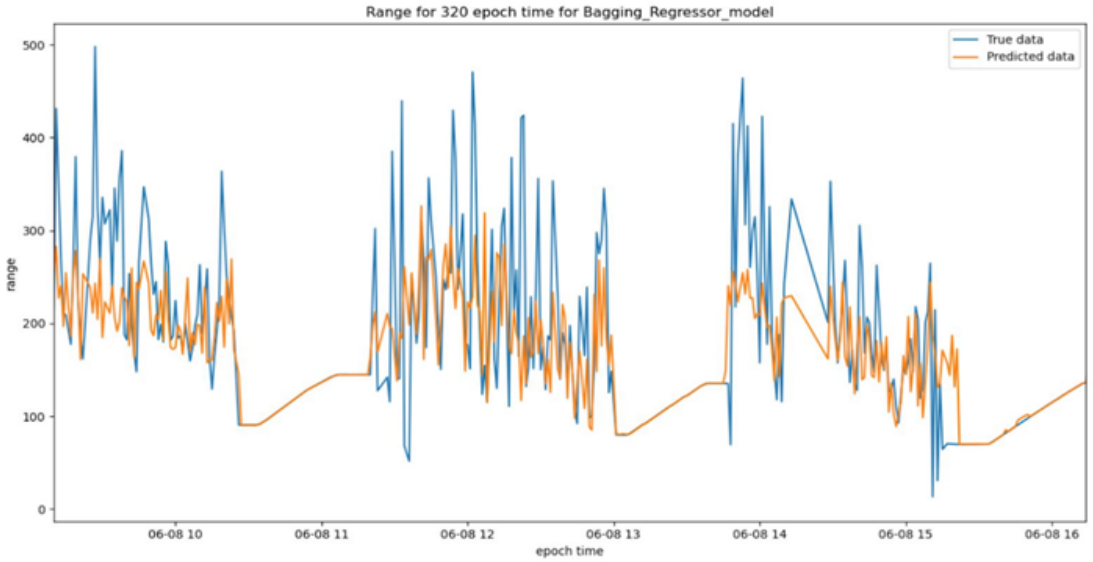
AdaBoost Regressor Model

Next, we have used the AdaBoost Regressor Model to fit our data, which is a meta-estimator. It fits a regressor on the dataset and then again fits the copy of that regressor on the dataset but with adjustment of weight of instances on the basis of errors. For AdaBoost Regression, we got an RMSE score of 75.63687847244836 which is better than Linear Regression.



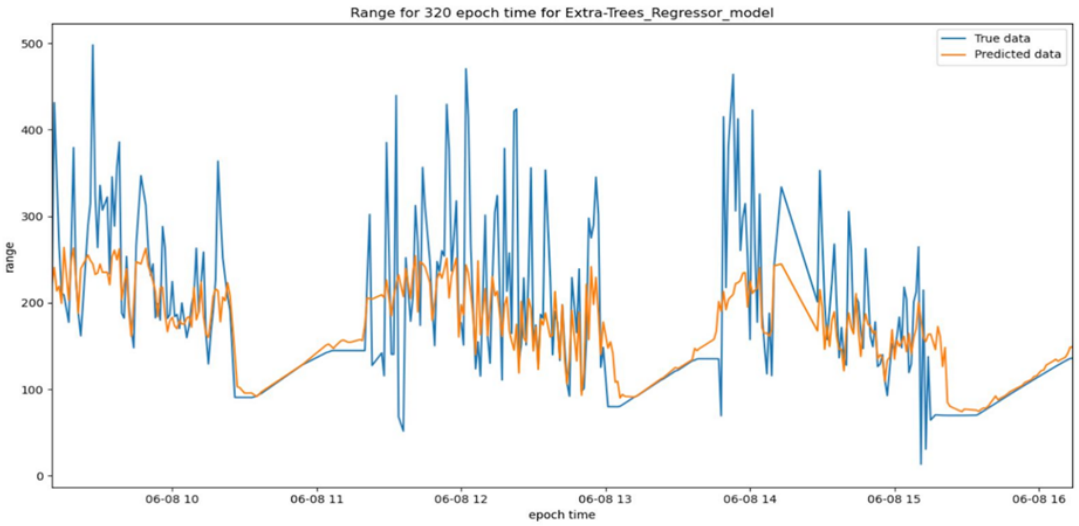
Bagging Regressor Model

The Bagging Regressor Model is also a meta-estimator like AdaBoost Regressor but it fits the regressors on random subsets of the original data and chooses the regressor by voting. We got the RMSE Score of 70.94967401057501 for this model.

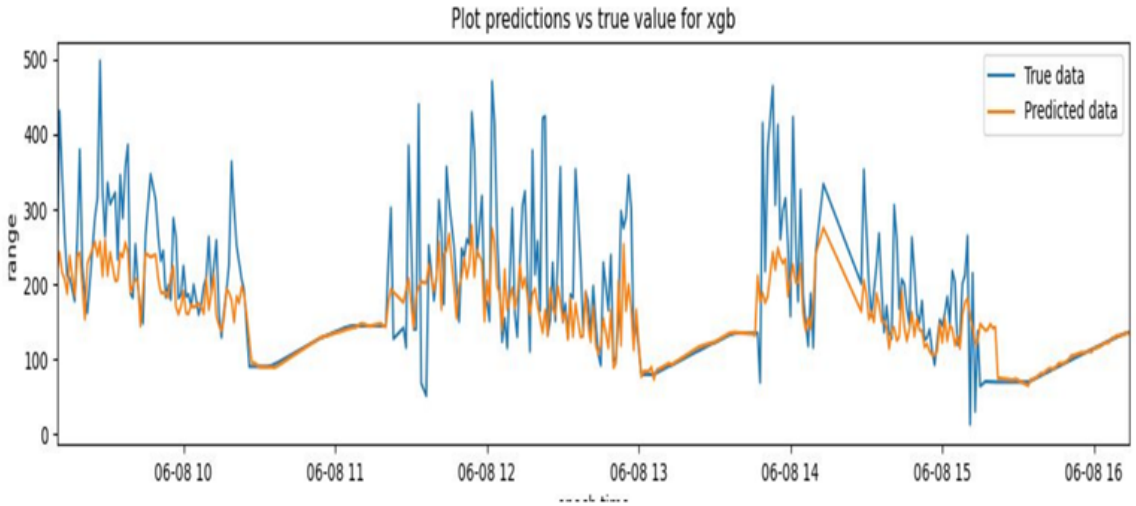


Extra Trees Regressor Model

This model fits a number of randomized decision trees on various sub-samples of the dataset. It uses averaging to prevent over-fitting. That's the reason it performed well with respect to the above models. We got an RMSE Score of 68.20844281020672 for ExtraTreesRegressor_model.



XGBoost Regressor



Of all the singular models we built that had no ensembling, Extreme gradient Boost provided the least RSME of them all. XGBoost is a decision tree based boosting model that combines multiple decision trees in a stage-wise fashion. Each decision tree is a weak learner which would make mistakes when learning on a subset of training data, the next decision tree would then make an attempt to improve upon this by correcting for errors and biases i.e it tries to minimize a loss function. When training the model we had to focus on tuning the hyperparameters which included the following :

1. Number of iterations

2. Max depth of the decision tree

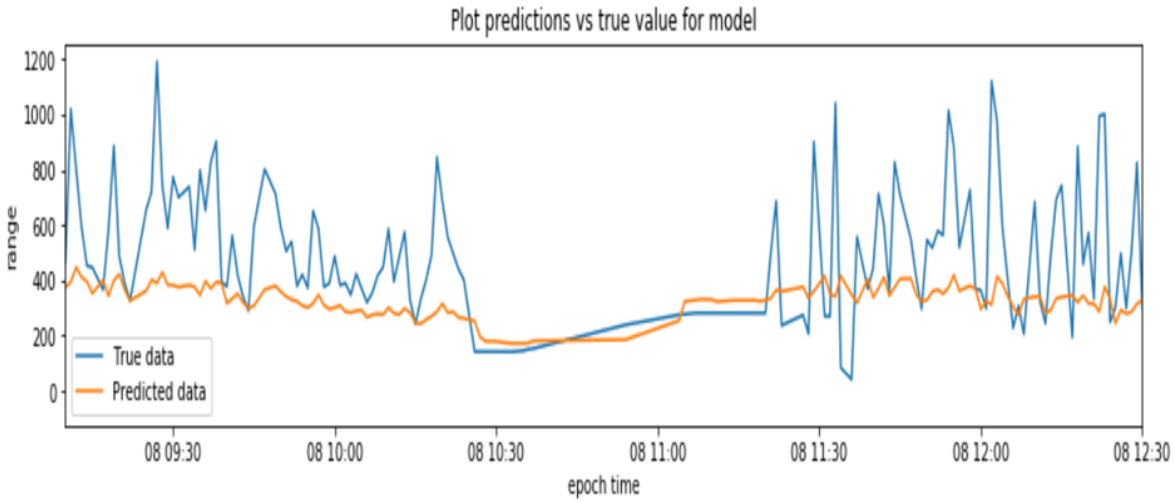
3. Max leaf nodes of the decision tree

4. The number of columns to be randomly sampled for a decision tree There are several more, but we found these to be the most important. After tuning these parameters on the training set by trying to lower the Root Mean Squared Error yet not trying to overfit the data the following was the R.M.S.E on the test data: 66.45261942378283.

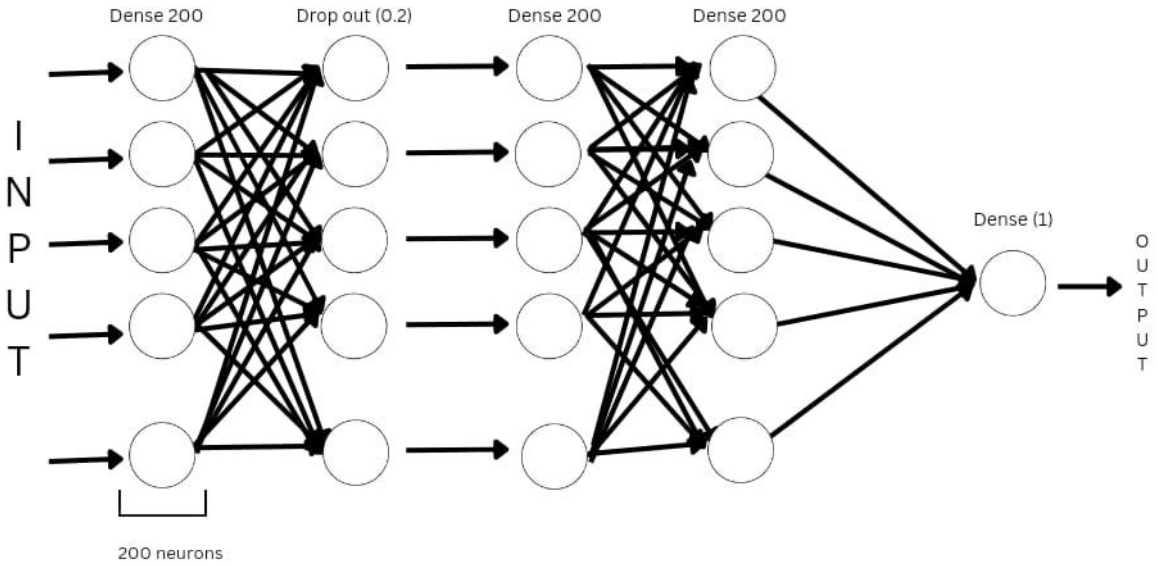
Long Short Term Memory Model (LSTM Model)

LSTM is implemented in Python using the Keras library. LSTM is designed to avoid long term dependency problems. In LSTM the repeating module has a different structure. Instead of having a single neural network, there are four. The key parameter in LSTM is the cell state which consists of some linear interactions. LSTM can add or remove information from the cell state. Gates enable us the option to let information through. The sigmoid function which outputs 0 or 1 helps us decide how much of each value would be let through the gate.

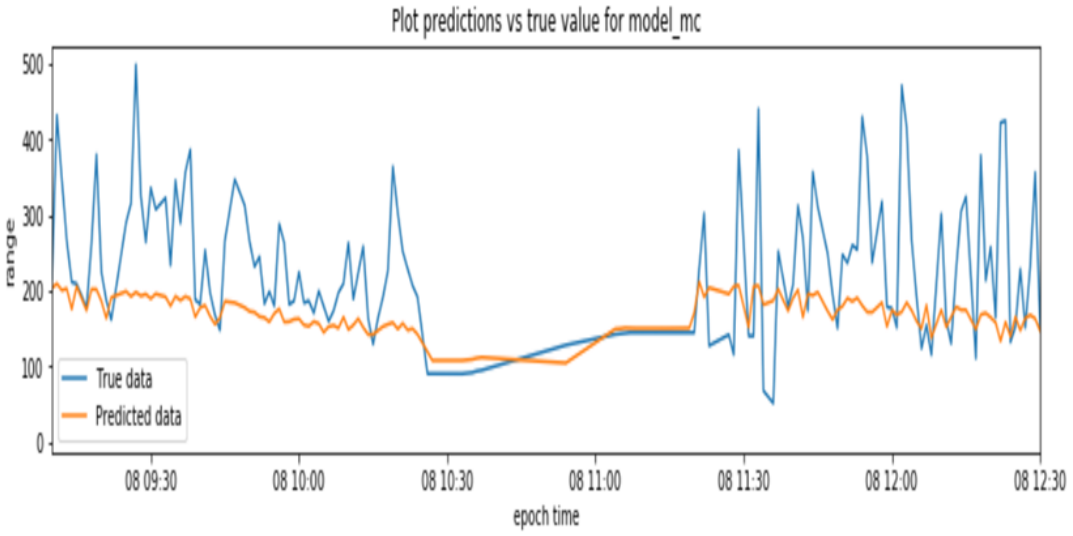
And we got an RMSE score of 198.45261437643283.



Neural Network Model



We use a 5 layer artificial neural network to train the neural network on the features to predict the range. Our neural net consists of 4 dense layers and 1 dropout layer. The dense layer is a deeply connected neural network layer and the number of units in the output layer is 200. We use ReLu as our activation function. We also have a dropout layer to avoid the overfitting of the data. It is a form of regularization. We set the probability of dropping out nodes to 0.2, i.e. we drop 20% of the selected nodes. We got an RMSE Score of 70.82734268643283.



ENSEMBLE MODEL

Up to this point we trained and used individual models i.e. without ensembling of more than one model. Here we tried combining more than one model to calculate the predicted values for the Range of Electric Vehicle.

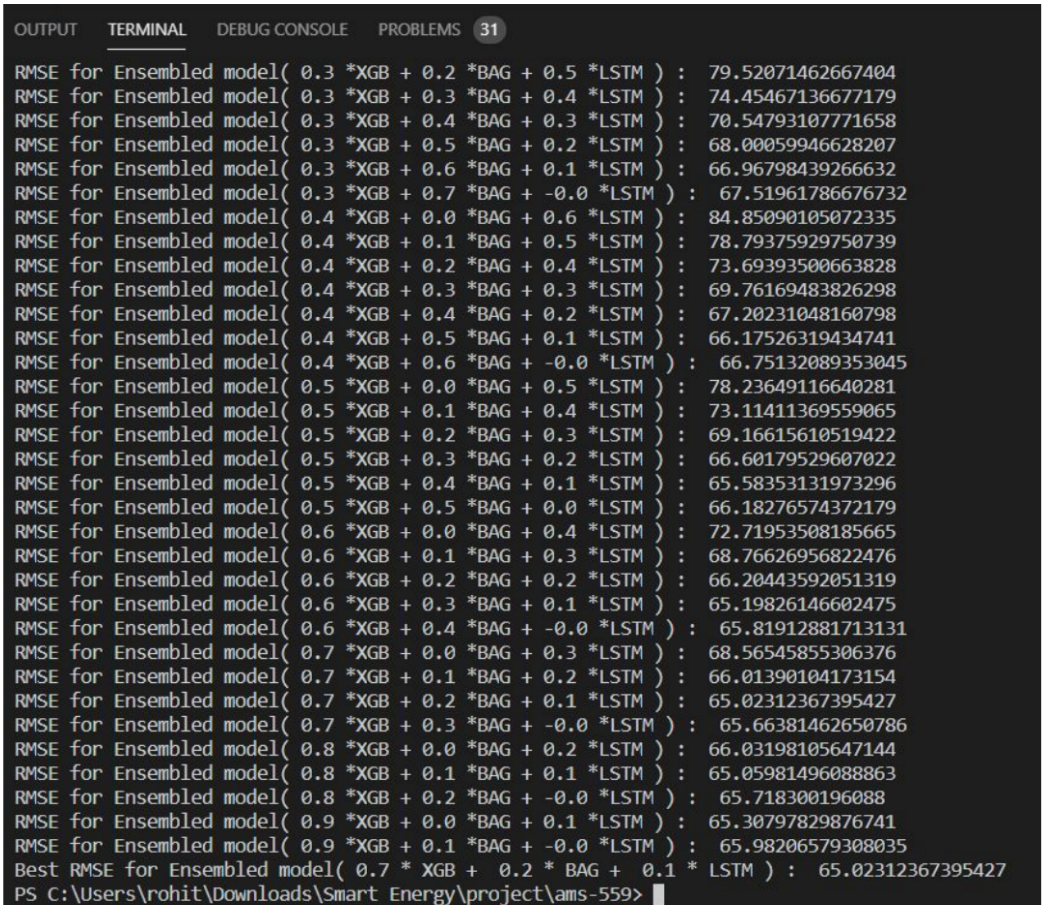
We have tried the combination of XGB Regressor, Bagging Regressor, Extra-Trees Regressor, LSTM with AdaBoost, LSTM with RMSProp Optimizer and NN with Adam Optimizer because these models gave better RMSE scores individually.

Out of many combinations of mentioned models, we got the best results for the combination of XGB + BAG + LSTM with Adam. Then we calculated the best weights for these models which estimate the least RMSE Score as shown in the figure below.

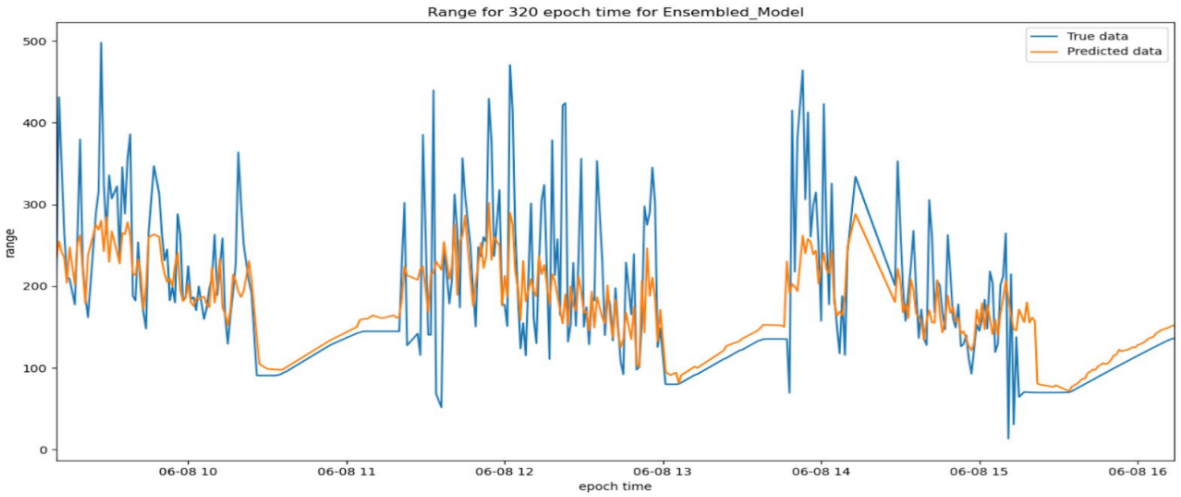
Code snippet

```
weights = np.arange(0,1,0.1)    #[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
min_RMSE = float('inf')

for i in range(len(weights)):
    for j in range(len(weights)):
        if (weights[i]+weights[j] <= 1.0):
            new_y_pred = y_pred_XGB*round(weights[i],1) + y_pred_BG*round(weights[j],
            1) + y_pred_LSTM_adam*(round(1-weights[i]-weights[j],1))
            RMSE = regressors.Calc_RMSE(y_test_range.values,new_y_pred)
            print("RMSE for Ensembled model(",round(weights[i],1),"*XGB +",round
            (weights[j],1),"*BAG +",round(1-weights[i]-weights[j],1),"*LSTM ) : ",
            RMSE)
            if min_RMSE>RMSE:
                min_RMSE= RMSE
                best_weights = [round(weights[i],1),round(weights[j],1),round
                (1-weights[i]-weights[j],1)]
                best_y_pred = new_y_pred
```



The best combination of weight we found for our ensembled model was 0.7 for XGB regressor, 0.2 for Bagging Regressor, and 0.1 for LSTM Regressor i.e. [0.7 * XGB + 0.2 * BAG + 0.1 * LSTM] with least and best RMSE Score of 65.02312367395427. We can observe that in the following plot



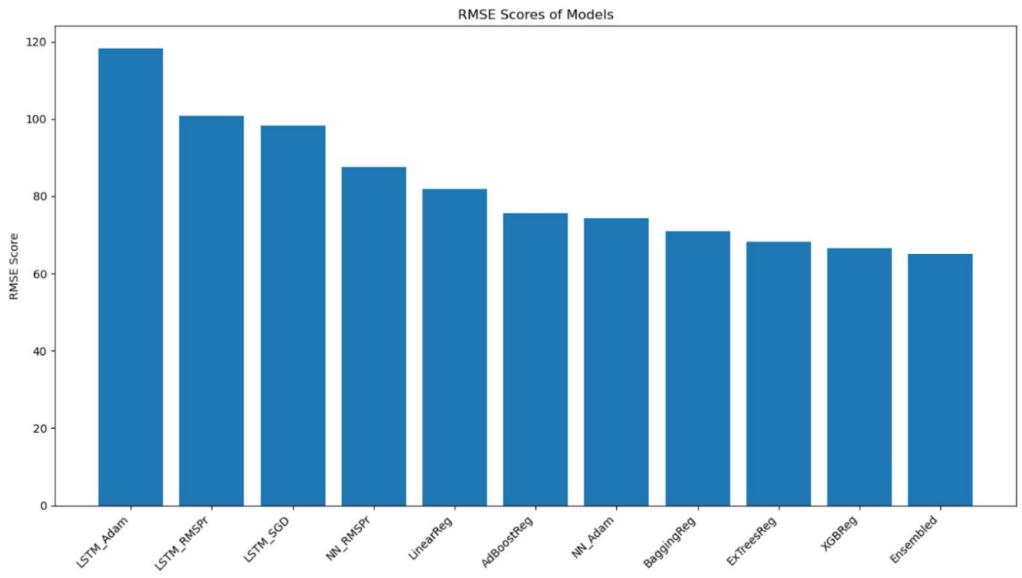
Model	RMSE Score
Linear Regression	81.78280563366944
AdaBoost Regressor	75.63687847244836
Bagging Regressor	70.94967401057501
Extra-Trees Regressor	68.20844281020672
XGB Regressor	66.45261942378283
LSTM with Adam Optimizer	118.21029465014225
LSTM with RMSProp Optimizer	100.80652164432739
LSTM with SGD Optimizer	98.24036065279401
NN with Adam Optimizer	74.25204016912768
NN with RMSProp Optimizer	87.64033313157483
Ensembled Model **(0.7*XGB + 0.2*BAG+0.1*LSTM)**	**65.02312367395427**

CHAPTER 6 : Conclusion and future scope

● From the beginning to this point, we have used various individual models and ensemble models(combinations of models). Resultantly, we got stats as shown in the table below. ● The best non-ensembled model or individual model was the XGBoost Regressor as highlighted with blue color in the table with the RMSE Score of 66.45261942378283. And the overall best performance was given by our ensembled model(highlighted with red color) with a score of 65.02312367395427.

6.1 ANALYSIS

We can explain these results by saying that the Range of Electric Vehicle shows linear relation with time(i.e. decreasing with time as battery charge goes down). Because of that linear relation, it was better explained by our models like XGB Regressor in individual models. Similarly, In ensembled model weight of XGB and BAG Regressor was higher as compared to LSTM, where LSTM is a Recurrent Neural Network that generally best explains back memory-based relations which lacks here due to more linearity. ● Below is the comparison of RMSE scores of the different models using a bar graph:



CONCLUSION

Our approach is focused on developing an electric vehicle (EV) range predictor using machine learning techniques. The project involves several regression models, including Linear Regression, AdaBoost Regression, Bagging Regression, Extra Trees Regression, XGBoost Regression, LSTM with different optimizers, and Neural Network with different optimizers.

By evaluating multiple regression models and incorporating an ensemble approach, the model aims to improve the accuracy and reliability of the EV range predictions.

It contributes to the field of electric vehicles by providing a valuable tool for estimating range, which is a critical factor for EV users and industry stakeholders. The developed models and the ensemble approach can assist in optimizing EV usage, route planning, and charging infrastructure management, thereby promoting the adoption and efficiency of electric vehicles.

Future work could involve exploring additional regression algorithms, optimizing hyperparameters, and incorporating more advanced techniques such as deep learning models to further enhance the prediction accuracy and robustness of the EV range predictor. Additionally, real-time data integration and validation would be valuable for improving the practical applicability of the developed models in real-world scenarios.

REFERENCES

Bi, J.; Wang, Y.; S., S.; and Y., C. 2018. Residual range estimation for battery electric

vehicle based on radial basis function neural network. Measurement, 128: 197–203. Bi, J.; Wang, Y.; and Zhang, J. 2018. A data-based model for driving distance estimation of battery electric logistics vehicles.
J Wireless Com Network, 251: 501–520. Cheah, L. 2021. Overcoming range limits. Nat Energy, 6: 17–18. Ferreira, J. C.; Monteiro, V.; and Afonso, J. L. 2013. Dynamic range prediction for an electric vehicle. In 2013 World Electric Vehicle Symposium and Exhibition (EVS27), 1–11. Fetene, G. M.; Kaplan, S.; Mabit, S. L.; Jensen, A. F.; and Prato, C. G. 2017. Harnessing big data for estimating the energy consumption and driving range of electric vehicles. Transport. Res.
Part D: Transport and Env., 54: 1–11. Javed, M. A.; Ben Hamida, E.; and Znaidi, W. 2016. Security in Intelligent Transport Systems for Smart Cities: From Theory to Practice. Sensors, 16: 1–25.
arey, A.; Kaushik, S.; Khalighi, B.; Cruse, M.; and Venkatesan, G. 2020. Electric Vehicle Range Improvement by Utilizing Deep Learning to Optimize Occupant Thermal Comfort. In NeurIPS 2020 Workshop on Tackling Climate Change with Machine Learning. Zhao, L.; Yao, W.; Wang, Y.; and Hu, J. 2020. Machine Learning-Based Method for Remaining Range Prediction of Electric Vehicles. IEEE Access, 2020: 212423–212441.