INTERNSHIP PROJECT REPORT

Topic:

IOT APPLICAION IN LOGISTICS AND TRANSPORTATION

Tata Consultancy Services Ltd

Thane Olympus Campus

Duration of internship:

6th January 2020 to 27th April 2020

Department:

A&I

Submitted by: Meghna Roy Chowdhury

Employee ID : 1838796

Supervisor: Mr Meghraj Nalge

Index

ACKNOWLEDGEMENT

I wish to thank those who were involved in the successful completion of my internship at TATA CONSULTANCY SERVICES.

It is my proud privilege to express my profound gratitude to my mentor Mr Meghraj Nalge for guiding me and for providing me with such an avenue to help realize how interesting it is to work in today's industry. I'd also like to thank Mr Ashish for helping me out at the campus. I'd also like to thank my HR , Mr Gaurav Ghelani for providing me this opportunity.

I'm also thankful to the management of Vellore Institute of Technology, Chennai Campus - Dean of SENSE, Dr.Sivasubramaniam and Program Chair Dr.Vetrivelan.P providing me this valuable opportunity to have industrial exposure.

Last but not the least ,I would also like to thank my parents, for being my motivation to take up this internship.

.

TIMELINE

- January 6th 2020- Joining
- January 9th to February 15th – Prework
- February 17th to April 24th – Project work

I. PREWORK – LITERATURE SURVEY (January-February)

1. Research about use cases and IOT frameworks of transportation and logistics industry.

Fleet Management

a. Role of IOT:

In logistics, the supply chain is made up of various stakeholders including manufacturers, transport companies, and retailers. Information sharing and connectivity is essential for all of them. Therefore, IoT has an important role to play in this industry.

i. Prompt and easy communication… Scheduling, load management, driver and vehicle tracking, and effective routing are some additional benefits offered by Internet of Things

ii. Internet of Things has the ability of derive collectivity data from various sensors. This enables fleet management companies **to get useful insights into driver behavior, adherence to laws, vehicle speeding and idling** etc..

iii. Proper implementation of **IoT reduces overall costs** by optimum utilization of resources and improved performance.

iv. Internet of Things also allows logistic companies to move daily operations into the cloud and helps remote tracking of fleets at any time from any place.

b. Framework that can be used:

- RFID
- GPS
- Predictive analysis tools (machine learning)
- IOT frameworks
- Mobile IOT
- Cloud platforms

c. Analysis :

- Driver behavior monitoring -
- Fuel monitoring
- Loss prevention
- Predictive fleet maintenance
- weather APIs
- traffic reporting
- smart parking
- maintenance monitoring
- effective routing

Public transit management

a. Role of IOT:

i. reduce traffic accidents and increase overall safety.
ii. Providing customers and
iii. Analyze best possible routes
iv. Increase safety
v. IoT devices can ensure drivers are adhering to established industry and/or employer guidelines
vi. Better security

b. Framework that can be used-

- Machine learning algorithms
- Geofencing
- GPS
- Cloud platforms

c. Analysis-

i. **Real-time vehicle location information and data**
   - real-time vehicle location information and data
   - estimated trip times, and notifications when they should leave their current locations to catch a bus or train
   - analyze and improve routes and options based on aggregated data.

ii. **Predictive, preventive maintenance**
   - to monitor the health of a fleet from any connected device.
   - assess the condition of vehicles and foresee failures before they occur.
   - set up alerts for low battery, check engine, oil change, coolant temperature, inspection reminders

Smart inventory management

a. Role of IOT:

The role of IoT in inventory management boils down to turning the data fetched by RFID readers into meaningful insights about inventory items' location, statuses, movements, etc., and giving users a corresponding output

b. Framework that can be used –

i. machine learning – a component of an IoT-based inventory management solution architecture – can forecast the amount of raw materials needed for the upcoming production cycle.

c. Analysis

1. **Automation of inventory tracking and reporting -** Automated asset tracking and reporting save up to 18 hours of working time per month and reduces the probability of human error using IOT and RFID

2. **Constant visibility into the inventory items in terms of quantity, location and movements-**
   - flow of raw materials and component
   - work-in-progress and finished goods
   - real-time updates about the status, location, and movement of the items

3. **Inventory optimization-**
   real-time data about the quantity and the location of the inventory items,….
   hence manufacturers can lower the amount of inventory on hand while meeting the needs of the customers at the end of the supply chain.

4. **Identifying bottlenecks in the operations**
   With the real-time data about the location and the quantity of the inventory items, manufacturers can reveal bottlenecks in the manufacturing process and pinpoint machines with lower utilization rates.
5. **Lead time optimization**
   data about the amount of available inventory and machine learning-driven demand forecasts, solutions based on IoT allow manufacturers to reduce lead times

Optimal asset utilization

Transportation Asset Management links user expectations for system condition, performance, and availability with system management and investment strategies. Regardless of the definition, the focus is on performance of assets. The underlying goal of asset management is to take a broad approach to resource allocation and programming decisions that will provide greater value to the system and overall satisfaction for end users through improvements in program effectiveness and system performance.

a. Role of IOT
   - asset tracking became much faster and easier
   - Physical assets like location, status, and other information can be tracked and managed online in the real-time.
   - The truck owner can know the actual load of their truck and its location.
   - Asset's latitude and longitude can be known through IoT.
   - Threshold and tolerance of the device can be known by the advanced analytics system and used to manage the trucks ahead.

b. Framework that can be used
   - **Biz4Intellia,** an end-to-end IoT solution provider - track the real-time location of their truck and can know how much load there is on the trailer of the respective truck. Not only this, but the latitude and longitude of an asset can also be known by IoT in transportation.

c. Analysis:
   i. **Monitoring Provides Clear Accountability and Feedback - These are done by** tracking all the devices like sensors, axels, and tells about the threshold and tolerance of the device
   ii. **Decisions Based on Quality Information -** AI-generated predictive analytics can be used to avoid risk, optimize routes, predict demands, prevent inaccurate stocking, and respond proactively to future developments.
   iii. **Analysis of Options and Tradeoffs-** analyze the collected data and identify patterns to take the right actions and optimize business processes.

2. Specific use cases , IOT sensors to be used , model to be used for analysis

Vehicle Monitoring

1. Excessive heating of the engine –
   - Coolant Temperature sensor
   - LM-35 temperature sensor

   If the temperature crosses the threshold value, the microcontroller reduces the speed of the vehicle gradually to zero; and this information is sent to the officials and the driver so that appropriate actions can be taken.

2. Fuel issues:
   i. Blockage in pipe
   ii. Leakage detection
   iii. Detection of the amount of fuel – level sensor

   - Ultrasonic sensors –
     - Measure the entire pipe distance and blocked location of the distance.
     - Subtract blocked location of pipe and entire pipe length to get the exact location of the blockage in the pipe.

   - Flow Sensor + Gas Detection Sensor+ Level Sensor :
     - The information of that respective sensor is being sent to a microcontroller.
     - Based on the input received from the different sensors, the microcontroller compares these values with the threshold.
     - The driver is notified and the location of the blocked location of the fuel channel to the officials through GSM could easily locate which blocked channel and could take appropriate steps.

The mathematical equation:

The equation of float rate of fuel can be shown as follows:
Q=V*A
(Q is glide rate/total drift of gas through the pipe)
V is average speed of the flow.
A is the cross-sectional place of the pipe.

1) Pulse frequency (Hz) = 7.5Q , Q is float price in Liters/minute.
2) Flow Rate (Liters/hour) = (Pulse frequency x 60 min) / 7.5Q.)

3. Tracking vehicle location for emergency response and geo-fencing
   i. GSM and GPS module
      - integrate with Google Maps
      - In case of wrong route, the authorities are informed

4. Ignition locks or unlock through IoT:
      - Checks whether the authorized user has locked or unlocked the ignition of vehicle.

Analysis:

Dataset with values of temperature of engine, and sensors to analyse the data and predict how long the vehicle will run.

This can be done with **Random Forest decision**- will decide the chances of accident based on the parameters. The live location will be sent to the appropriate people.

Driver Monitoring

- Driver health:
  - Sensing body temperature – Using thermometer or LM35 sensor.
  - Heart-rate monitoring – Using Pulse sensor

  These can be integrated into one system like a FitBit. The data can be viewed through an app. If any particular parameter is above threshold, an intimation will be given to the driver and the officials so that appropriate action can be taken.

- Detect distracted driver using Image Processing-

  Distractions can be classified as:

  > Texting
  > talking on the phone
  > eating
  > reaching out to the back of the vehicle while driving

  We try to understand the training data and behavior of data. Using Image pre-processing and deep learning methods for identifying the behavior of the driver. Notify the driver to avoid accidents.

- Driver safety
  - Seatbelt on – **this can be detected with an IR sensor. The required information can be sent via internet.**

  - Alcohol detection

    This can be detected using **MQ2 sensor. The IOT device used can be Zigbee. .**
    If the threshold is crossed, the MC-1 turns OFF the ignition of the vehicle and makes the LCD to display. The MC-1 activates the GPS module to detect the current location and send the message to the concerned person.

  Analysis:

  Driver behavior (what kind of distraction, alcohol, and seatbelt) can be analyzed in different cases and the driver and concerned people can be informed.

> This can be done using **Support Vector Machine**(due to the hyper plane that is made and makes image processing simpler)
> This can be done using **LVQ.** Each of the distraction cases can be classifies into classes and can be learnt during the training period. And image can be classified into these classes in the testing stages and the model can learn.
> Image pre-processing and **Deep learning CNN** for identifying the behavior of the driver.

Route Optimization

- **Sensing bumps and potholes by fleet vehicles (for optimizing subsequent fleet routes, if needed)**

Attach IoT sensors like **accelerometer, gyroscopes, and IR scanner integrated with Raspberry Pi or Zigbee or any controller** onto vehicles as data collection devices. Data collected from these sensors are aggregated and analyzed using machine learning algorithms to automatically classify the various road surface conditions.

With the system, there can be a cost-effective way to monitor roads, which in turn allows for faster repairs and ultimately, improve the overall safety of roads

The conditions considered in this study include:

- **smooth surfaces**
- **uneven surfaces**
- **Potholes**
- **speed bumps**
- **Rumble strips.**

Statistical features such as the following are considered, both time and frequency domain forms

- **minimum**
- **maximum**

- **standard deviation**
- **median**
- **average**
- **skewness**
- **kurtosis**

Selection of features is performed using:

- **Ranker**
- **Greedy Algorithm**
- **Particle Swarm Optimization (PSO)**

Classification is done using:

- **k-Nearest Neighbor (k-NN)**
- **Random Forest (RF)**
- **Support Vector Machine (SVM) with linear and polynomial kernels**

- **Monitor traffic to sense average speed of vehicles on the road (more speed implies less traffic)**

  Digital Image Processing technique and sensors data, resulting in output as signals management. An algorithm is used to predict traffic density to minimize traffic congestion.

  Besides this, RFIDs are also used to prioritize the emergency vehicles like ambulances, fire brigades, etc. by implementing RFID tags in such vehicles. In the case of emergency situations, such as an explosion or burning of something, fire and smoke sensors are also deployed on the road to detect such situations.

  Support Vector Machine (SVM) is best for DIP.

Weather Condition Monitoring

- Monitor wind speed (anemometer)
- Monitor humidity (Humidity and temperature sensor – DHT11)
- Monitor temperature

What to do:

> Collect data from various vehicles on the road.
> Vehicles moving on the road can wirelessly communicate the weather and road condition data.
This data helps to build more accurate forecast and provide flexible real time monitoring at different time horizon.
> Minute details like **temperature, fog, road condition, light, flood, stormy** and other condition that would add up to reliability and accuracy of the report. Hence, along with promoting smart and intelligent driving system, it will also **uplift the safety and security of drivers with notifiable fall in road accidents** caused because of weather hazards
> Sensors are installed on windshields, wipers and tires of car. These sensors in integration with IoT help in collecting weather data which is further pooled in cloud for analysis

The sensors which can be used are:

i. temperature sensor
ii. humidity sensor
iii. light sensor
iv. rain level sensor / rain gauge
v. Soil Moisture
vi. Wind speed sensor

We can use data from multiple weather stations to train simple machine learning models, which can provide usable forecasts about certain weather conditions for the near future within a very short period of time. The models can be run on much less resource intensive environments.

The observations of weather data includes:

- minimum-maximum temperature (in Degrees Celsius)
- rainfall (amount recorded for day in mm)
- evaporation (mm)
- sunshine (number of hours of bright sunshine in a day)
- rain today , Rain tomorrow(will be predicted)
- wind gust speed (km/h)
- humidity (percent)
- pressure (Atmospheric pressure (hpa)

- cloud (0 for clear sky variably 8 for complete overcast)

Two variables, “Rain Today” and “Rain Tomorrow” are factors with levels “Yes” and “No” which are further converted to numeric for Regression Algorithms implementation

Analysis:

- Regression technique - We find that **Random Forest Regression (RFR**) ensembles multiple decision trees while making decision.
  We can show comparison of several other state-of-the-art ML techniques with the RFR technique.
  The incorporated regression techniques are **Ridge Regression (Ridge), Support Vector (SVR), Multi-layer Perceptron (MLPR), and Extra-Tree Regression (ETR)**.

- **Linear Regression, Regression Decision Tree, Binary Logistic Regression, Decision Tree Classification, Principle Component Analysis and Clustering** techniques of machine learning to forecast weather achieving higher accuracy.

Package Monitoring

- Detect physical shocks and smashes to ensure product is in good condition when delivered to the customer
- Shipping time prediction

Dataset can include:

- Weather patterns in the model as a way to improve its accuracy, if access to reliable data can be guaranteed.
- Position of the container in the delivery vehicle (The position of the container determines when it gets loaded and unloaded from the ship.)

Can be done using:

- **Neural Network**
- **Linear Regression**
- **Random Forest algorithm** Random Forest performs better, probably because there is not enough complexity in the data for the Neural Network to really shine.

3. Selection of 2 use cases and defining datapoints

Use case 1:

- Hypothesis - Vehicle is going for servicing too much

- Possible reasons : Rash driving, bad road conditions , faulty/duplicate spare parts

- Possible data points:
  i. Model
  ii. Year of manufacture
  iii. Weight
  iv. Monthly servicing (number of times)
  v. Route taken
  vi. Road condition
  vii. Mileage
  viii. Distance travelled(average monthly)
  ix. Travel time (per day)

Use case2:

- Hypothesis – Very frequent fuel refill

- Possible reasons:
  i. Blockage in pipe
  ii. Leakage of fuel
  iii. overheating
  iv. Stealing of fuel

- Possible data points:
  i. Model
  ii. Year of manufacture
  iii. Weight
  iv. Route taken
  v. Mileage
  vi. Distance covered
  vii. Average fuel consumed per month
  viii. Stoppage time

II. ACTUAL PROJECT – PREDICTING LIFETIME OF ELECTRIC VEHICLES

(February- April)

1. Introduction to Electric Vehicles and Li-ion battery

Lithium-ion batteries are currently used in most portable consumer electronics such as cell phones and laptops because of their high energy per unit mass relative to other electrical energy storage systems. They also have a high power-to-weight ratio, high energy efficiency, good high-temperature performance, and low self-discharge.

Lithium-ion batteries power almost every electronic device in our lives, including phones and laptops. They're at the heart of renewable energy and e-mobility. For years companies have tried to predict how many charging cycles a battery will last before it dies. Better predictions would enable more accurate quality assessment and improve long-term planning.

But that's difficult, because every battery ages differently, depending on its usage and conditions during manufacturing

Through this project, we will find , how many cycles has a battery cell lived through and how many cycles will it last before it breaks given measurements during a limited amount of charging cycles.

2. Defining problem statement and data points based on use cases

Problem Statement- >

Predicting **remaining cycle life** of used and unused batteries ; i.e predicting the number of times a battery can be fully charged and discharged before the reach the end of their functional life.

Thereby, we're predicting their **rate of degradation and remaining useful life**

This way we can determine **the driveable range of the electric vehicle**.

Some of the required datapoints needed for this prediction includes:

1. Age of the battery
2. Time of testing - The battery needs to be tested at different time intervals as the charge capacity differs at different times.
3. Internal resistance
4. Voltage
5. Current

6. temperature
7. Charge capacity
8. discharge capacity
9. charge energy
10. discharge energy
11. state of charge(SoC- it depends on the charging and discharging voltage)
12. charge throughput

To determine the driveable range of an electric vehicle, the following data must be needed as well:

1. vehicle model
2. Road/traffic condition
3. mileage
4. Power
5. Driver behaviour

3. Creating the dataset with hypothesis

About the dataset:

Number of attributes: 921 , Number of columns: 21

```
In [3]: dataset.shape
Out[3]: (921, 21)
```

Made my own dataset with the following datapoints:

- Datetime\n(DDMMYYYY)
- 'Today'
- 'Age'
- 'Cycle'
- 'Temperature',
- 'ChargeTime'
- 'Cell voltage (V)'
- 'Current capacity (mAH rati ng)'
- 'Crate'
- 'SOC'
- 'DOD'
- 'SelfDischargeRate'
- 'SOH'
- 'InternalResistance
- 'RoadCondition'
- 'Power'
- 'Driver'
- 'Distance',
- 'Stop time\n(while engine is ON) (hrs)'
- 'TotalOnTime',
- 'RemainingLife'

Hypothesis:

1. Temperature is more than room temperature(25-30 C):
    - the SOC decreases
    - charging capacity decreases
    - Internal resistance of battery increases
    - DOD increases
    - remaining life of battery decreases
2. As the age of the battery increases,
    - self discharging capacity increases
    - charging capacity decreases
    - self discharging increases
    - remaining lie of battery will reduce
3. Remaining life of battery depends on road condition
    - Hilly road – Uses more power
    - City road- Has more stoppage due to traffic, hence battery will consume more power
    - Rainy condition- Vehicle has to move slowly, so less power is required.
4. Charging time:
    - Charging more than required time leads to more discharging and hence the battery life reduces
    - Charging less than required results in less battery capacity and hence battery life reduces
6. Driver behaviour:

    Depending on driver behaviour(rash, average, slow), the battery life differs.

4. TASK 1:
Performing different analysis on the dataset to confirm hypothesis

i. LINEAR REGRESSION

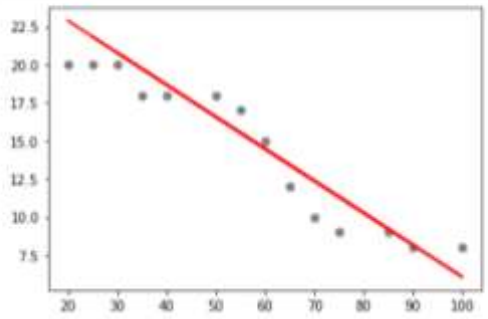
Libraries used:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

a. Temperature vs Remaining Life:

Linear Line

```
In [45]: plt.scatter(X_test, y_test,  color='gray')
         plt.plot(X_test, y_pred, color='red', linewidth=2)
         plt.show()
```

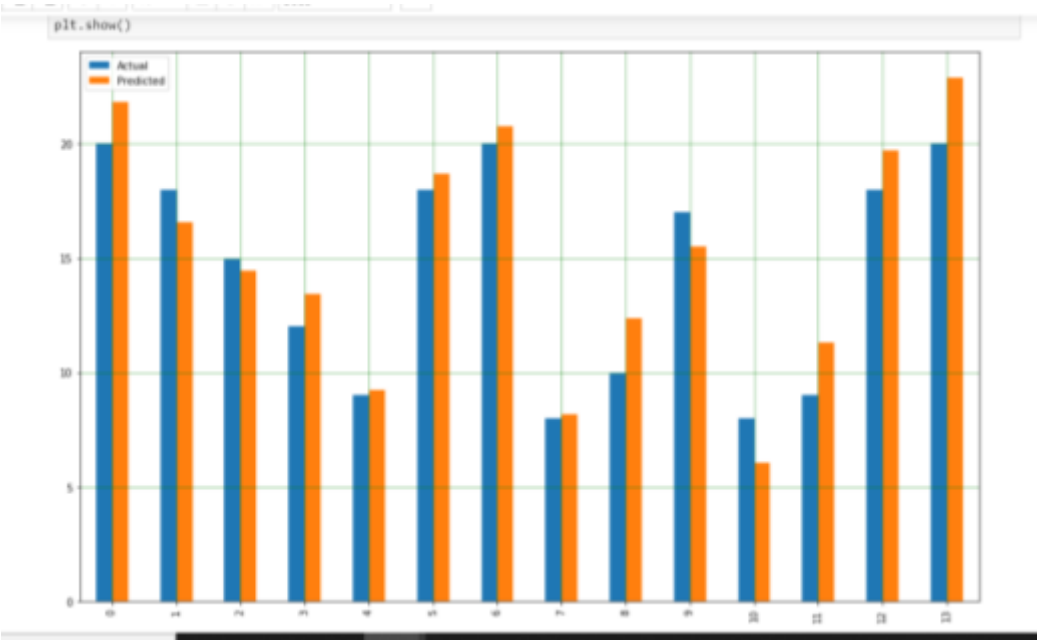


Coefficients:

```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[27.07594937]
[[-0.21012658]]
```

Predicted vs actual values



Error Metrics

In [46]:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1.4086799276672686
Mean Squared Error: 2.6367340398745585
Root Mean Squared Error: 1.6238023401493664
```

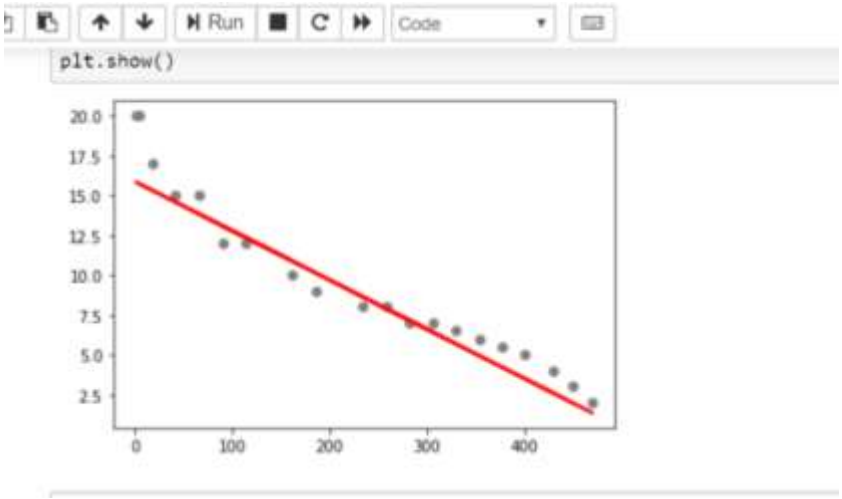
Accuracy

```
metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
0.9233515351201461
```

b. Age vs Remaining Life:

Linear Line

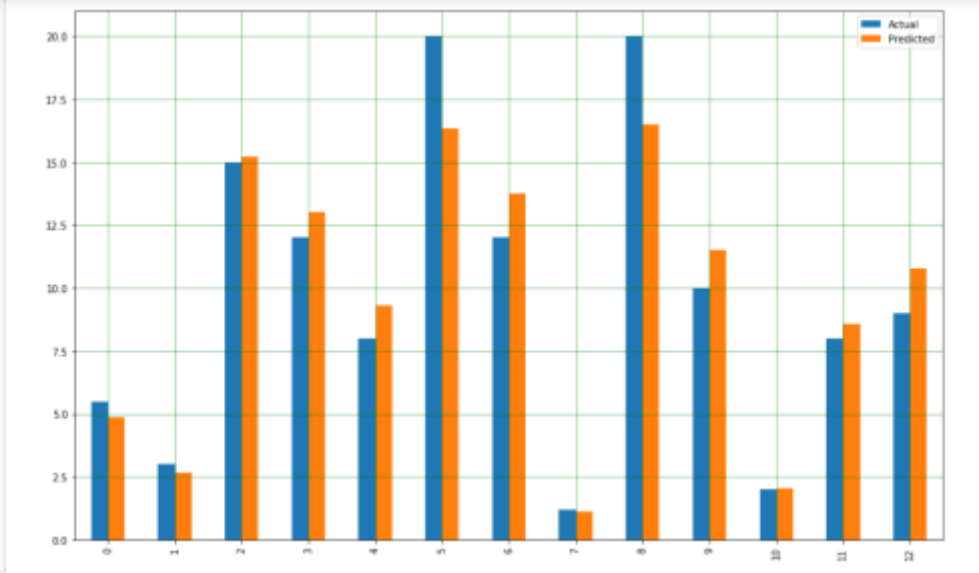


Coefficients:

```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[17.09683443]
[[-0.03291702]]
```

Predicted vs actual values



Error Metrics

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1.2197363838300972
Mean Squared Error: 2.6567856311864184
Root Mean Squared Error: 1.6299649171642985
```

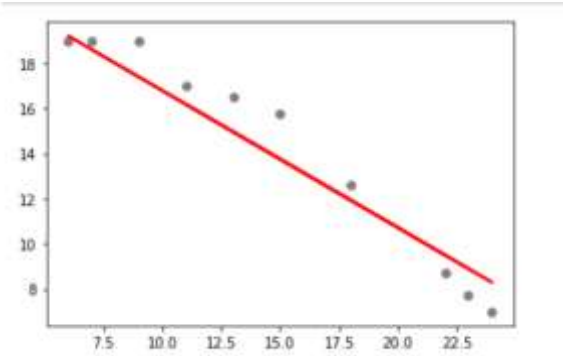
Accuracy:

```
metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
0.9341363185917134
```

c. Charge time vs Remaining Life

Linear Line

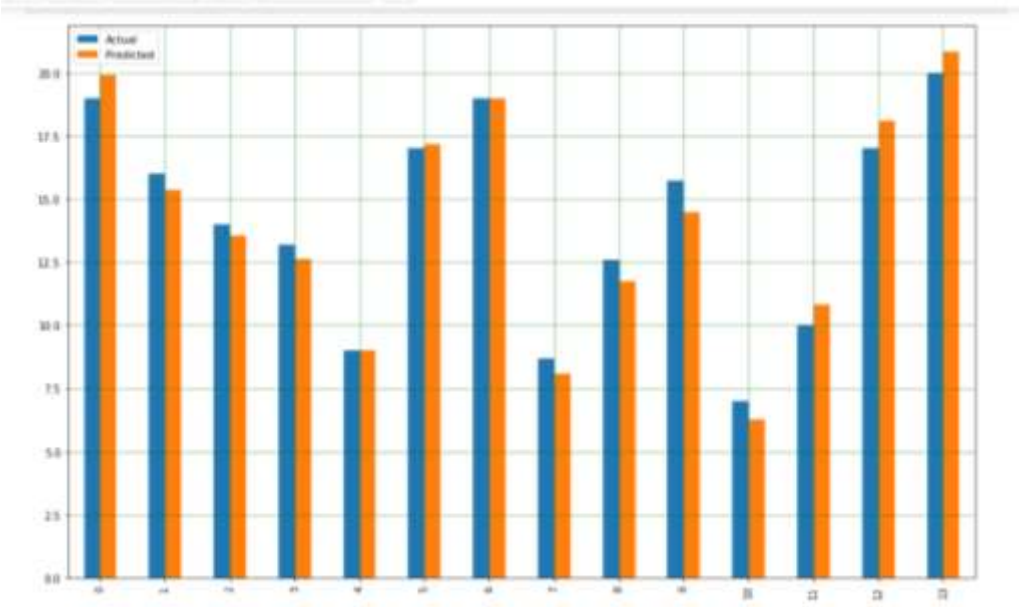


Coefficients:

```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[22.86502311]
[[-0.60739599]]
```

Predicted vs actual values



Error Metrics

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1.0505161787365183
Mean Squared Error: 1.3933214813117751
Root Mean Squared Error: 1.180390393603648
```

Accuracy

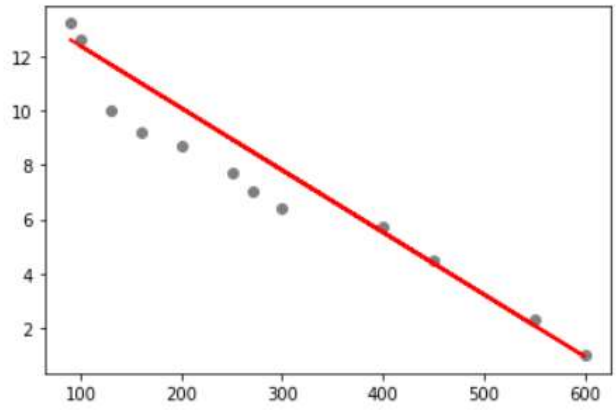
```
metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
0.7981774701844286
```

d. Cycles vs Remaining Life

Linear Line :

```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```

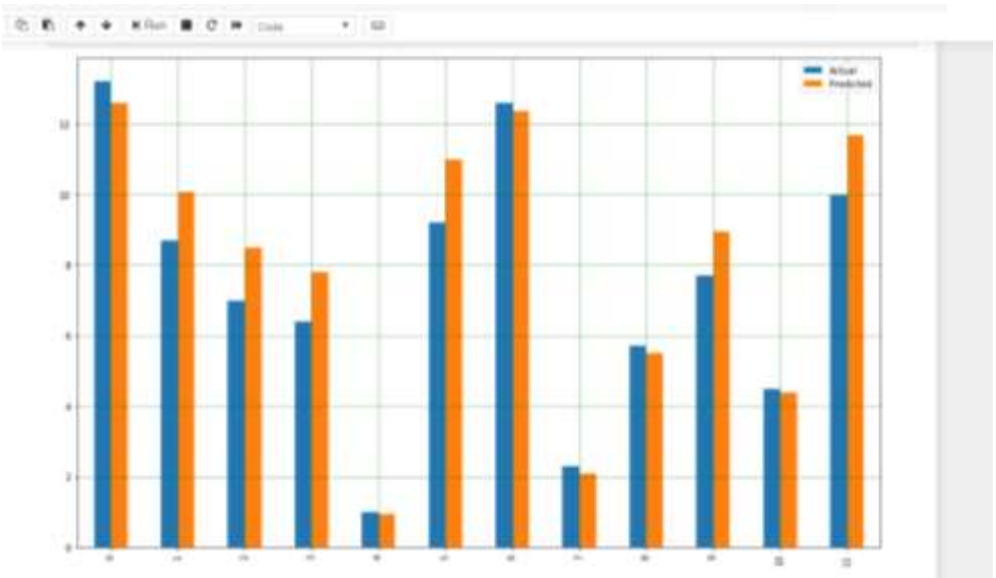


Coefficients:

```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[14.65263158]
[[-0.02283934]]
```

Predicted vs actual values



Error Metrics

```
In [111]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
          print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
          print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

          Mean Absolute Error: 0.8665512465373965
          Mean Squared Error: 1.1844480136227746
          Root Mean Squared Error: 1.0883234875820582
```

Accuracy

```
metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
: 0.9161378443942128
```

5. Multilinear regression

Libraries used:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

Features used:

Dependant Variables:

- 'Age'
- 'Cycle'
- 'Temperature'
- 'ChargeTime'

- 'RoadCondition'
- 'Driver'
- 'Distance'
- 'TotalOnTime'

Independent Variable:

- Remaining Life
-

Input is taken in a numpy array:

```
#X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

Intercept and coefficients:

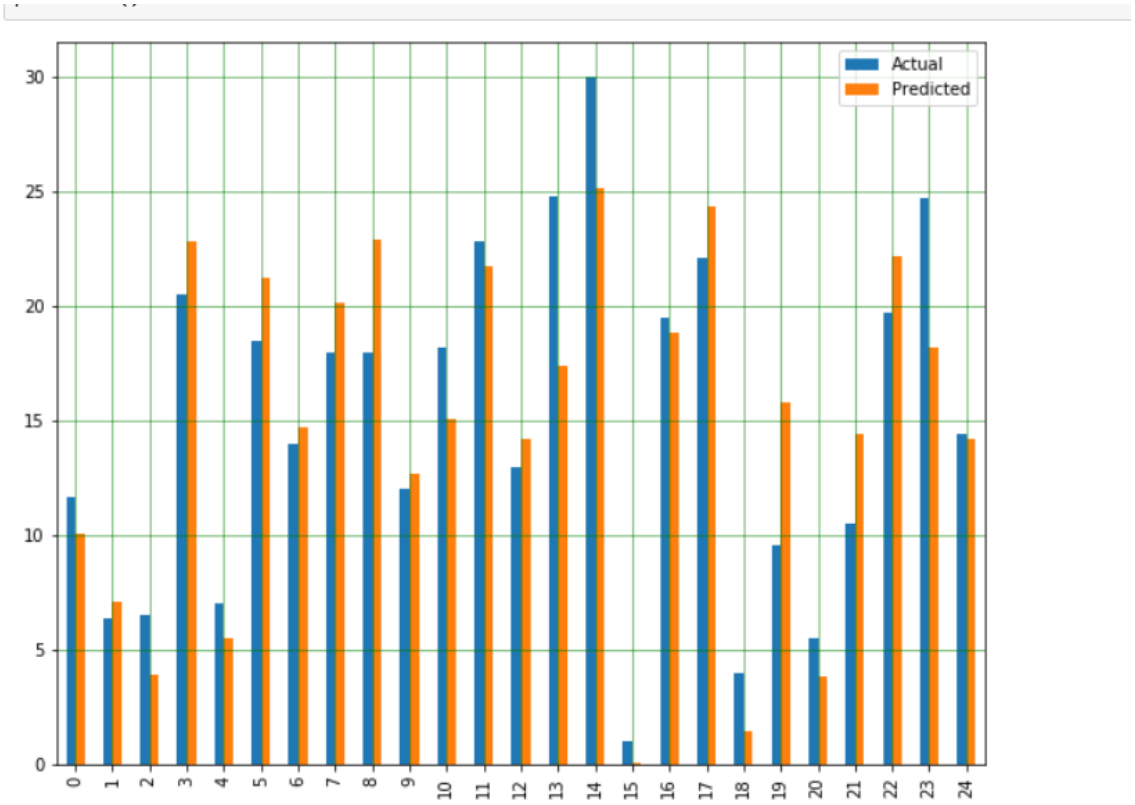
```
In [35]: reg.intercept_
Out[35]: 17.360084656669805
```

```
In [38]: coeff_df = pd.DataFrame(reg.coef_.reshape(-1,1), x.columns, columns=['Coefficient'])
         coeff_df
```

Out[38]:

	Coefficient
Age	-0.030619
Cycle	-0.030155
Temperature	0.004716
ChargeTime	-0.471727
Driver	0.339890
RoadCondition	0.423331
Distance	0.000000
TotalOnTime	1.660915

Prediction vs Actual:



Error metrics:

a) Mean Absolute Square, Mean square , Root mean square

```
In [45]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

         Mean Absolute Error: 2.255071760866235
         Mean Squared Error: 9.433320342542189
         Root Mean Squared Error: 3.0713710851250435
```

b) R sqaure

```
In [55]: reg.score(x,y)
Out[55]: 0.51047259213822
```

c) Adjusted R square

$$R^2_{adj.} = 1 - (1 - R^2) * \frac{n-1}{n-p-1}$$

```
In [56]: x.shape
Out[56]: (921, 8)

In [57]: r2 = reg.score(x,y)
         n = x.shape[0]
         p = x.shape[1]

         adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
         adjusted_r2
Out[57]: 0.5061784920692571
```

R square value must be between 0.2 and 0.9 . It defines how well your model fits the data.

R square and adjusted r square must be almost equal.

Accuracy

The accuracy is about 70% when we take the above features.

```
In [200]: metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
Out[200]: 0.6938014876058244
```

The accuracy decreases to around 54 % if we consider other features because only the above features show a somewhat linear model.

```
In [421]: metrics.r2_score(np.exp(y_test), np.exp(y_pred), sample_weight=None, multioutput='uniform_average')
Out[421]: 0.5492505654051909

In [422]: #X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime
          #x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
          #x_temp1=np.transpose(x_temp)
          #y_temp=np.array([20])


          #'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
          #'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
          #'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
          #'Power', 'Driver', 'Distance', 'TotalOnTime'
          x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
          x_temp1=np.transpose(x_temp)
          y_temp=np.array([20])
```

6. Random Forest

Libraries Used:

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
from sklearn import metrics
```

```
import matplotlib.pyplot as plt
```

Features used :

Dependant Variables:

- 'Age'
- 'Cycle'
- 'Temperature'
- 'ChargeTime'
- 'RoadCondition'
- 'Driver'
- 'Distance'
- 'TotalOnTime'

Independent Variable:

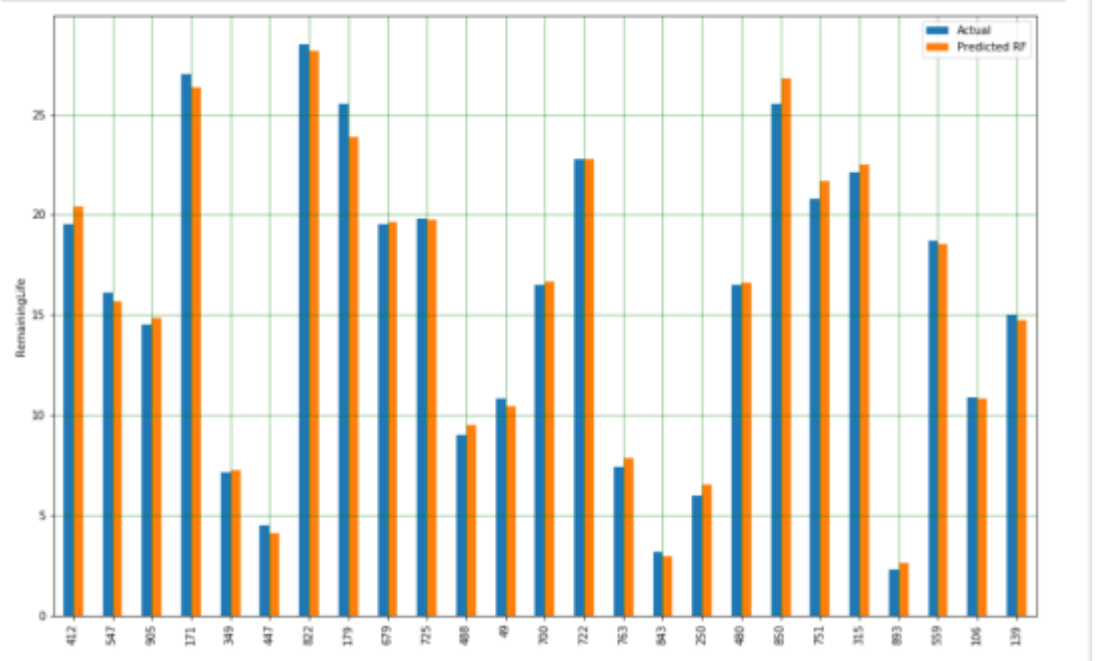
- Remaining Life

```
#X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].value
x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

Parameters:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

Prediction vs Actual:



Error metrics:

i. Mean square error = 0.46
ii. Mean absolute error= 0.45
iii. Root mean square error= 0.68

```
print("Random Forest training mse = ",rf_mse," & mae = ",rf_mae," & rmse = ", sqrt(rf_mse))
```

```
Random Forest training mse =  0.46072155790960456  & mae =  0.45794067796610166  & rmse =  0.6787647294236823
```

```
metrics.r2_score(y_test, y_pred_rf, sample_weight=None, multioutput='uniform_average')
```

```
29]: 0.9865053730596288
```

The accuracy increases about 0.4% when we include more features

```
In [62]: metrics.r2_score(y_test, y_pred_rf, sample_weight=None, multioutput='uniform_average')
```

```
Out[62]: 0.9909745690102187
```

```
In [63]: #X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
#x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
#x_temp1=np.transpose(x_temp)
#y_temp=np.array([20])


#'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
#'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
#'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
#'Power', 'Driver', 'Distance', 'TotalOnTime'
x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

7. Decision Tree

Libraries Used:

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
from sklearn import metrics
```

```
import matplotlib.pyplot as plt
```

Features used :

Dependant variables:

- 'Age'
- 'Cycle'
- 'Temperature'
- 'ChargeTime'
- 'RoadCondition'
- 'Driver'
- 'Distance'
- 'TotalOnTime'

Independent Variable:

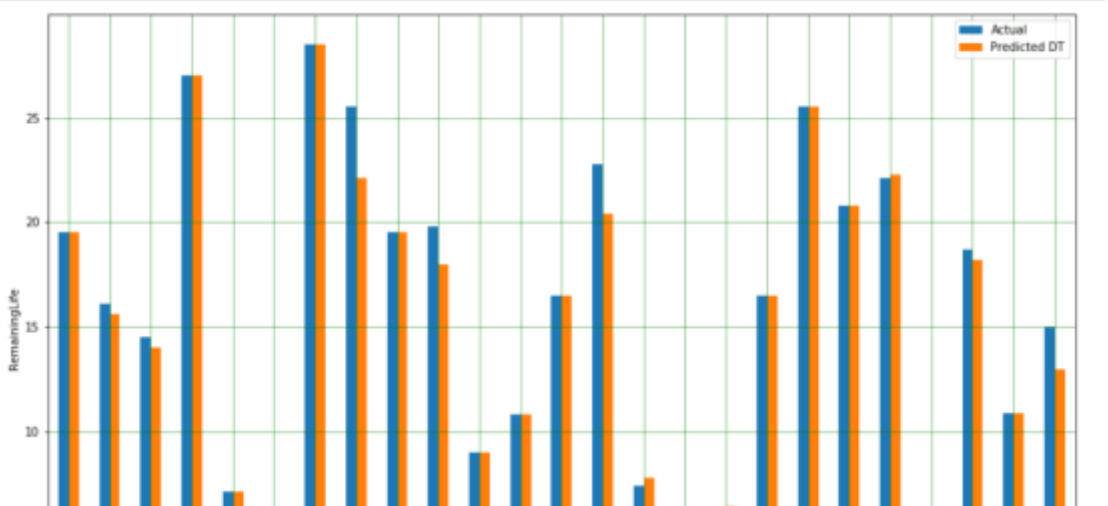
- Remaining Life

```
]: #X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

Parameters:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

Prediction vs Actual:



Error metrics:

i. Mean square error = 0.61
ii. Mean absolute error= 0.40
iii. Root mean square error= 0.78

```
print("Decision Tree training mse = ",tree_mse," & mae = ",tree_mae," & rmse = ", sqrt(tree_mse))
```

```
Decision Tree training mse =  0.6101242937853109  & mae =  0.4097175141242939  & rmse =  0.7811045344800598
```

Accuracy:

The accuracy is the model is around 98% with the above features.

```
metrics.r2_score(y_test, y_pred_tree, sample_weight=None, multioutput='uniform_average')
```

```
6]: 0.9801180731984511
```

The accuracy of the model increases by 0.8% when we include more features
.

```
metrics.r2_score(y_test, y_pred_tree, sample_weight=None, multioutput='uniform_average')
```

```
0]: 0.9880478032464268
```

```
#X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
#x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
#x_temp1=np.transpose(x_temp)
#y_temp=np.array([20])


#'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
#'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
#'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
#'Power', 'Driver', 'Distance', 'TotalOnTime'
x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

8. K Nearest Neighbour

Libraries Used:

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neighbors import KNeighborsRegressor
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
from sklearn import metrics
```

```
import matplotlib.pyplot as plt
```

Features used :

Dependant Variables:

- Age',
- 'Cycle'
- 'Temperature'
- 'ChargeTime'
- 'Cell voltage (V)',
- 'Current capacity'
- 'Crate'
- 'SOC'
- 'DOD',
- 'SelfDischargeRate'
- 'SOH'
- 'InternalResistance'
- 'RoadCondition'
- 'Power'
- 'Driver'
- 'Distance'

- 'TotalOnTime

Independent variable:

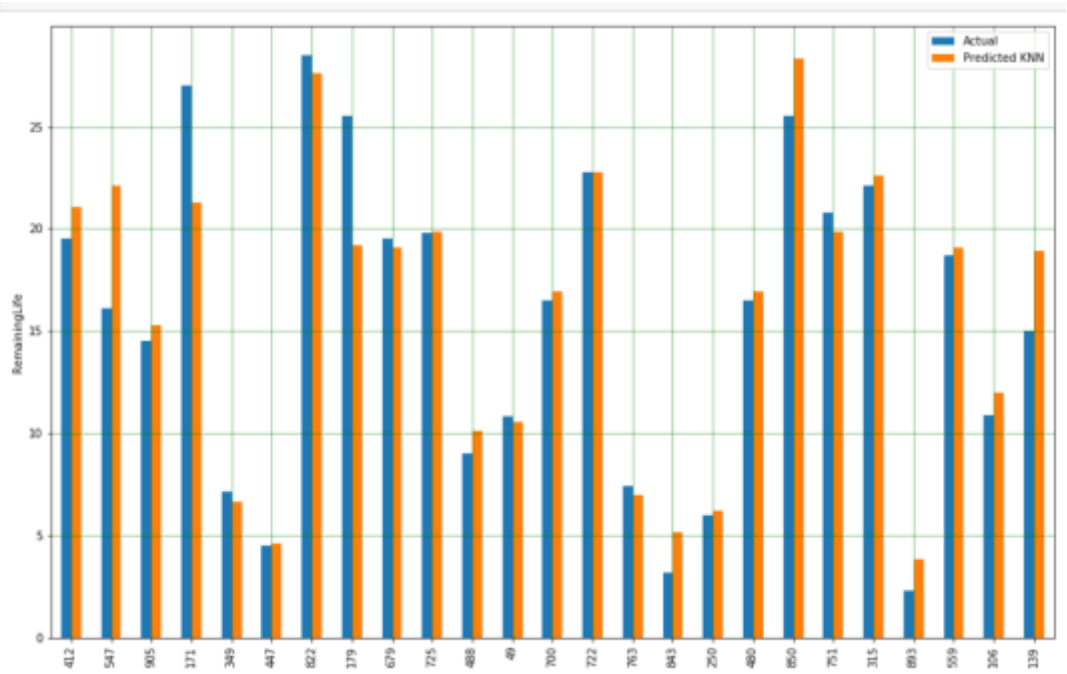
- Remaining Life

```
48]: #X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
#x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
#x_temp1=np.transpose(x_temp)
#y_temp=np.array([20])


#'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
#'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
#'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
#'Power', 'Driver', 'Distance', 'TotalOnTime'
x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

Parameters:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Prediction vs Actual:



Error metrics:

i. Mean square error = 9.94
ii. Mean absolute error= 2.14
iii. Root mean square error= 3.15

```
print("KNN mse = ",mse," & mae = ",mae," & rmse = ", sqrt(mse))
```

```
KNN mse =  9.946127779661015  & mae =  2.1444124293785314  & rmse =  3.15374821120219
```

Accuracy:

The accuracy is the model is around 90% with the above features.

```
In [54]: metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
Out[54]: 0.9035578193560151
```

When we reduce the number of features to 'Age', 'Cycle', 'Temperature', 'ChargeTime' 'RoadCondition', 'Driver', 'Distance', 'TotalOnTime' -> **the accuracy is about 80%**

```
from sklearn import metrics
```

```
metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
[76]: 0.8051576090813405
```

```
#X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

9. Neural Network (Multi layered Perceptron)

Libraries Used:

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
from sklearn import metrics
```

```
import matplotlib.pyplot as plt
```

Features used :

Dependant Variables:

- Age',
- 'Cycle'
- 'Temperature'
- 'ChargeTime'
- 'Cell voltage (V)',
- 'Current capacity'
- 'Crate'
- 'SOC'
- 'DOD',
- 'SelfDischargeRate'
- 'SOH'
- 'InternalResistance'

- 'RoadCondition'
- ’Power'
- 'Driver'
- 'Distance'
- 'TotalOnTime

Independent variable:

- Remaining Life

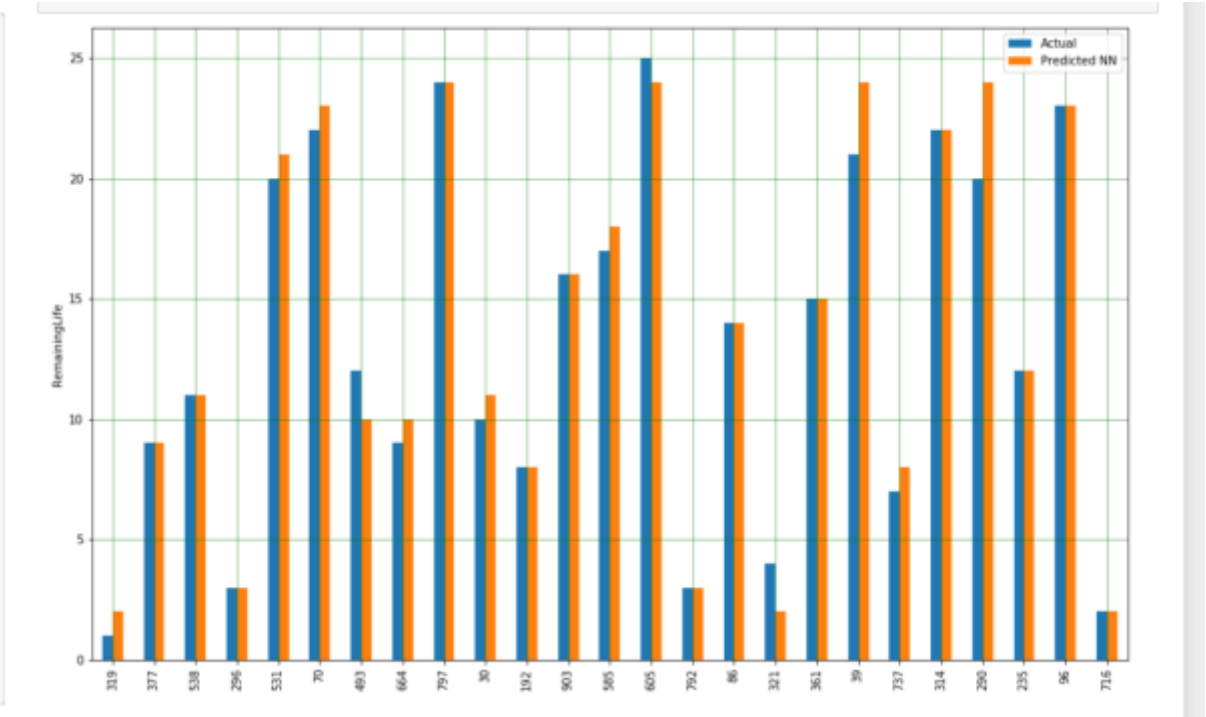
```
#'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
#'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
#'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
#'Power', 'Driver', 'Distance', 'TotalOnTime'
x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

Parameters:

```
: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(150, 100, 50), learning_rate='constant',
                learning_rate_init=0.001, max_iter=500, momentum=0.9,
                n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                random_state=123, shuffle=True, solver='adam', tol=0.0001,
                validation_fraction=0.1, verbose=False, warm_start=False)
```

i. Hidden layer size: 150,100,50
ii. max_iter=500
iii. activation = 'relu'
iv. Optimizer='adam'

Prediction vs Actual:



Accuracy:

The accuracy is the model is around 97% with the above features.

```
In [155]: metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
Out[155]: 0.976360252447635
```

When the number of features are reduced, the model gives an accuracy of about 40%

```
In [262]: metrics.r2_score(y_test, y_pred, sample_weight=None, multioutput='uniform_average')
Out[262]: 0.42162124374124

In [263]: #'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)',
          #'Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
          #'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition',
          #'Power', 'Driver', 'Distance', 'TotalOnTime'
          #x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)
          #x_temp1=np.transpose(x_temp)
          #y_temp=np.array([20])


          #X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
          x_temp=np.array([20,4,26,6,1,2,100,7]).reshape(-1,1)
          x_temp1=np.transpose(x_temp)
          y_temp=np.array([20])
```

10. Time Series Model

What is a time series?
A time series is usually modelled through a stochastic process Y(t), i.e. a sequence of random variables. In a forecasting setting we find ourselves at time t and we are interested in estimating Y(t+h), using only information available at time t.

How to validate and test a time series model?
Due to the temporal dependencies in time series data, we must ensure that training sets contains observations that occurred prior to the ones in validation sets.
A possible way to overcome this problem is to use a sliding window. This procedure is called **time series cross validation**

Models -> ANN, LSTM

Libraries used:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.optimizers import Adam
from keras import regularizers
import keras.backend as K
```

```
Using TensorFlow backend.
```

Scaling- StandardScaler

Parameters:

```
learning_rate = 0.01
training_epochs = 200
display_step = 200
factor = 0.10
batch = 128
seed = 7
beta_1 = 0.9
beta_2 = 0.999
epsilon = None
decay = 0.001
np.random.seed(seed)
dims = X_train_sc.shape[1]
```

1) Artificial Neural Network

```
In [24]:
model = Sequential()
model.add(Dense(128,
                input_shape=(dims,),
                activation='relu',
                kernel_regularizer=regularizers.l2(0.01)))
```

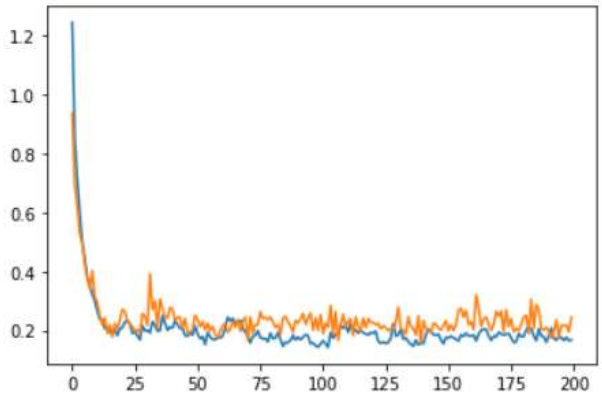
Summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               4224
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 14,593
Trainable params: 14,593
Non-trainable params: 0
_________________________________________________________________
```

Loss vs value loss after 200 epoch;

```
In [26]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
```

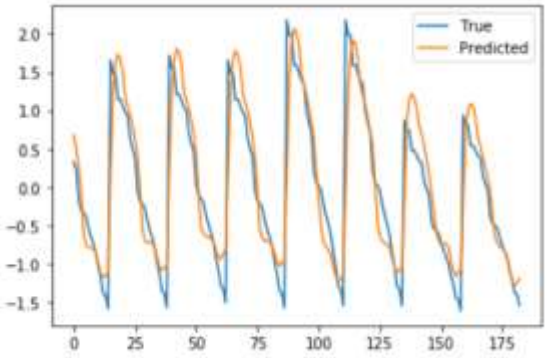
```
Out[26]: [<matplotlib.lines.Line2D at 0x1fcc8d81b00>]
```



True value vs predicted value

```
In [27]: y_pred = model.predict(X_test_sc)
         plt.plot(y_test_sc, label='True')
         plt.plot(y_pred, label='Predicted')
         plt.legend()
```

```
Out[27]: <matplotlib.legend.Legend at 0x1fccc1789b0>
```

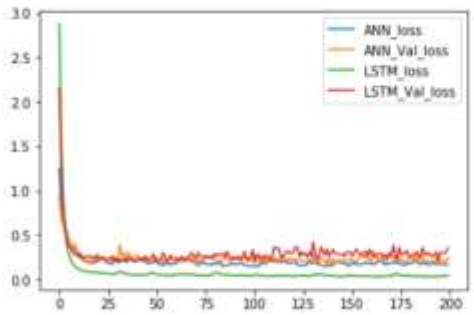


2) Long Short Tern Memory

```
model_lstm = Sequential()
model_lstm.add(LSTM(68, input_shape=(1, dims),
                    activation='relu',kernel_initializer='lecun_uniform',
                    return_sequences=False,
                    kernel_regularizer=regularizers.l2(0.01)))
```

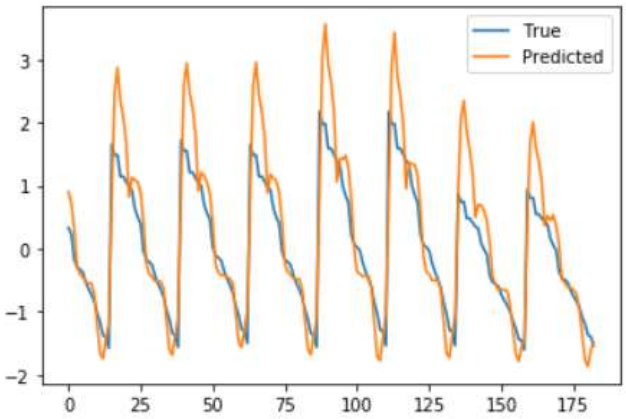
Comparison of LSTM and ANN loss after 200 epoch:

```
In [30]: plt.plot(history.history['loss'], label='ANN_loss')
         plt.plot(history.history['val_loss'], label='ANN_Val_loss')
         plt.plot(history_model_lstm.history['loss'], label='LSTM_loss')
         plt.plot(history_model_lstm.history['val_loss'], label='LSTM_Val_loss')
         plt.legend()

Out[30]: <matplotlib.legend.Legend at 0x1fcceffbf98>
```



True value vs predicted value

```
In [31]: y_pred_lstm = model_lstm.predict(X_tst_t)
         plt.plot(y_test_sc, label='True')
         plt.plot(y_pred_lstm, label='Predicted')
         plt.legend()

Out[31]: <matplotlib.legend.Legend at 0x1fcd1fef7b8>
```



5. TASK 2:

Provide user inputs , model tells if Battery will stay or not.

a) Import Required Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPClassifier


from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

from sklearn import metrics
```

b) Import the dataset

c) User Input:

1. Dependant variable:

   The user has two choices-

   i. Enter exact values:
   - Age',
   - 'Cycle'
   - 'Temperature'
   - 'ChargeTime'
   - 'Cell voltage (V)',
   - 'Current capacity’
   - 'Crate'
   - 'SOC'
   - 'DOD',
   - 'SelfDischargeRate'
   - 'SOH'
   - 'InternalResistance'
   - 'RoadCondition'
   - ’Power'
   - 'Driver'

- 'Distance'
- 'TotalOnTime

ii. Enter certain values:

- 'Age'
- 'Cycle'
- 'Temperature'
- 'ChargeTime'
- 'RoadCondition'
- 'Driver'
- 'Distance'
- 'TotalOnTime'

2. Independent Variable:
   Remaining Life

Based on the number of inputs, the program decides the dependant variables.

```
#X = dataset[['Age', 'Cycle', 'Temperature','ChargeTime','RoadCondition', 'Driver', 'Distance', 'TotalOnTime']].values
x_temp=np.array([20,4,26,4,1,2,500,20]).reshape(-1,1)

#'Age', 'Cycle', 'Temperature', 'ChargeTime', 'Cell voltage (V)', Current capacity (mAH rating)', 'Crate', 'SOC', 'DOD',
#'SelfDischargeRate', 'SOH', 'InternalResistance', 'RoadCondition', 'Power', 'Driver', 'Distance', 'TotalOnTime'
#x_temp=np.array([20,4,26,6,3.8,80,0.5,90,10,10,90,150,1,100,2,100,7]).reshape(-1,1)

x_temp1=np.transpose(x_temp)
y_temp=np.array([20])
```

d) Scaling –
   Using MinMaxScaler

e) Model-

   Creation of different models:

f) Check for accuracy :

   Check which model is most accurate and use that model for prediction.

g) Result:
   Check if the battery will last for the given amount of time based on the most accurate model's prediction.

CASE 1:

a) Input:

Dependant Variable (x_temp) -

- 'Age' = 20 (years)
- 'Cycle' = 4
- 'Temperature' = 26 (C)
- 'ChargeTime' = 4 (hrs)
- 'RoadCondition' = 1 (i.e Hilly)
- 'Driver' = 60 (ie Average Speed)
- 'Distance' = 100 (in km)
- 'TotalOnTime' = 5 (hours)

Independent Variable (y_temp) –

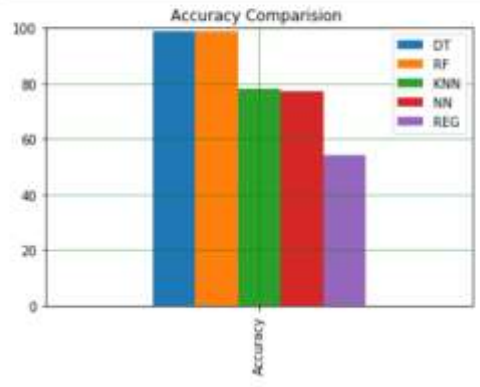
- Remaining life = 20 (yrs)

Interpretation-> With all the given conditions, will my battery last for 20 years?

b) Accuracy:

	DT	RF	KNN	NN	REG
Accuracy	98.932525	98.958008	78.016063	77.240633	54.059435

```
In [488]: df2.plot(kind='bar',figsize=(6,4))
          plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
          plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
          plt.ylim(0,100)
          plt.title('Accuracy Comparision ')
          plt.show()
```



We see that Decision Tree Model has the maximum accuracy so the program uses Decision Tree Algorithm to predict if the battery will last for 20 years or not.

c) Output:

The model predicted that according to the given conditions, the battery will last for about 14 years. So the battery will NOT last for 20 years.

```
df3 = pd.DataFrame({'Actual': y_temp, 'Predicted': pred,'Yes/No':res})
df3
```

']:

	Actual	Predicted	Yes/No
0	20	13.9	No

CASE 2:

a) Input:

Dependant Variable (x_temp) -

- Age', = 20 (years)
- 'Cycle' = 4
- 'Temperature' = 26 (C)
- 'ChargeTime' = 6 (hours)
- 'Cell voltage (V)', = 3.8 (V)
- 'Current capacity' = 80 (%)
- 'Crate' = 0.5
- 'SOC' = 80 (%)
- 'DOD', = 20 (%)
- 'SelfDischargeRate' = 10(%)
- 'SOH' = 90 (%)
- 'InternalResistance' = 150 (milli ohm)
- 'RoadCondition' = 2 (City)
- 'Power' = 100 (kWh)
- 'Driver' = 100 (Rash)
- 'Distance' =100 (km)
- 'TotalOnTime = 7 (hrs)

Independent Variable (y_temp) –

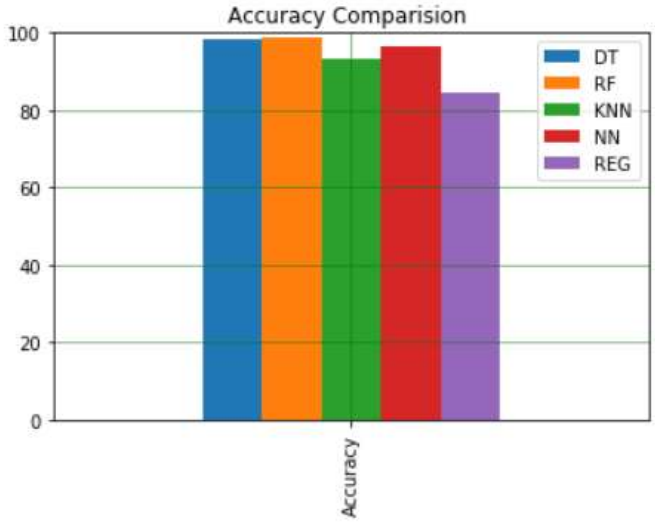
- Remaining life = 20 (yrs)

Interpretation-> With all the given conditions, will my battery last for 20 years?

b) Accuracy:

Out[79]:

	DT	RF	KNN	NN	REG
Accuracy	98.338365	98.968175	93.13513	96.757294	84.385064



We see that Random Forest Model has the maximum accuracy so the program uses Random Forest Algorithm to predict if the battery will last for 20 years or not.

c) Output:

The model predicted that according to the given conditions, the battery will last for about 19.6 years, which is around 20 years. So, the battery WILL last for around 20 years.

```
df3 = pd.DataFrame({'Actual': y_temp, 'Predicted': pred,'Yes/No':res})
df3
```

]:

	Actual	Predicted	Yes/No
0	20	19.6	Yes

CONCLUSION

- The remaining life of a battery depends on various factors such a s Temperature, Charge time, age of the battery, the charge-discharge cycle of the battery, etc.

- By being able to predict the remaining life time of a battery and the amount of time left before the battery discharges will be able to help notify EV drivers about when to charge the vehicle.

- Random forest regressor and Decision tree regressor are the best models to predict the remaining life in both conditions:
  i. Average conditions provided
  ii. Exact conditions provided

- K Nearest neighbour will also work well in both conditions
- Neural network model works well only when exact conditions are provided
- Multiple linear regression works well when the features are linearly dependent on remaining life time

- **Reasons behind less accuracy:**
  - **Need more data**: We need to have a huge amount of data to get the best possible prediction.
  - **Bad assumptions**: We made the assumption that this data has a linear relationship, but that might not be the case. Visualizing the data may help you determine that.
  - **Poor features**: The features we used may not have had a high enough correlation to the values we were trying to predict.

References

- https://towardsdatascience.com/predicting-battery-lifetime-with-cnns-c5e1faeecc8f
- https://pandas.pydata.org/
- https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- https://www.tensorflow.org/guide/keras/train_and_evaluate
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://www.sciencedirect.com/science/article/pii/S2314728817300144
- FRESCO PLAY (MACHINE LEARNING)
- UDEMY COURSE – DATA ANALYTICS UNSING PYTHON