REPORT

ON

"Machine Learning for Solar Energy Prediction"

Objective: To Build ML model to predict the power production of a solar power station from the measurements of a set of weather features.

Department of Electronics and Communication Engineering

B.M.S COLLEGE OF ENGINEERING

(Autonomous College Affiliated to Visvesvaraya Technological University, Belgaum)

Bull Temple Road, Basavanagudi, Bangalore-560019

SUBMITTED BY:

Student Name	USN
Subramanya K	1BM18EC154
Venkatesh Subramanya Iyer Giri	1BM18EC177
Roshan Nayak	1BM18EC122
K Shivanithyanathan	1BM18EC063
K S Eshwar Subramanya Prasad	1BM18EC062
Priyanshu M	1BM18EC103
Tanay Somnani	1BM18EC165

Under the Guidance of

Renganayaki S

(System Specification Engineer, Nokia)

1. Introduction:

Renewable energy resources offer many advantages over traditional energy resources such as fossil fuel but the energy produced by them fluctuates with changing weather conditions. Companies need accurate forecasts of energy production in order to have the right balance of renewable and fossil fuels available. If accurate PV forecasting is lacking, any unexpected fluctuations in solar energy capacity may have significant impacts on the daily operations and physical health of the entire grid and may negatively affect the quality of life of the energy consumers. Power forecasts typically are derived from numerical weather prediction models, but statistical and machine learning techniques are increasingly being used to produce more accurate forecasts.

Project aims at developing machine learning models which include regression, deep neural networks to evaluate the performance and obtain a statistical model to achieve the objective.

2. Data Collection

The Training dataset was taken from Kaggle AMS 2013-2014 Solar Energy Prediction Contest. The dataset is of size 2.84 GB. The data are in netCDF4 files with each file holding the grids for each ensemble member at every time step for a particular variable. Each netCDF file contains the latitude-longitude grid and time step values as well as metadata listing the full names of each variable and the associated units.
Each netCDF4 file contains the total data for one of the model variables and are stored in a multidimensional array.
The solar energy was directly measured by a pyranometer at each Mesonet site every 5 minutes and summed listed in each column. Solar Output from solar Farms are represented in kWh units.

3. Data processing

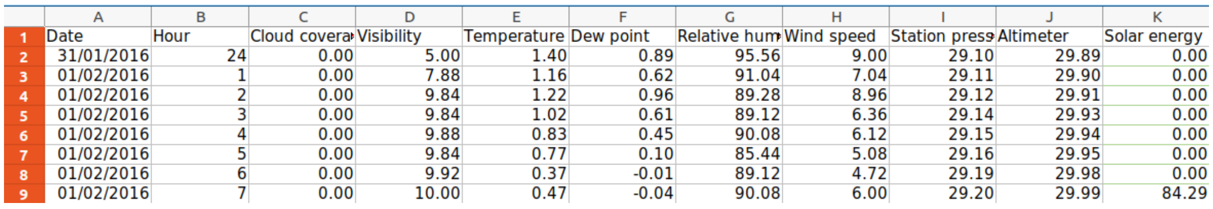
Weather features were contained in netCDF4 files stored in a multidimensional array. NetCDF libraries available in R language were used to convert netCDF4 format file into csv file for future data clearing and processing.
Data processing and Cleaning were performed in R due to the large size of data to be processed.

Handling Missing values:
- The dataset had many missing values for weather features and for solar output for daily and hourly dataset.
- Missing values were replaced by using Linear Interpolation technique considering nearest not-null values.
- Weather features from netCDF4 file having many missing values were dropped from the dataset.

Daily dataset was prepared by averaging over the entire day (for sunlit hours) to represent each data by single data instance.

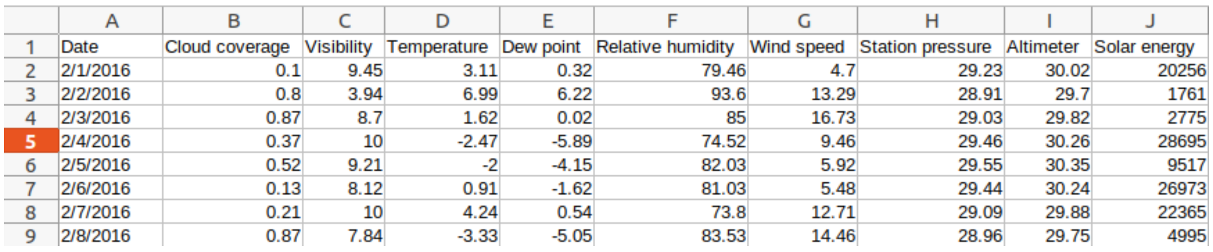
The feature Sky Condition was of type String which was converted to a numeric value. The corresponding processed feature is called Cloud coverage and is in % of the sky which is covered by clouds.

Hourly Dataset: This dataset contains weather features for every hour of the day and corresponding solar energy output in kWh.



	A	B	C	D	E	F	G	H	I	J	K
1	Date	Hour	Cloud covera	Visibility	Temperature	Dew point	Relative hum	Wind speed	Station press	Altimeter	Solar energy
2	31/01/2016	24	0.00	5.00	1.40	0.89	95.56	9.00	29.10	29.89	0.00
3	01/02/2016	1	0.00	7.88	1.16	0.62	91.04	7.04	29.11	29.90	0.00
4	01/02/2016	2	0.00	9.84	1.22	0.96	89.28	8.96	29.12	29.91	0.00
5	01/02/2016	3	0.00	9.84	1.02	0.61	89.12	6.36	29.14	29.93	0.00
6	01/02/2016	4	0.00	9.88	0.83	0.45	90.08	6.12	29.15	29.94	0.00
7	01/02/2016	5	0.00	9.84	0.77	0.10	85.44	5.08	29.16	29.95	0.00
8	01/02/2016	6	0.00	9.92	0.37	-0.01	89.12	4.72	29.19	29.98	0.00
9	01/02/2016	7	0.00	10.00	0.47	-0.04	90.08	6.00	29.20	29.99	84.29

Daily Dataset: This dataset contains average daily weather features and sum of solar energy generation power in kWh.



	A	B	C	D	E	F	G	H	I	J
1	Date	Cloud coverage	Visibility	Temperature	Dew point	Relative humidity	Wind speed	Station pressure	Altimeter	Solar energy
2	2/1/2016	0.1	9.45	3.11	0.32	79.46	4.7	29.23	30.02	20256
3	2/2/2016	0.8	3.94	6.99	6.22	93.6	13.29	28.91	29.7	1761
4	2/3/2016	0.87	8.7	1.62	0.02	85	16.73	29.03	29.82	2775
5	2/4/2016	0.37	10	-2.47	-5.89	74.52	9.46	29.46	30.26	28695
6	2/5/2016	0.52	9.21	-2	-4.15	82.03	5.92	29.55	30.35	9517
7	2/6/2016	0.13	8.12	0.91	-1.62	81.03	5.48	29.44	30.24	26973
8	2/7/2016	0.21	10	4.24	0.54	73.8	12.71	29.09	29.88	22365
9	2/8/2016	0.87	7.84	-3.33	-5.05	83.53	14.46	28.96	29.75	4995

4. Dimensionality Reduction

Dimensionality reduction refers to the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data.

Principal Component Analysis (PCA)

We used Principal Component Analysis (PCA) to find the direction of combination of features which captures the variability of the features. PCA was used to provide low dimensional visualization (2 D) from high dimensional space.

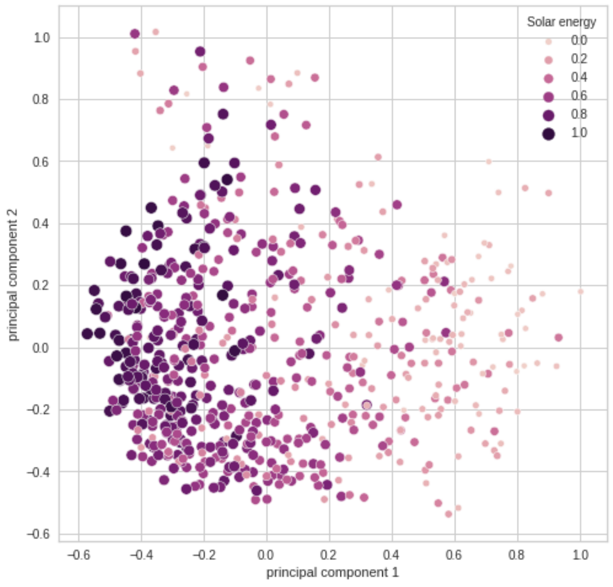
Principal Component Analysis (PCA)

```
scaler = MinMaxScaler()
#scalar = StandardScaler()

df = pd.DataFrame(scaler.fit_transform(df.iloc[:,1:]), columns=df.columns[1:])

x = df[['Cloud coverage','Visibility','Temperature','Dew point','Relative humidity','Wind speed',
       'Station pressure','Altimeter']]
pca = PCA().fit(x)
z = pca.transform(x)
print(z)
plt.scatter(z[:,0],z[:,1])
```

Below is the 2 component PCA visualization (colour coded on normalized Solar output)



Two component PCA were used as predictors to fit Multiple regression model and model performance was evaluated.

```
from sklearn.model_selection import KFold

crossvalidation = KFold(n_splits=5, shuffle=True)
model = LinearRegression().fit(df_pca.iloc[:,0:2],df_pca.iloc[:,-1])

n_scores = cross_val_score(model,df_pca.iloc[:,0:2],df_pca.iloc[:,-1], scoring="r2"
                           ,cv=crossvalidation, n_jobs=1)

print(str(n_scores))
print('Mean r2_score: %.3f' % (np.mean(n_scores)))
```

```
[0.47294389 0.52498537 0.50114031 0.54833157 0.59913496]
Mean r2_score: 0.529
```

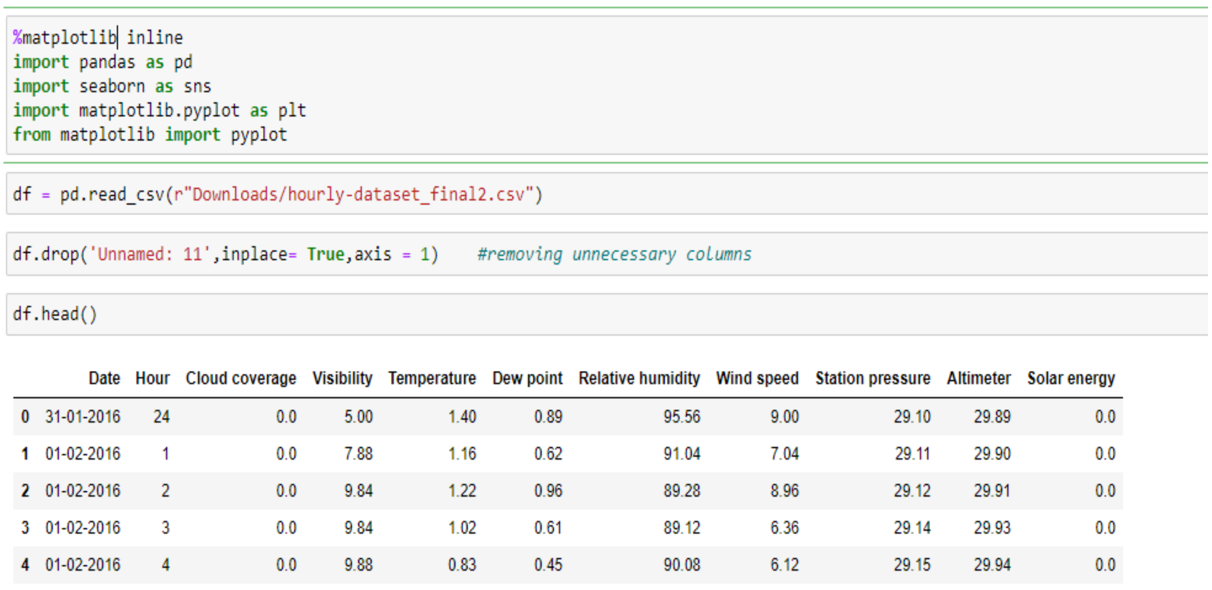
Model Evaluation:

Test set/cross validation	R2 score
Using test set (size = 30%)	0.64
Using cross-validation (k=5)	0.529

5. Exploratory Data Analysis

In statistics, exploratory data analysis is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

We are using a dataset ,but in this dataset we have considered for the whole day,even during night hours



```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
```

```
df = pd.read_csv(r"Downloads/hourly-dataset_final2.csv")
```

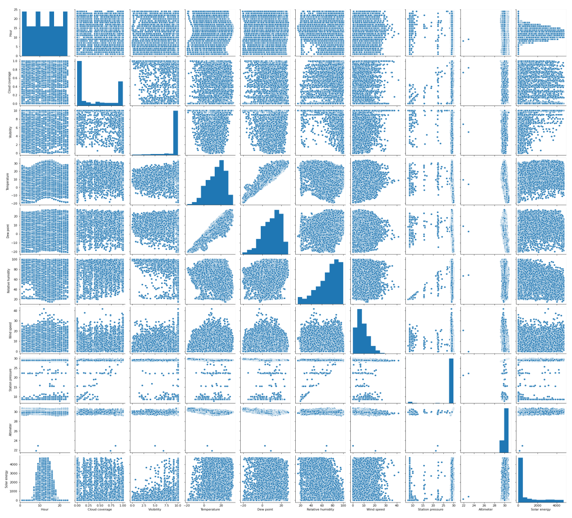
```
df.drop('Unnamed: 11',inplace= True,axis = 1)     #removing unnecessary columns
```

```
df.head()
```

	Date	Hour	Cloud coverage	Visibility	Temperature	Dew point	Relative humidity	Wind speed	Station pressure	Altimeter	Solar energy
0	31-01-2016	24	0.0	5.00	1.40	0.89	95.56	9.00	29.10	29.89	0.0
1	01-02-2016	1	0.0	7.88	1.16	0.62	91.04	7.04	29.11	29.90	0.0
2	01-02-2016	2	0.0	9.84	1.22	0.96	89.28	8.96	29.12	29.91	0.0
3	01-02-2016	3	0.0	9.84	1.02	0.61	89.12	6.36	29.14	29.93	0.0
4	01-02-2016	4	0.0	9.88	0.83	0.45	90.08	6.12	29.15	29.94	0.0

Pairplot

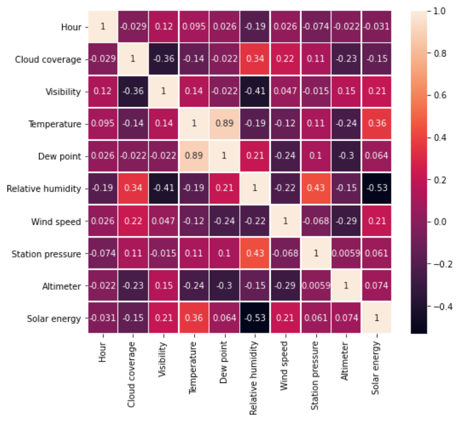
Pairplot visualizes given data to find the relationship between them where the variables can be continuous or categorical. Plot pairwise relationships in a data-set. Pairplot is a module of seaborn library which provides a high-level interface for drawing attractive and informative statistical graphics



Heatmap

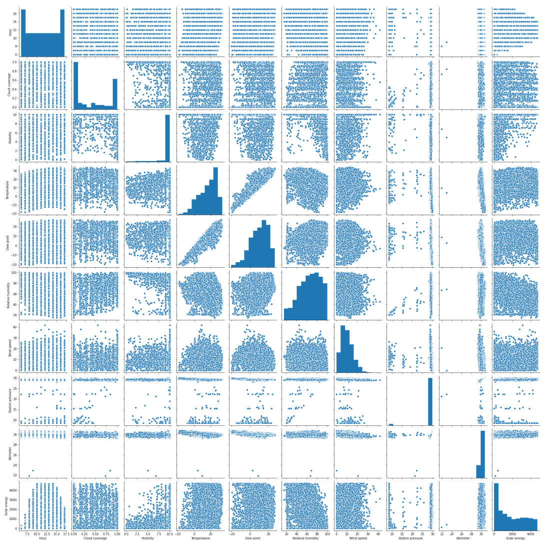
A heatmap is a graphical representation of data that uses a system of color-coding to represent different values

```
correlation_matrix = df.corr()
pyplot.figure(figsize=(8,7))
sns.heatmap(correlation_matrix, annot=True ,linewidths=0.5)
plt.show()
```

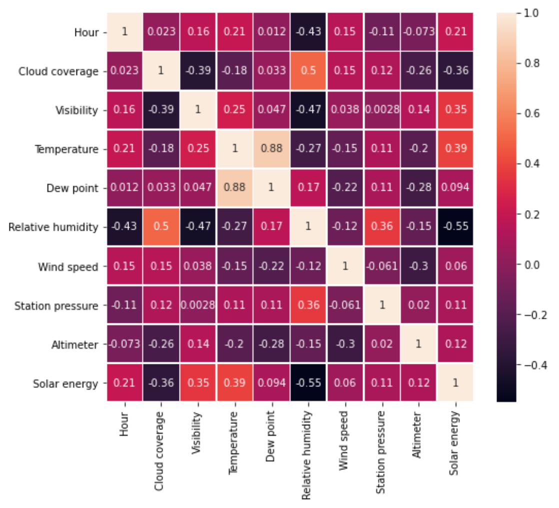


Now we will remove the night hours and plot the heatmap and pairplots for this dataset.

In this dataset,we had an extra column ‘random’ which we used for training.We drop it from the table and proceed with our analysis



Pairplot



Heat Map

6. Model Selection and Evaluation

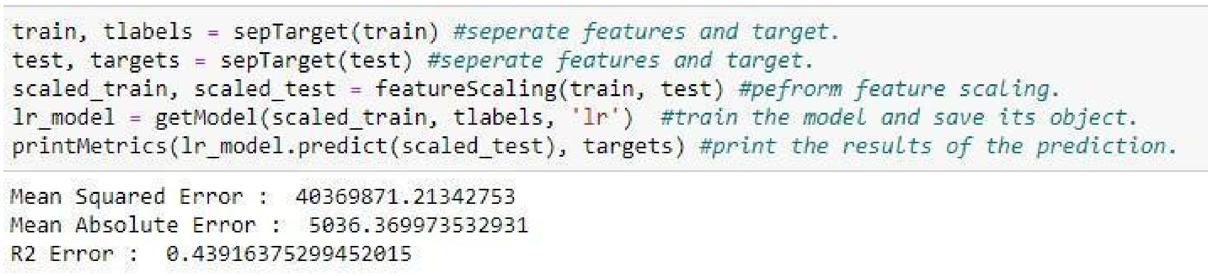
Linear Regression:

Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable.

During the model building phase first the data had to be split into the train and the test dataset. Test dataset was allotted 30% of the total dataset. Then the target and the independent features for both train and test dataset were separated. Next step would be feature scaling. Linear regression models usually work well when the features are in the same range. This prevents the model training from getting dominated by the features with larger values. Hence here we have used MinMaxScaler from Sklearn library to achieve this. MinMaxScaler scales the values of all the features in the range 0 to 1.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

This is the formula for the MinMaxScaler. After feature scaling the model was trained on the preprocessed, ready to train dataset. The trained models instance is assigned to the lr_model variable. This object is used to predict and test the MSE, MAE, and R2 Error of the model.

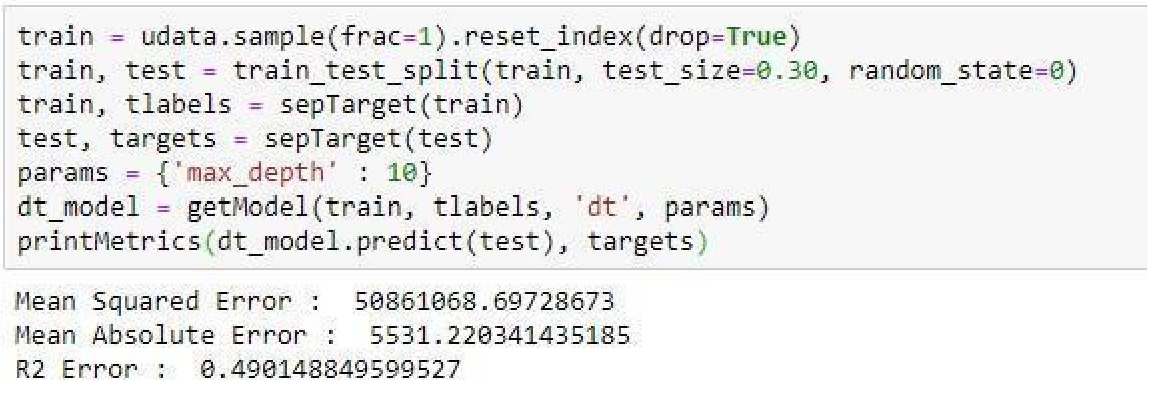


```python
train, tlabels = sepTarget(train) #seperate features and target.
test, targets = sepTarget(test) #seperate features and target.
scaled_train, scaled_test = featureScaling(train, test) #pefrorm feature scaling.
lr_model = getModel(scaled_train, tlabels, 'lr')  #train the model and save its object.
printMetrics(lr_model.predict(scaled_test), targets) #print the results of the prediction.
```

```
Mean Squared Error :  40369871.21342753
Mean Absolute Error :  5036.369973532931
R2 Error :  0.43916375299452015
```

Decision Tree:

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs and utility.



```python
train = udata.sample(frac=1).reset_index(drop=True)
train, test = train_test_split(train, test_size=0.30, random_state=0)
train, tlabels = sepTarget(train)
test, targets = sepTarget(test)
params = {'max_depth' : 10}
dt_model = getModel(train, tlabels, 'dt', params)
printMetrics(dt_model.predict(test), targets)
```

```
Mean Squared Error :  50861068.69728673
Mean Absolute Error :  5531.220341435185
R2 Error :  0.490148849599527
```

Support Vector Regressor:

SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data. We will be using the rbf kernel which works best for regression.

```
svr=SVR(kernel='rbf')
model_train(X_train,X_test,Y_train,Y_test,svr,"SVR")
Mse of SVR :
SVR 1674274.1390601671
```

Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.
We used a model with 1000 trees.

```
rf=RandomForestRegressor(n_estimators=100)
model_train(X_train,X_test,Y_train,Y_test,rf,"Random Forest")
Mse of Random forest is
Random Forest 465756.90855626593
```

As it is evident that random forest performs the best out of all the model ,we will use rf and improve its accuracy

7. Time Series Forecasting

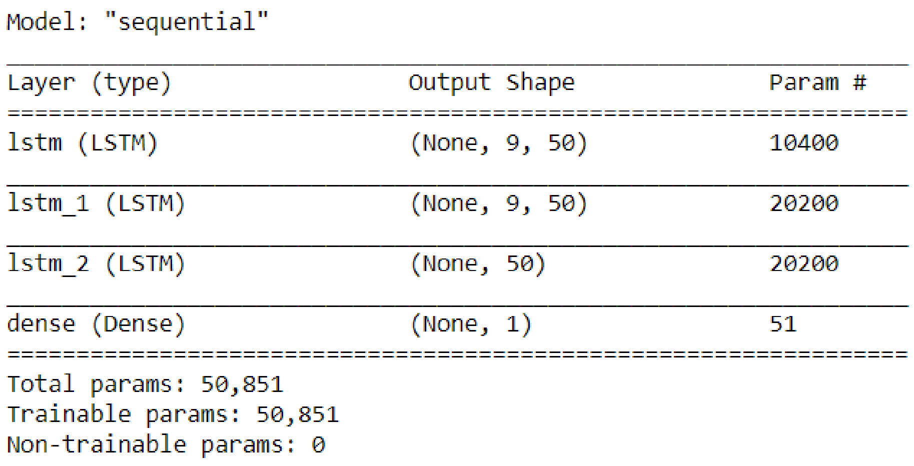
Forecasting in general is the process of making predictions based on past and present data and most commonly by analysis of trends.
In Machine Learning we Forecast Time series data by training the model on past values.
Using the technique we can forecast future data which can be helpful.

In this project we have forecasted solar energy data by analyzing past solar energy values.
We use LSTM,i.e, Long short term memory. It is a type of RNN but it solves many drawbacks which RNN has.

Preprocessing:
We have used slicing to train the model ,basically we take the solar energy data and suppose our sequence length is 10. We would take data from 0-9 as input
And output would be 10th data ,the model has to predict 10th index data.

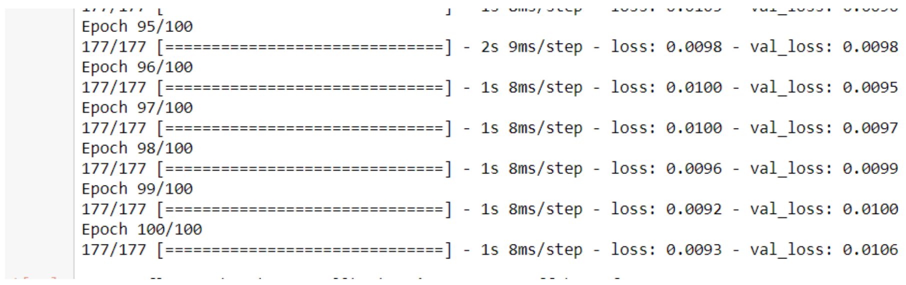
For the next input we increase input by one,i.e, we give 1-10th as input and 11th as output and so on.

Model architecture: We have used 3 LSTMs and one Dense layer for our model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 9, 50)             10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 9, 50)             20200
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
```

Training the model: Epoch we used is 100,val_loss and loss is given below



```
Epoch 95/100
177/177 [==============================] - 2s 9ms/step - loss: 0.0098 - val_loss: 0.0098
Epoch 96/100
177/177 [==============================] - 1s 8ms/step - loss: 0.0100 - val_loss: 0.0095
Epoch 97/100
177/177 [==============================] - 1s 8ms/step - loss: 0.0100 - val_loss: 0.0097
Epoch 98/100
177/177 [==============================] - 1s 8ms/step - loss: 0.0096 - val_loss: 0.0099
Epoch 99/100
177/177 [==============================] - 1s 8ms/step - loss: 0.0092 - val_loss: 0.0100
Epoch 100/100
177/177 [==============================] - 1s 8ms/step - loss: 0.0093 - val_loss: 0.0106
```

8. Application Development:

We are developing a web application to help the user to utilise our machine learning model and predict the total solar energy generation at a particular location, if they want to set up a solar energy plant. Currently we have a web page where the user has to input a city name and they will get the total solar energy generated at that city. The model can predict solar energy at a particular latitude and longitude, but for now we have structured the page such that the user enters just the city name. Each city will have a particular latitude and longitude assigned to it, and this data is used in the model.

Data collection for web application:

The only data that we collect right now from the user is the city name. In the upcoming activities, we are planning to implement a map so that the user can pinpoint the exact location with ease. Along with the city name, we need more data for the model, for which the web APIs (Application Programming Interface) will help out.

Application Programming Interfaces (APIs) used:

There are two APIs that are used here:

a. Weatherbit: Weatherbit API is used to retrieve current weather observations from over 47,000 live weather stations. It is a free API which can be used to retrieve current weather data, but with a limitation of 500 calls per day, which is sufficient for the current project plan. This API provides us data features like Latitude, Longitude, Timezone, Station, and weather-related features used in the model, like Temperature, Pressure, Wind Speed, Cloud Coverage, Visibility and Dew Point. The data is retrieved from the API by using Flask (code explained below). We provide the city name, from which the latitude and longitude is retrieved and used in the API, and the API provides the weather information required.
b. Opentopodata: It is an elevation API, which is used for the "Altimeter" feature. It provides 100 locations per request and a maximum of 1000 calls per day for free. The elevation is provided using the city name or the latitude and longitude. Since they use different databases for different regions of the world, the database we use has to be changed with respect to the regions of the world we use the application in.

Unit conversion of collected data:

The data collected from the APIs are having different units compared to the dataset used in the models. Hence we need to change the units of the data collected from the API. Some of the features are having the same units, hence no changes are required. Given below are the features with their unit conversions:

Feature	Dataset Unit	API Unit
Cloud Coverage	% Range	% Range
Visibility	Miles	KM
Temperature	Degree Celsius	Degree Celsius
Dew Point	Degree Celsius	Degree Celsius
Relative Humidity	% Range	% Range
Wind Speed	MPH	m/s
Station Pressure	Inches of Mercury	Mb (Millibars)
Altimeter	Inches of Mercury	Meters
Solar Energy	kW/day	kW/day

Conversions:

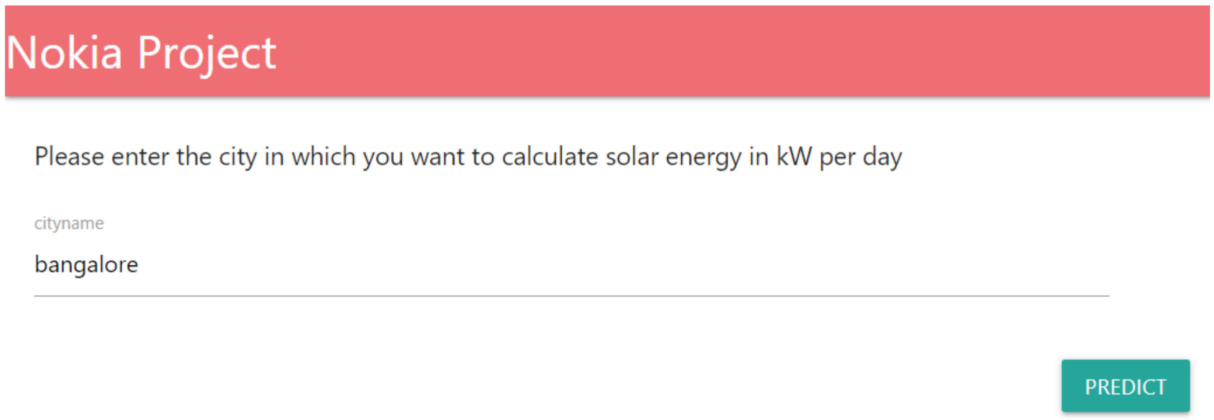
Feature	Conversions
Cloud Coverage	No Conversions
Visibility	KM -----> Miles 1KM = 0.621371 Miles
Temperature	No Conversions
Dew Point	No Conversions
Relative Humidity	No Conversions
Wind Speed	m/s -----> MPH 1m/s = 2.23694 MPH
Station Pressure	Mb -----> inchHg 1mb = 0.02953 inchHg
Altimeter	Meters -----> InchHg Formula to convert actual altitude to altimeter readings: Alt = (Pmb − 0.3) × (1+(((1013.25^0.190284× 0.0065)/ 288) × (hm/(Pmb − 0.3)^0.190284)))^ 1/0.190284 (Pmb is station pressure (in mb) and hm is elevation (in meters))
Solar Energy	No Conversions

Flask:

Flask is a micro web framework written in Python. Flask is classified into a micro-framework that means it has little to no dependencies on external libraries. We have used flask to post the name of the city from the web page to the flask server, and the server will find the latitude and longitude of the city. This latitude and longitude is the one which is taken as common and assigned by Google Maps. Using the latitude and longitude we will find other features from the above mentioned APIs.

Construction of Web Application:

Below is the homepage (index.html



Please enter the city in which you want to calculate solar energy in kW per day

bangalore

Code for the homepage:

```
    <form class="col s12" method="POST" action="{{url_for('home')}}">
      <div class="row">
        <div class="input-field col s12 l6">
          <input id="cityname"  class="active" type="text" name="cityname"/>
          <label for="cityname">cityname</label>
        </div>
      </div>

      <div class="row" style="display: flex; justify-content: center; align-items: center;">
        <input type="submit" class="btn" value="predict" ></input>
      </div>
    </form>
```

Here we will post the name of the city to the server when the user submits it, using Flask.

```
from flask import Flask,render_template,request
import pickle
from flask.helpers import flash
import requests
model = pickle.load(open(linearmodel.pkl','rb'))
app = Flask(__name__)
@app.route('/')
def man():
    return render_template('index.html')
```

Above code will render the index.html when the user goes to the route “www.solardetect.com /" . The name solardetect is given as an example. When the user submits he is directed to the “www.solardetect.com/predict” page.

```
def getCorrectUnit(X):
   X[0] = X[0]/100
   X[1] = 0.621371 *X[1] #0.621371
   X[5] = 2.23694 *X[5]   #2.23694
   pmb = X[6]
   hm = X[7]
   X[6] = 0.02953 *X[6] #0.02953
   pmbmin0_3 = pmb-0.3
   pmbmin0_3toPower0_190284 = pmbmin0_3**0.190284
   hmBYpmbmin0_3toPower0_190284 = hm/pmbmin0_3toPower0_190284
   rightEq = (1+0.0000842288*hmBYpmbmin0_3toPower0_190284)**5.25530260032
   X[7] = pmbmin0_3*rightEq
   X[7]= 0.02953*X[7]
   return X
```

The above function is the unit conversion function, which performs the conversions as mentioned above. Meanwhile the function given below is the route decorator (home() function), which will be called.

```
def getWeatherData(lat,long):
   urlWeatherBit = "https://api.weatherbit.io/v2.0/current?lat="+str(lat)+                "
&lon="+str(long)+"&key=f6bba80d9ad242b4b82bfb54364059ea"
   res = requests.request("GET", urlWeatherBit)
   resFromWebit = res.json()
   visibility = float(resFromWebit['data'][0]['vis'])
   cloudcoverage = float(resFromWebit['data'][0]['clouds'])
   temperature = float(resFromWebit['data'][0]['temp'])
   dewpoint = float(resFromWebit['data'][0]['dewpt'])
   relativehumidity = float(resFromWebit['data'][0]['rh'])
   windspeed = float(resFromWebit['data'][0]['wind_spd'])
   stationpressure = float(resFromWebit['data'][0]['pres'])
   urlAltitude ="https://api.opentopodata.org/v1/aster30m?locations="+str(lat) + "
,"+str(long)+""
   resAltimeter = requests.request("GET", urlAltitude).json()
   altimeter=float(resAltimeter['results'][0]['elevation'])
```

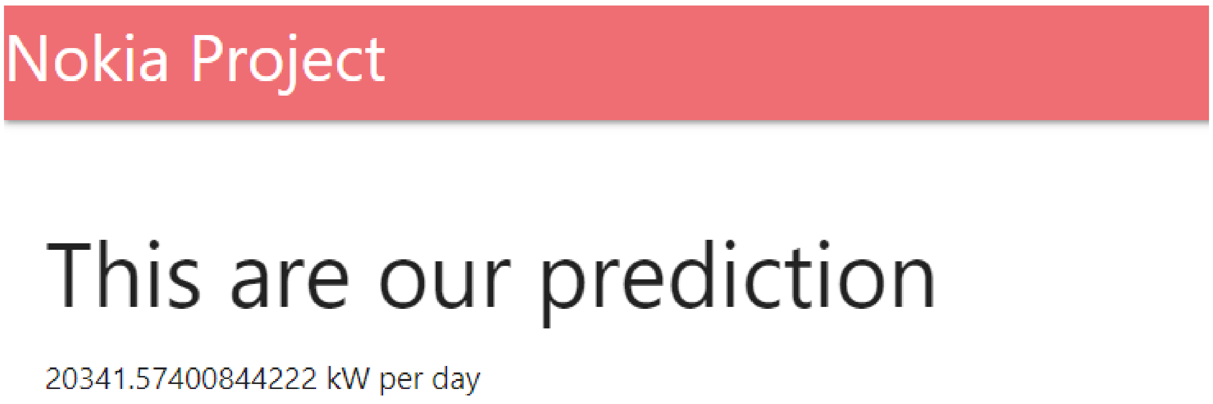
```
    todayData = date.today().strftime("%d-%m-%Y")
    day,month,year = tuple(str(todayData).split('-'))
    year = int(year)
    month = int(month)
    day = int(day)
    X  = [cloudcoverage,visibility,temperature,dewpoint, relativehumidity,   windspeed,
stationpressure,altimeter,year,month,day]
    return X
```

The above function is the data collection function from the APIs. The weatherbit and opentopodata APIs are called and the data for the features are retrieved.

```
@app.route('/predict',methods=['POST'])
def home():
    cityname = request.form['cityname']
    lat,long = latandlong(cityname)
    X = getWeatherData(lat,long)
    X = getCorrectUnit(X)
    X = list(X)
    Xmax = [1.0,10.0,28.18,25.02,97.85,24.83,29.87,30.67,2017,12,31]
    Xmin = [0.0,1.15,-16.06,-18.72,21.25,1.03,8.59,29.48,2016,1,1]
    XN= [x-xmn for x, xmn in zip(X, Xmin)]
    XD = [xmx-xmn for xmx, xmn in zip(Xmax, Xmin)]
    scaled =  [xn / xd for xn, xd in zip(XN, XD)] #scaled=(X−Xmin)/(Xmax−Xmin)
    scaled = list([scaled])
    pred = model.predict(scaled)
    print(pred)
    return render_template('after.html',data=pred)
```

Here we will take the city name from the index.html webpage. Then we will call latandlong (city name) to get the latitude and longitude. Then we will use it to find the other features required for the model to predict solar energy in the given location. After collecting the data from the APIs, unit conversion is done, as mentioned above. Later, the minimum and maximum values of the features are initialised, Because we trained our linear model with normalised value. So that the model uses this data and the collected data and predicts the answer.

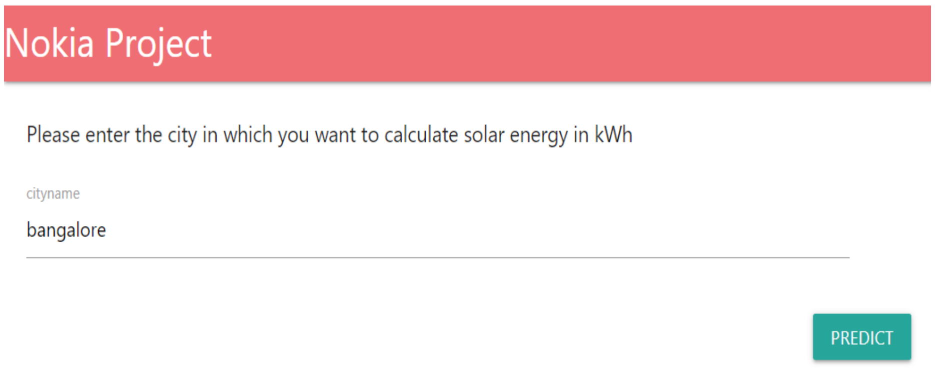
The predicted output is then sent to “after.html” to print the output


Nokia Project

This are our prediction

20341.57400844222 kW per day

This is the results web page which displays the output that is predicted by the machine learning model. The output is in the total solar energy that can be produced in a day.
We have also trained a Random Forest model for hourly dataset. For random forests there is no need of scaling. So we will skip that part then everything other than that is the same, In random forest we will get the output in kW per hour(kWh).That is the total amount of solar energy that is predicted for the current hour.
Index.html page.

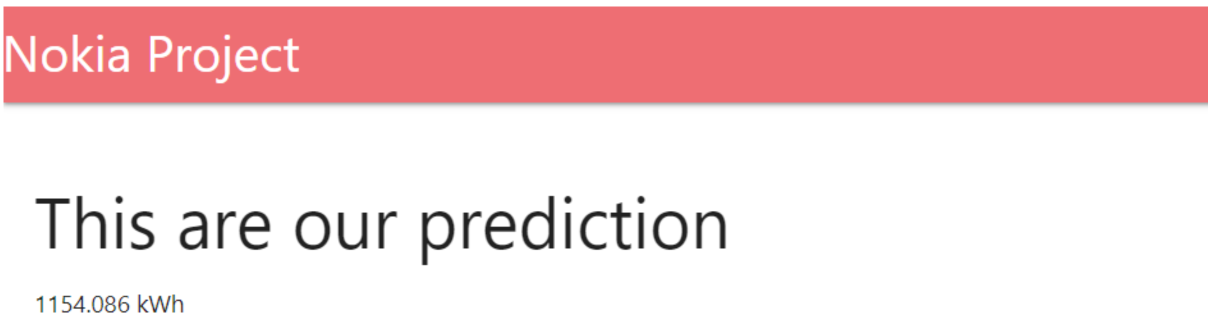

Nokia Project

Please enter the city in which you want to calculate solar energy in kWh

cityname

bangalore

After.html.


Nokia Project

This are our prediction

1154.086 kWh

Here we will get the predicted output of the given location in kWh.

9. Planned Activities

- Work on Boosting techniques and Ensemble techniques for improving model performance
- Work on Developing model using neural networks
- Application Development: Data visualization of Hourly prediction from the model and provide useful summary for energy generation per year, per month and per day estimation.