B.M.S COLLEGE OF ENGINEERING

(An Autonomous College under VTU, Belagavi)

Bull Temple Road, Bangalore - 560 019

Project Report-2020-21

On

"Machine Learning for Solar Energy Prediction"

Submitted as a part of Alternate Assessment for the Elective course

MACHINE LEARNING

offered by

ELECTRONICS AND COMMUNICATIONS ENGINEERING

In association with

NOKIA NETWORKS, BANGALORE

Submitted By

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

Nokia Mentor: Renganayaki S,
Designation: System Specification Engineer, Nokia

FIC: Dr.Suma MN
Professor

Table of Contents

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 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 is 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 files into csv files 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 the 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.

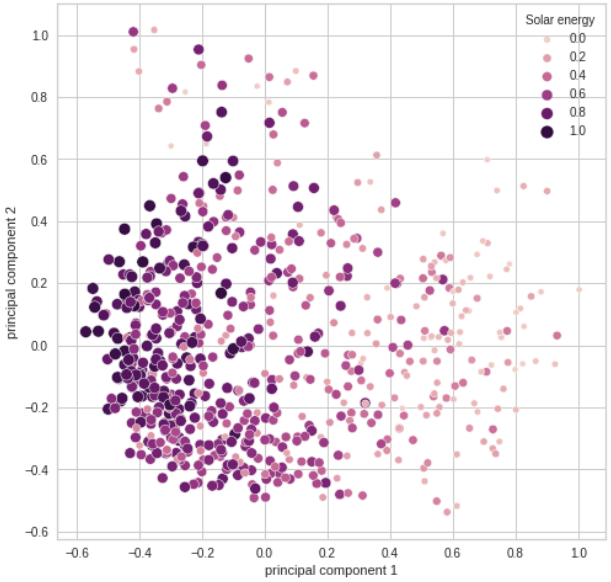
Principal Component Analysis (PCA)

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 the Multiple regression model and model performance was evaluated.

Model Evaluation:

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

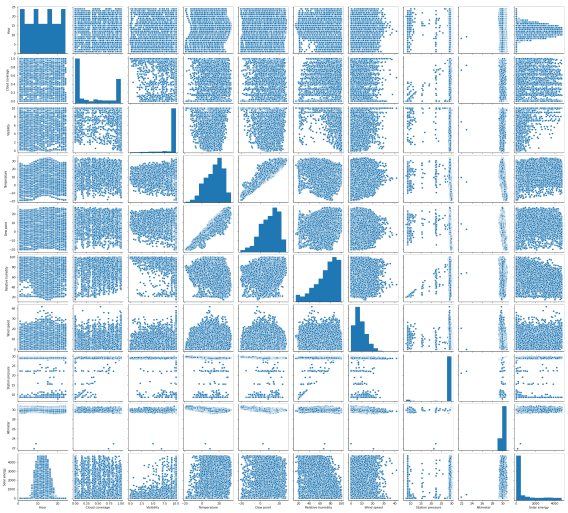
5. Exploratory Data Analysis

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

Pairplot

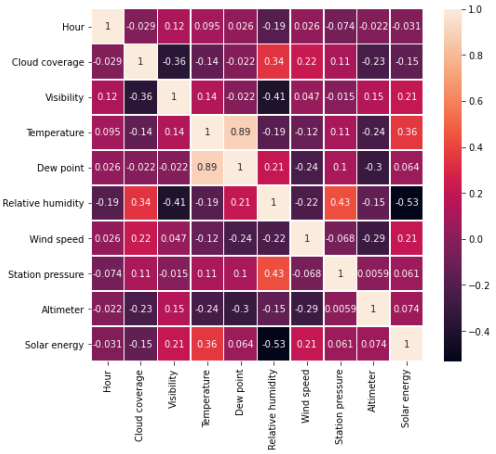
Pairplot visualizes given data to find the relationship between them where the variables can be continuous or categorical. Plot pairwise relationships in a data-set.



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()
```



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.

```
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.

```
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
```

Hourly Dataset:

XGBOOST Regressor:
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting technique. During the model training it makes use of software and hardware optimization techniques like parallel processing, tree-pruning, handling missing values, and regularization to avoid overfitting/bias. to yield superior results using less computing resources in the shortest amount of time.

```
xgb = xg.XGBRegressor(
        n_estimators = 800,
        seed = 123, eta=0.025,
        max_depth=8,
        subsample=0.75)
xgb.fit(train_X, train_y)
pred = xgb.predict(test_X)
r2 = r2_score(test_y, pred)
print("R2 : % f" %(r2))
```

R2 score of XGBOOST is 0.836636

LightGBM Regressor:
Light GBM is a fast, distributed, high-performance based on gradient boosting technique and decision tree algorithm. It splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise.

```
gbm = lgb.LGBMRegressor(**lgb_params)
gbm.fit(train_X, train_y)
pred = gbm.predict(test_X)
r2 = r2_score(test_y, pred)
print("R2 : % f" %(r2))
```
R2 score of LightGBM is 0.825761

RandomizedSearchCV:

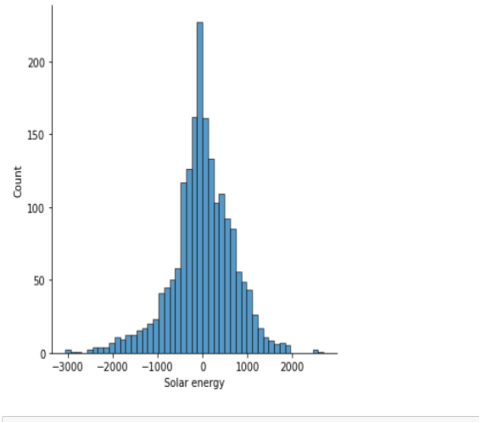
RandomizedSearchCV is used for hyper parameter tuning,we create a grid and input values for all parameters we want to experiment with then the algorithm returns the best model with minimum error from those combinations of parameters we entered

Random Forest Regressor:
Here RandomizedSearchCV is used for Random Forest Regressor

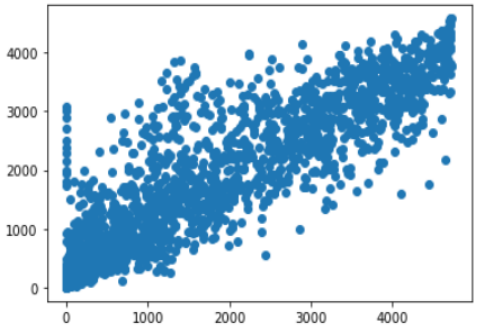
```
In [64]: n_e=[int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
         max_f=['auto','sqrt']
         max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
         min_samples_split = [2, 5, 10, 15, 100]
         min_samples_leaf = [1, 2, 5, 10]

In [65]: random_grid = {'n_estimators': n_e,
                        'max_features': max_f,
                        'max_depth': max_depth,
                        'min_samples_split': min_samples_split,
                        'min_samples_leaf': min_samples_leaf}

In [67]:
         from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor()

In [68]: rf_rcv=RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 5

In [69]: rf_rcv.fit(X_train,y_train)
```

The randomizedsearchcv chooses the best rf model with the lowest error, from the parameters we entered.

Gives an **R2 Score** of 0.7855505262319216



Error Distribution plot



Scatter Plot of y_pred and y_test

Gradient Boosted Regression(GBR):

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

In [77]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

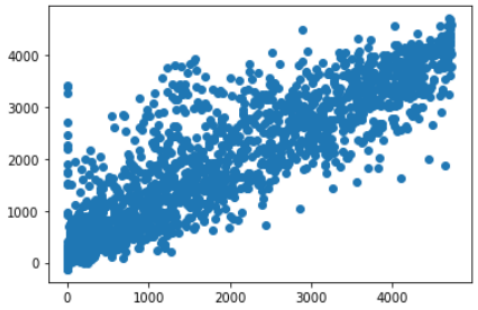
In [78]:
```
gb = GradientBoostingRegressor()


learning_rate = [0.001, 0.01, 0.1, 0.2]


n_estimators=list(range(500,1000,100))

max_depth=list(range(4,9,4))

min_samples_split=list(range(4,9,2))

min_samples_leaf=[1,2,5,7]

max_features=['auto','sqrt']

param_grid = {"learning_rate":learning_rate,
              "n_estimators":n_estimators,
              "max_depth":max_depth,
              "min_samples_split":min_samples_split,
              "min_samples_leaf":min_samples_leaf,
              "max_features":max_features}

gb_rs = RandomizedSearchCV(estimator = gb, param_distributions = param_grid)
```

In [79]:
```
gb_rs.fit(X_train,y_train)
```

R2 Score of 0.79

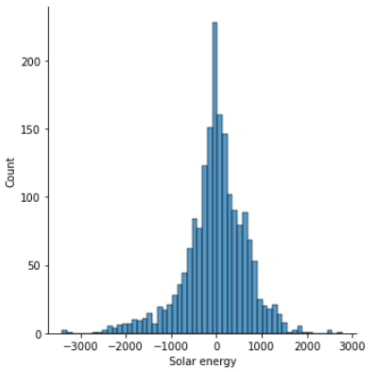
Scatter Plot

```
In [83]: plt.scatter(y_test,pred)
Out[83]: <matplotlib.collections.PathCollection at 0x2727f76ab48>
```



Distill Plot

```
In [84]: sns.displot(y_test-pred)
Out[84]: <seaborn.axisgrid.FacetGrid at 0x2727f77bec8>
```



Stacking in Machine Learning:

Stacking technique has Base estimators and a final estimator. Here we are introducing another layer of learning in which the model can learn how to use the outputs of the base models to predict the final output instead of blindly taking the average of the outputs of the base estimators. Hence this technique is better than other ensemble techniques like random forest.

Here we have trained two stacking technique based models. First one using Linear regression as the final estimator and the second one using Neural Network as the final estimator.

Stacking using Linear Regression as final estimator:

```
meta_model = LinearRegression()
stacking_model = StackingRegressor(
    estimators=base_models,
    final_estimator=meta_model,
    cv=5,
    n_jobs=-1,
    verbose=2)
stacking_model.fit(train_X, train_y)
stack_preds = stacking_model.predict(test_X)
r2 = r2_score(test_y, stack_preds)
print("R2 : % f" %(r2))
```

Result:
R2 Score = 0.831950.

Stacking using Neural Network as final estimator:

```
meta_model_nn = MLPRegressor(random_state=123, max_iter=500,
learning_rate_init=0.01, verbose=True, early_stopping=True, validation_fraction=0.15,
hidden_layer_sizes=(4, 16, 1))
Stacking_model_nn = StackingRegressor(estimators=base_models,
final_estimator=meta_model_nn, cv=5, n_jobs=-1,verbose=2)
stacking_model_nn.fit(train_X, train_y)
stack_preds_nn = stacking_model_nn.predict(test_X)
r2 = r2_score(test_y, stack_preds_nn)
print("R2 : % f" %(r2))
```

Result: **R2 Score** = **0.844822**

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.

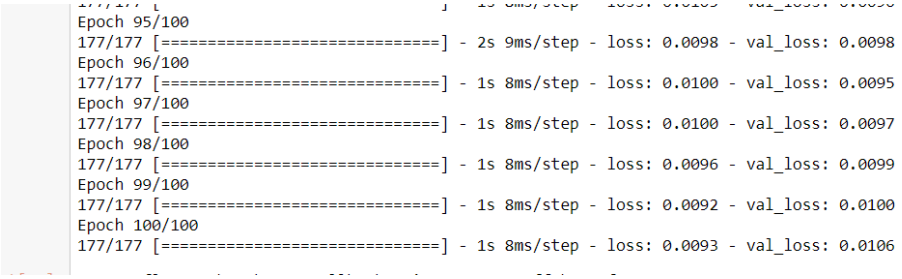
In this project we have forecasted solar energy data by analyzing past solar energy values. We use LSTM,i.e, Long short term memory. It is a type of RNN but it solves many drawbacks which RNN has.

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 utilise our machine learning model and predict the total solar energy generation at a particular location, which can be useful for setting up a solar energy plant. We have created a web application where the user has to input latitude and longitude of the location and a particular date (for historical solar energy data) and they will be provided with the total solar energy generated at that city with all other information including energy savings and plots, cost analysis, environmental impact, and nearest solar services. Energy savings analysis includes information on energy produced per hour and per day. Cost analysis shows the saving in electricity in terms of cost. Environmental impact includes the amount of carbon dioxide that can be reduced by producing energy by means of solar panels. Plots are created for hourly solar energy output for the input date, and daily solar energy output for 30 days previous to the input date.

Technology Stack:

1. Flask: The micro web framework written in Python, is used for the configuration of web servers for the application. It helps in collecting data from APIs and performing functions needed for analysis of the data.

2. Materialize CSS: The CSS framework used for smooth experience of web applications, which uses the principles of Material Design. A single underlying responsive system allows for a more unified user experience.

3. Leaflet.js: It is an open source JavaScript library for mobile-friendly interactive maps. The map data is taken from OpenStreetMap and Imagery is created through Mapbox.

4. Chart.js: A simple yet flexible JavaScript charting library. It provides different types of maps that are required. This library is used for plotting the daily and monthly solar energy predictions.

5. Heroku: A cloud platform as a service supporting several programming languages. It is basically used for hosting web applications. Since the web application here is small enough, Heroku supports hosting with smooth experience.

Application Programming Interfaces (APIs) used:

There are four 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.
c. Opencagedata: This API is used to get the latitude and longitude of a particular location which is entered by the user. The API takes in the city name as a parameter, and pinpoints the exact latitude and longitude, along with providing other information like nearest city (in case the user enters a town or village), and other information related to the terrain.
d. Timezonedb: Timezonedb is used to get the current time and timezone of the location entered. Just like opencagedata, the user gives the city name as a parameter and the API outputs the timezone and current time, along with other information related to time. This is needed, since the WeatherBit API takes in the current time of the server where it's based, and hence we might get wrong answers from the model for current hour solar energy prediction.

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:

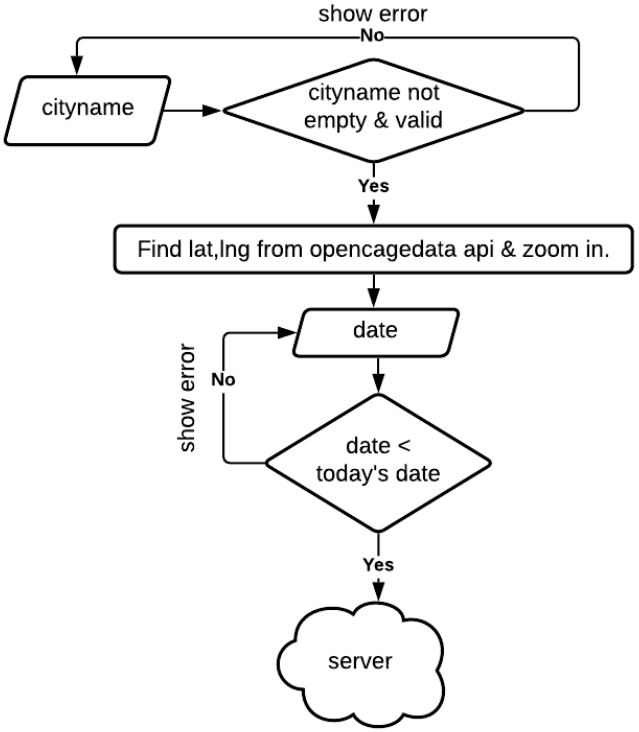
Feature	Dataset Unit	API Unit
Cloud Coverage	% Range	% Range
Visibility	Miles	KM
Temperature	Degree Celsius	Degree Celsius
Dew Point	Degree Celsius	Degree Celsius
Relative Humidity	% Range	% Range
Wind Speed	MPH	m/s
Station Pressure	Inches of Mercury	Mb (Millibars)
Altimeter	Inches of Mercury	Meters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

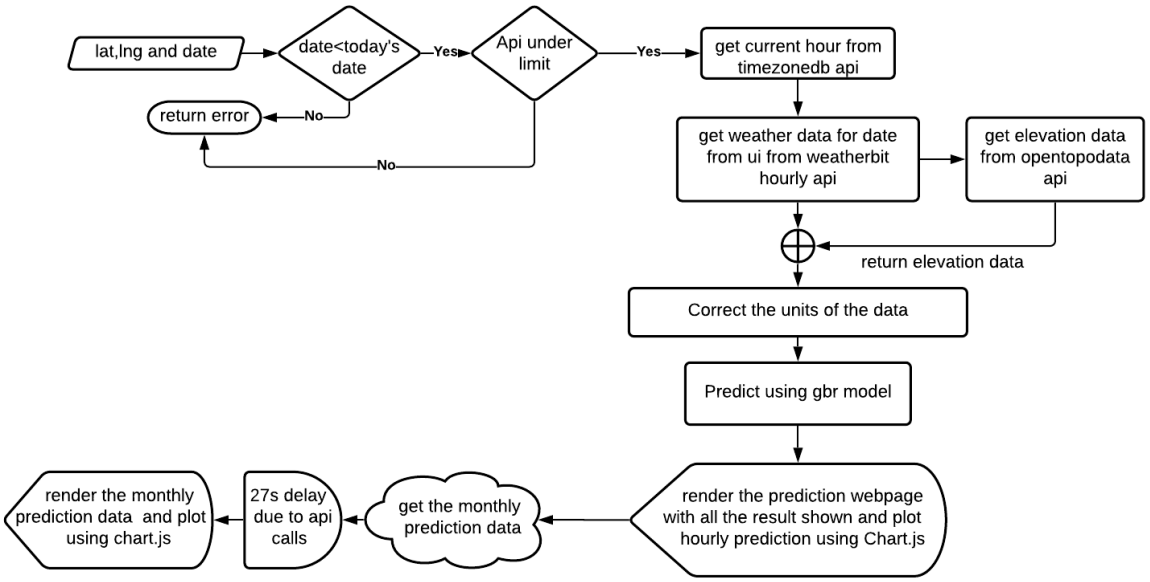
Web application flowchart :

Input Page:



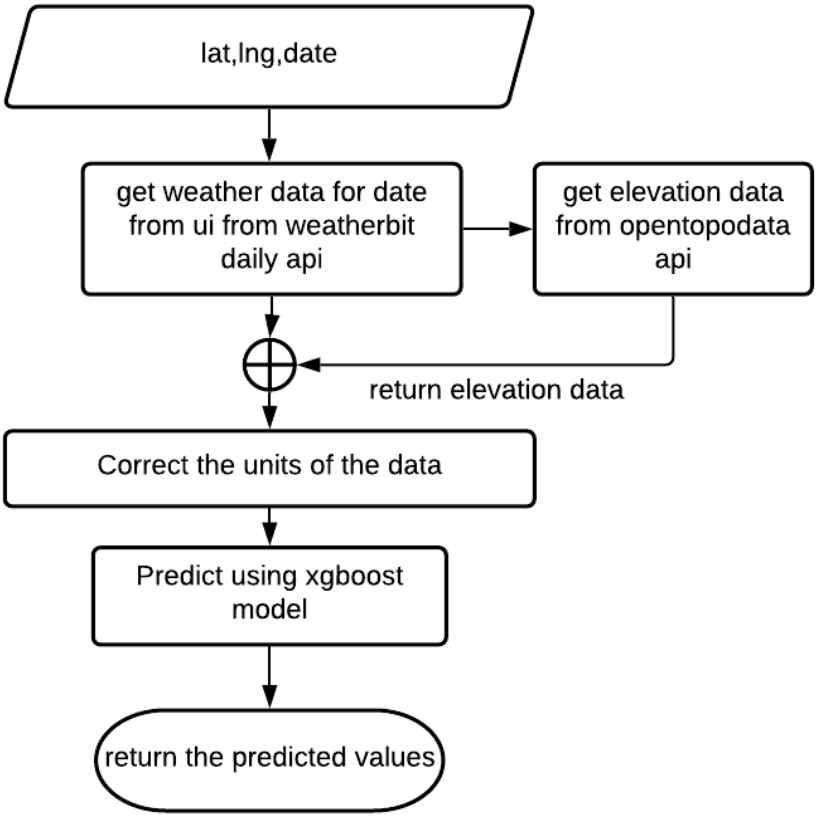
The following section consists of the functional process of the web application. The user has to enter the city name of his/her choice, and then the first error handling is performed. If the city name is wrong or the city name is left blank, the error message: "Enter complete name of your city"/"Enter valid city name" is given out. Opencagedata API uses the city name to give out the latitude and longitude of the location and zooms in on the map. Next the user has to provide a date for prediction, where the next error handling is performed. If the date is of the future it gives out an error message "Please select a date less than today's date". The date, latitude and longitude is then sent to the server, where the backend process will take place.

Result Page:



So the server will receive a request from the ui with latitude, longitude and date as parameters. First we will do a server side error check by checking whether the date is less than today's date. Because we can't get the weather data for dates which are greater than today's date. Then we check whether the api reached the request limit or not because we are using a trial version of the api which allows only 500 calls per day. We will send an error message as "please select a date less than today's date" or " The weather bit api has reached its request limit. Please try after sometimes" in JSON format if either of these conditions are false.

Next we will create data for prediction of solar output per every hour on that given day. Then we get the current hour from the timezonedb api which takes latitude and longitude as parameters. We have used api to get the current hour information because If we get the hour data from the python time library it will give an hour at the location where the server is located. Since our server is located in the US we were getting hours in the USA. Later in the application we found out this huge bug and shifted to Timezonedb api to get the current hour at a given location. Then we call weatherbit hourly api by passing latitude, longitude and start date and enddate as parameters to get the hourly data between those dates. We selected the end date as the date provided by ui and the start day as one day before the date provided by the application. From the result from the api we will select required features like and get the elevation data from opentopodata api. Then merge hour , elevation and weather data extracted from weatherbit api. Then we correct the units of this data so it matches the training dataset units , and we will predict using the gbr model. Then we will render the prediction page with all the info and we will plot the Hourly prediction using chart.js we will use hours in a day for x axis and solar energy per hour in y axis. Hourly data collection and prediction takes less time that's why we will collect it first and render the result page.



After rendering the result page we will call the server to get monthly prediction results to plot using chartjs. For this we created an api endpoint /monthly which takes latitude, longitude and start and end date as arguments. Here we will want a prediction of 30 days before the selected date. But since we are using a trial version of api we can't get data for more than one day. So we iterate through every alternate day for 30 days. So finally we will get the data for a month less that the selected date But we will only have 15 days alternative days data since If we want to get data for 30 days we should call api for 30 times which will take more than 60s to do this. So we are calling 15 times for alternative days which will take 15-30s. After collecting data we will add elevation data from opentopodata, Here we will not include data for it because it's a daily prediction model. We will predict this using xgboost and return with corresponding dates with JSON format. Then the second plot will be rendered by using data provided by the monthly endpoint.

9. Future Scope & Conclusion

- The study in this project can be expanded to develop accurate prediction systems to aid the users or organizations with questions: about affordability, weather and light patterns, angle and tilt, government incentives; which are currently scattered across the web.
- The resulting high-performance forecasting methods along with the knowledge of basic household load profiles will allow for more accurate quantification of active power reserve requirements.
- This forecasting approach is a key step towards the development of a data-driven decision-making framework for power system operation.
- Similar study and conclusion can be drawn on large wind farms' energy forecasting models and derive the impact study on its integration and load balancing at the city and the national grid solutions.