Solar Sight

Green! Solar! Sustainable!

Description: Estimates the amount of solar energy generated and the electricity demands using machine learning.

APK file: github.com/Agnij-Moitra/solar-sight/raw/app/solarsight-release.apk

Source Code: github.com/Agnij-Moitra/solar-sight/tree/main

Vision

It was rightly said by James Cameron, "The nation that leads in renewable energy will lead the world.". Our vision is to make solar energy accessible to the common people and add value to the lives of millions of people who rely on clean energy.

Problem

Due to the various uncertainties in weather conditions and solar radiation, Solar Power Plants cannot optimize their systems to incorporate the variance of atmospheric conditions. Similarly, for individual users they cannot estimate the amount of electricity produced.

Solution

Solar Sight is able to estimate the amount of electrical power generated in an area per 100 ft^2. It uses machine learning using python in the backend and Flutter for the app. Further it also predicts the energy demands of any Indian state using time-series estimation.

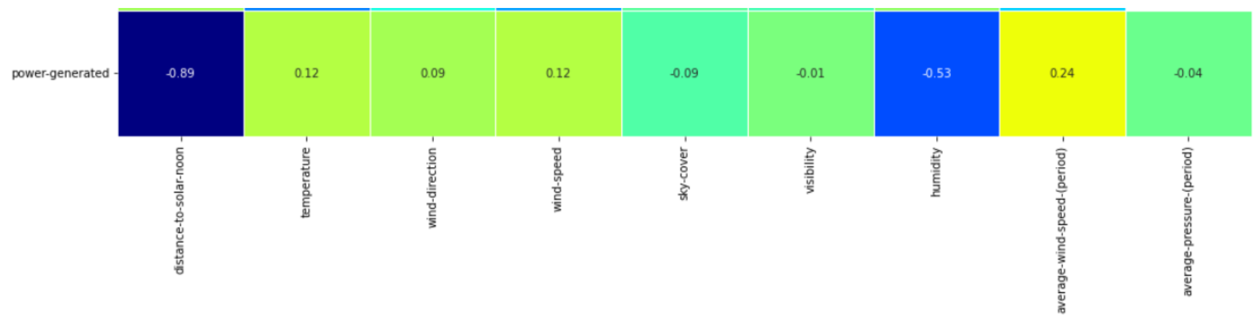
Target Clients and Commercial Viability

Solar Sight aims to help the existing individuals and corporations who use green energy calculate the amount of electricity generated and also find the electricity demands of a day. This is especially helpful for the solar power plant companies. It helps them improvise and optimize their power management systems.

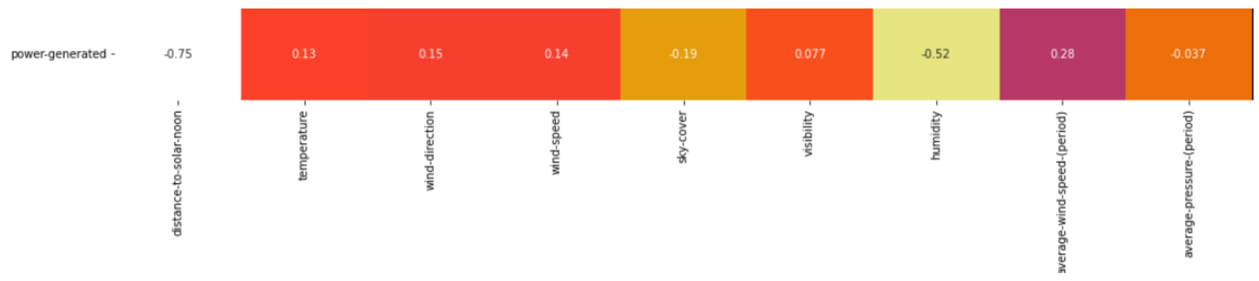
Implementation

Backend

- Using Spearman and Pearson correlation operators on our datasets we concluded that pressure, humidity, wind direction, wind speed and the radial distance to solar noon are correlated to the amount of electrical power generated.



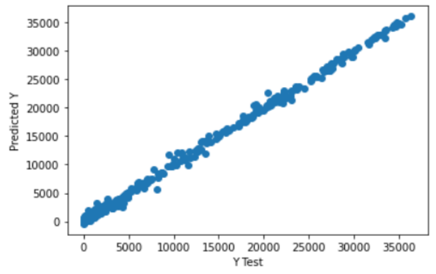
Spearman Correlation



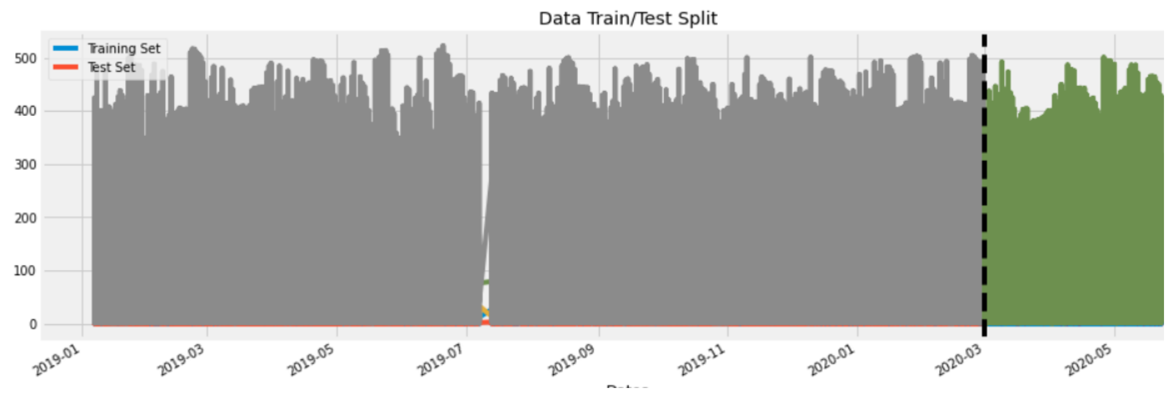
Pearson Correlation

- Then we made a SciKit Learn pipeline for the above featureset which included preprocessing using Mean Imputers, to fill the missing values, and OneHotEncoder for numeric transformations. After that we used XGradient Boosting Ensemble as our model. We had an accuracy of 96%, Mean Absolute Score of 240 and Mean Squared Error of 426. Then we used pickle to convert our model to a binary file in order to decrease the time complexity. Moreover, we used Open Weather Maps and IP Geolocations API to fetch the above features for any location, as defined in supplementary.py.

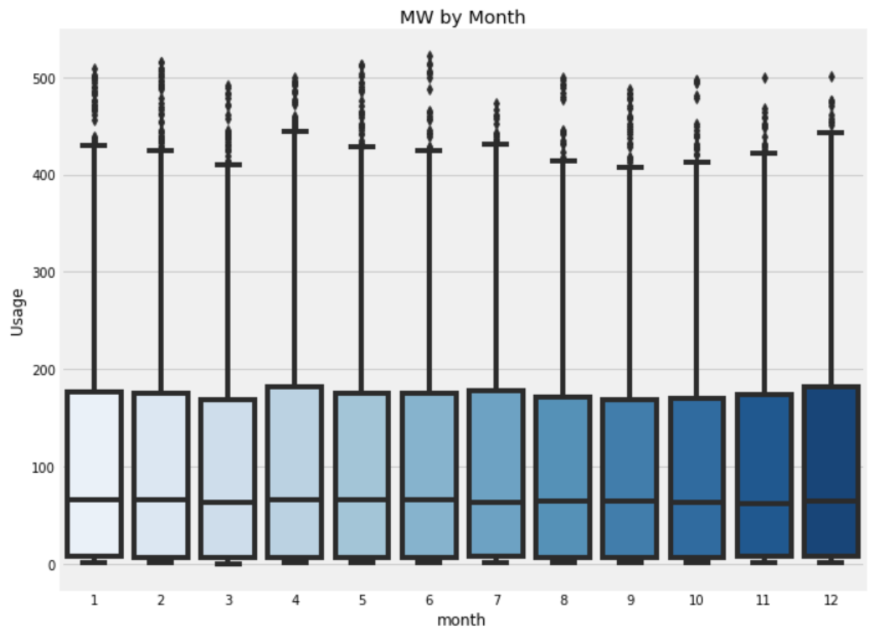
```
MAE: 240.5830164670638
MSE: 181832.9705986999
RMSE: 426.41877374090825
```



- For the time-series analysis to find the electricity demands we used the state wise daily electricity demands from 1 January 2019 to 5 December 2020.

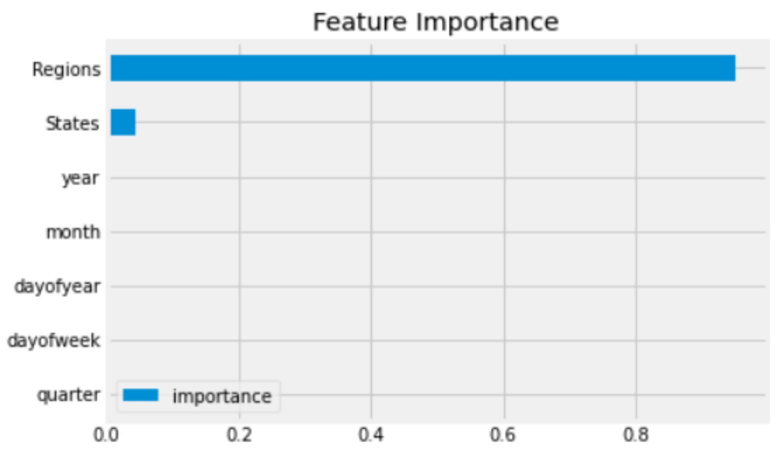


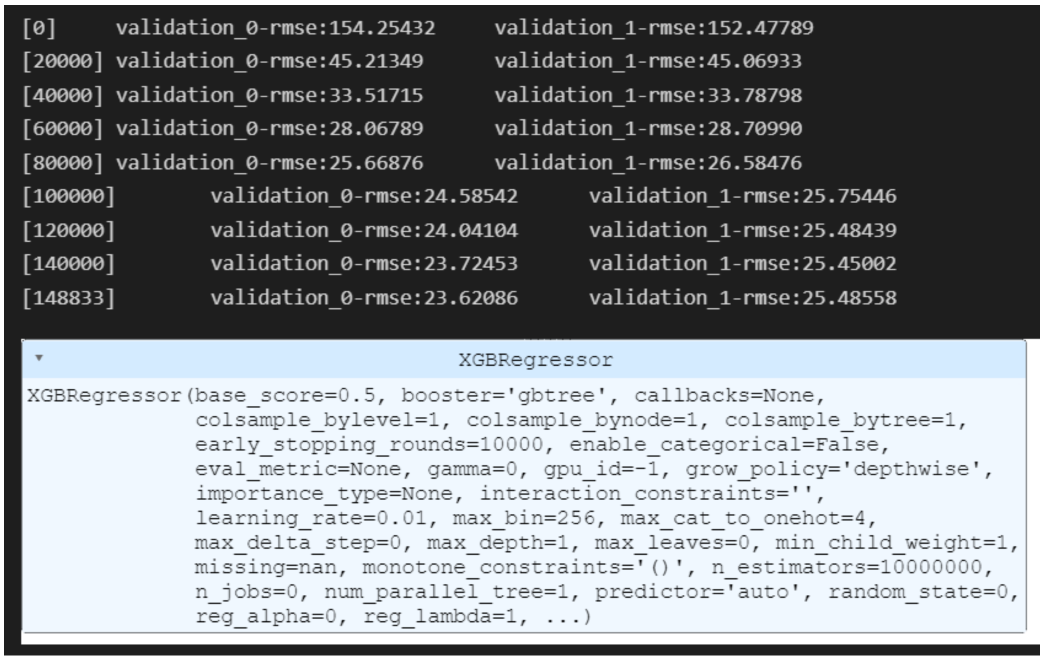
Train-Test split of the dataset



Monthly Electricity Consumption

- We used XGBoost Regressor for the time series analysis, it had an accuracy of 95% and root mean squared error of 25.45. As done before we converted the model to a pickle file and define a function that would return the predicted electricity demand.





```
[0]     validation_0-rmse:154.25432     validation_1-rmse:152.47789
[20000] validation_0-rmse:45.21349      validation_1-rmse:45.06933
[40000] validation_0-rmse:33.51715      validation_1-rmse:33.78798
[60000] validation_0-rmse:28.06789      validation_1-rmse:28.70990
[80000] validation_0-rmse:25.66876      validation_1-rmse:26.58476
[100000]        validation_0-rmse:24.58542      validation_1-rmse:25.75446
[120000]        validation_0-rmse:24.04104      validation_1-rmse:25.48439
[140000]        validation_0-rmse:23.72453      validation_1-rmse:25.45002
[148833]        validation_0-rmse:23.62086      validation_1-rmse:25.48558
```

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=10000, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.01, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=1, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=10000000,
             n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
             reg_alpha=0, reg_lambda=1, ...)
```

Rmse losses and the parameters that the model used

- Further we used Flask to create our api on Heroku which returns the estimated electricity generated and the electricity demands as a json file. It can be used in the form, Solar energy predictions is available as: https://solar-sight.herokuapp.com/api/?place=<location here> . And time series based electricity demands predictions is available as: https://solar-sight.herokuapp.com/api/time-series/?dayofyear=<dayofyear>&dayofweek=<ayofweek>&quarter=<quarter>&month=<month>&year=<year>&state=<state>&region=<region>.

Frontend

For the front end we used Flutter, first prompting the user for the location. Eventually, we used the location to make an api call using flutter's built-in functions and displayed the output.