Predict solar radiation for optimal solar power generation.

Waqar Ahmed

Abstract

Given meteorological data predict future solar radiation to determine viability of solar power generation. Verify that a regression model can be created to accurately predict ‘solar radiation’ based on training and test data.

Motivation

Being able to predict solar radiation allows us to make decisions on solar power viability and how best to configure solar equipment to optimize solar energy generation.

Dataset(s)

The meteorological data set includes data for:

- Solar Radiation (Watts per Meter^2)
- Temperature (Farenheit)
- Humidity (Percent)
- Wind direction (Degrees)
- Wind speed (Miles per hour)
- Pressure (Hg)
- Sunrise/set (Hawaii time)

The data set is for Hawaii over a four month period provided by NASA:

https://www.kaggle.com/datasets/dronio/SolarEnergy?select=SolarPrediction.csv

Data Preparation and Cleaning

Initially we want to clear rows with null data, this can be achieved through **dropna()** function, applying this:

```
In [32]: data = pd.read_csv("../SolarPrediction.csv")

In [34]: data.head()
         print(data.shape)

         (32686, 11)

In [35]: data = data.dropna()
         print(data.shape)

         (32686, 11)
```

The row counts before and after show there are no blank rows

Data Preparation and Cleaning

Further the data doesn’t need scaling as we are performing a regression.

The data set is inclusive of daytime and nighttime hours. Given this; we could trim the data to exclude nighttime hours, as solar radiation is not sufficient to generate electricity during this time. There are two possible ways we can achieve this:

1. Use the **SunSetTime/SunRiseTime** columns to identify if the recording is for daylight hours or night. We can use a binary label for this and then remove all recordings during night time.
2. Identify all ‘Radiation’ below threshold and filter out as nighttime hours

Data Preparation and Cleaning

Looking at the dataset there is no clear threshold below which night hours can be identified cleanly. As exploratory analysis shows there are daytime periods which also have low levels (maybe due to cloud coverage or other reasoning). Hence the most conclusive method of cleaning this raw dataset form night hour recordings would be to use the **SunSetTime/SunRiseTime** columns to filter out all data point during the night.

Data Preparation and Cleaning

Filtering out night time recordings from dataset (note the night duration recordings are not zero for ‘Radiation’. As there is some solar (lunar) power generated during night but this is negligible compared to daylight hours hence the reason to only look at daylight data)

```
day_rows = data[(data['Time'] > data['TimeSunRise']) & (data['Time'] < data['TimeSunSet'])]
```

Data Preparation and Cleaning

Filtering out night recordings reduces the data set by more than half, we now have **15608** row data from the initial raw data of **32686** rows. With min/max solar radiation values at:

Min: 1.19

Max: 1601.26

```
min_radiation = day_rows['Radiation'].min()
max_radiation = day_rows['Radiation'].max()
day_rows['SunRiseHour'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.hour
day_rows['SunRiseMin'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.minute
day_rows['SunSetHour'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.hour
day_rows['SunSetMin'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.minute
print(f'min: {min_radiation} , max: {max_radiation} rows: {day_rows.shape[0]}')
day_rows
```

```
min: 1.19 , max: 1601.26 rows: 15608
```

Data Preparation and Cleaning

The **SunSetTime** and **SunRiseTime** columns are strings in the dataset and as I want to use them as **features**. I will split the hours and minutes into separate columns as integers:

```
day_rows['SunRiseHour'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.hour
day_rows['SunRiseMin'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.minute
day_rows['SunSetHour'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.hour
day_rows['SunSetMin'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.minute
```

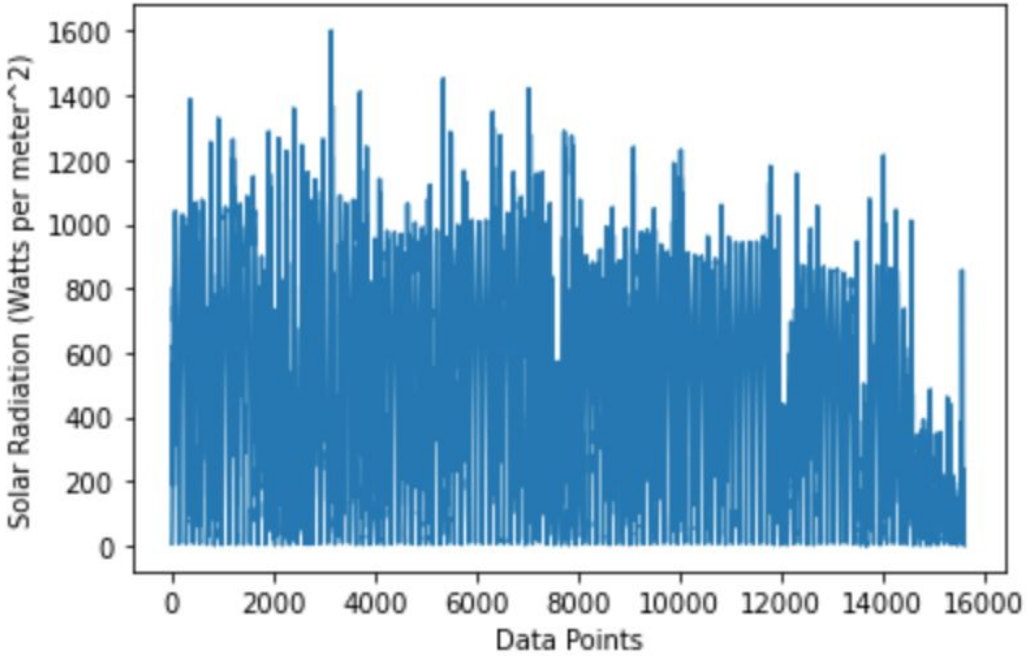
ne	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet	SunRiseHour	SunRiseMin	SunSetHour	SunSetMin
52	6.63	53	30.44	59	118.82	5.62	06:13:00	18:13:00	6	13	18	13
22	10.96	54	30.44	59	154.16	4.50	06:13:00	18:13:00	6	13	18	13
22	19.42	55	30.44	57	58.42	6.75	06:13:00	18:13:00	6	13	18	13

Research Question(s)

What is the predicted solar radiation based on the current data. Create and validate a regression model with training and test data based on ~70/30 % split. Identify accuracy of model and predict future solar radiation based on model.

Data analysis/exploration visualisation

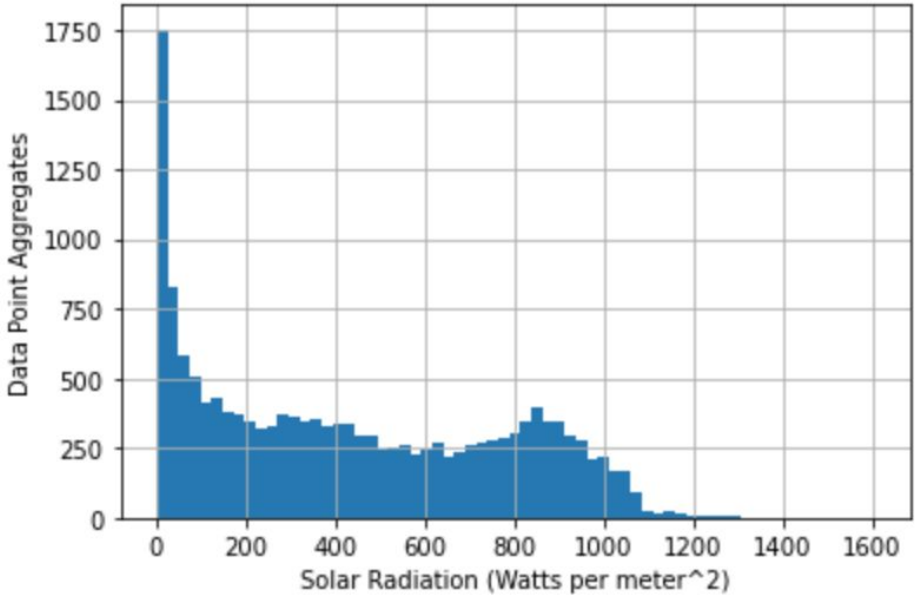
Graphing ‘Solar Radiation’ over time from our filtered data frame we get:



We see varied ‘Solar Radiation’ across time with some intermittent periods of high peaks.

Data analysis/exploration visualisation

Histogram of solar radiation:



Right skewed dataset contains more lower values from solar radiation than higher values. Not a uniform distribution.

Methods

The method of data analysis will compare three regression algorithms:

- ***Linear***, sklearn.linear_model **LinearRegression()**
- ***Decision Tree***, sklearn.tree **DecisionTreeRegressor()**
- ***Random Forest***, sklearn.ensemble **RandomForestRegressor()**

All methods will use the below **features** and **target**:

Features

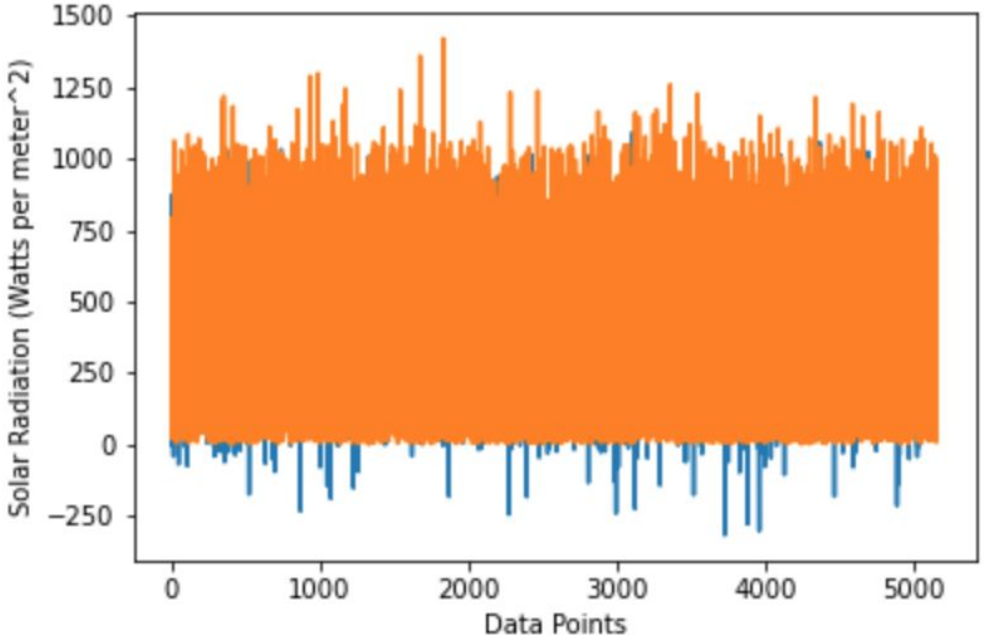
```
features = ['UNIXTime','Temperature', 'Pressure', 'Humidity','WindDirection(Degrees)',
            'Speed','SunRiseHour','SunRiseMin','SunSetHour','SunSetMin']
```

Target:

```
target = ['Radiation']
```

Findings (Linear regression)

Linear regression



There looks to be lots of overlap however the linear regression predictions (blue) has introduced negative values (which are not present in the actual target test data (orange))

Findings (Linear regression)

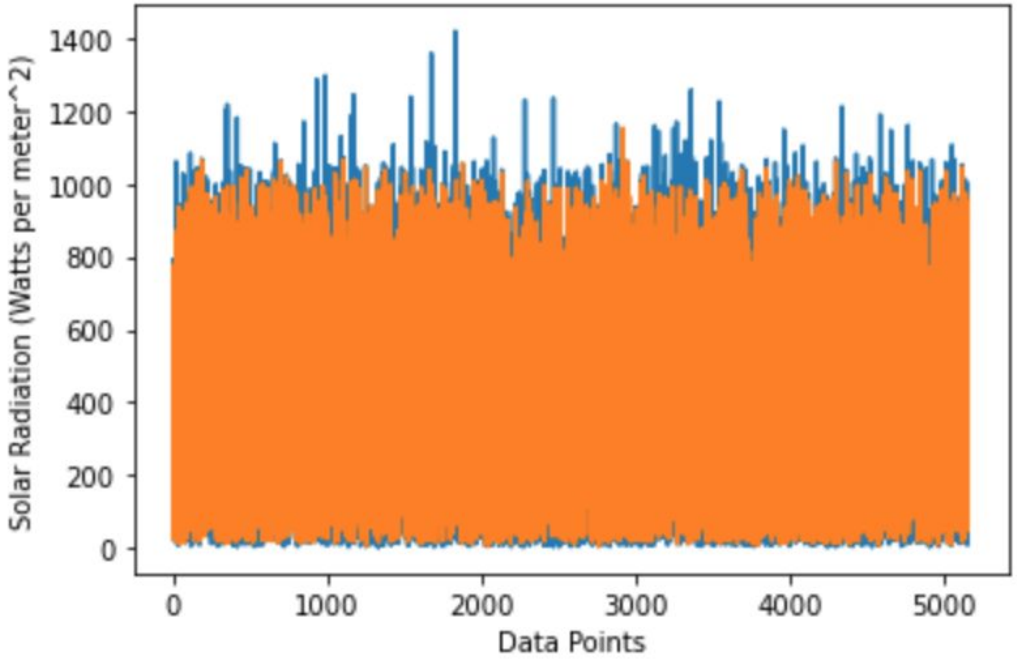
Calculating the RMSE we get a ridiculously high value of ~244 which indicates that our model is not very good.

```
# Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_prediction)))
```

```
Mean Absolute Error: 195.40384330203003
Mean Squared Error: 59704.3084841211
Root Mean Squared Error: 244.34465102416524
```

Findings (Random Forest Regression)

Random Forest Regression



This shows better visual match than linear regression and there are no negative values generated here.

Findings (Random Forest Regression)

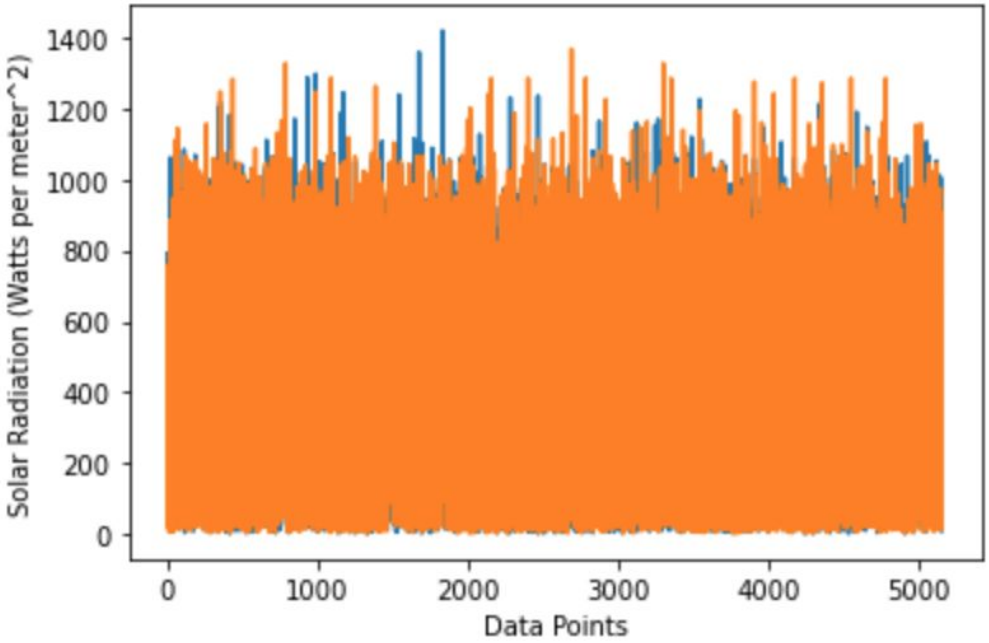
Calculating the RMSE we get a high value of ~126 which is substantially better than linear regression but still a high value indicating it's not a great model for predicting !

```
# Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, forest_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, forest_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, forest_prediction)))
```

```
Mean Absolute Error: 80.30258250663293
Mean Squared Error: 16118.780000598572
Root Mean Squared Error: 126.9597574060323
```

Findings (Decision Tree Regression)

Decision Tree Regression



This shows better visual match overall and there are no negative values generated here either.

Findings (Decision Tree Regression)

Calculating the RMSE we get a high value of ~173 which is substantially better than linear but still a high value indicating it's not a great model for predicting !

```
: # Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_tree_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_tree_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_tree_prediction)))
```

```
Mean Absolute Error: 96.19657736361873
Mean Squared Error: 30056.70319598137
Root Mean Squared Error: 173.36869151026482
```

Limitations

Unfortunately the dataset was only for a period of four months. Which doesn’t take into account seasonal changes e.t.c Realistically would look to use a dataset with far more data; stretching back at least 5-10 years to be able to make more conclusive extrapolations.

Conclusions

The data required substantial cleaning. Linear regression introduced anomalous negative values and gave the worst RMSE value (~242) which indicates that the model was not good for predicting. Best results where shown using a random forest regression giving an RMSE (~162) however this is still high (Given our data range is 1.19 - 1601.26) this indicates we have lots of outliers largely affecting RMSE. We would need to identify these and either further clean data or refine/tune the model.

That said the graphical visualisation show that the predictions generally follow the test target pattern and are not too wildly off.

Conclusions

Summary:

Regression Type	Mean Abs Err	Mean Sqrd Err	RMSE	Regressor score
Linear	195	59704	244	0.46
Random Forest	80	16119	126	0.98
Decision Tree	96	30057	173	1

Acknowledgements

I would like to acknowledge the source of my dataset:

(Thanks NASA for the dataset, Part of NASA hackathon)

https://www.kaggle.com/datasets/dronio/SolarEnergy?select=SolarPrediction.csv

Everything else I did myself.

References

I did all the work myself.

Jupyter notebook screenshots

In [1]:
```
import pandas as pd
import datetime
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from math import sqrt
import numpy as np
from sklearn.model_selection import train_test_split
```

In [2]:
```
data = pd.read_csv("../SolarPrediction.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet
0	1475229326	9/29/2016 12:00:00 AM	23:55:26	1.21	48	30.46	59	177.39	5.62	06:13:00	18:13:00
1	1475229023	9/29/2016 12:00:00 AM	23:50:23	1.21	48	30.46	58	176.78	3.37	06:13:00	18:13:00
2	1475228726	9/29/2016 12:00:00 AM	23:45:26	1.23	48	30.46	57	158.75	3.37	06:13:00	18:13:00
3	1475228421	9/29/2016 12:00:00 AM	23:40:21	1.21	48	30.46	60	137.71	3.37	06:13:00	18:13:00
4	1475228124	9/29/2016 12:00:00 AM	23:35:24	1.17	48	30.46	62	104.95	5.62	06:13:00	18:13:00

In [4]:
```
print(data.shape)
data = data.dropna()
print(data.shape)
data.dtypes
```

```
(32686, 11)
```

In [6]:
```
day_rows = data[(data['Time'] > data['TimeSunRise']) & (data['Time'] < data['TimeSunSet'])]
day_rows = day_rows.reset_index(drop=True)
min_radiation = day_rows['Radiation'].min()
max_radiation = day_rows['Radiation'].max()
day_rows['SunRiseHour'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.hour
day_rows['SunRiseMin'] = pd.to_datetime(day_rows['TimeSunRise'], format='%H:%M:%S').dt.minute
day_rows['SunSetHour'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.hour
day_rows['SunSetMin'] = pd.to_datetime(day_rows['TimeSunSet'], format='%H:%M:%S').dt.minute
print(f'min: {min_radiation} , max: {max_radiation} rows: {day_rows.shape[0]}')
day_rows
```

```
min: 1.19 , max: 1601.26 rows: 15608
```

Out[6]:

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet	SunRiseHour
0	1475208652	9/29/2016 12:00:00 AM	18:10:52	6.63	53	30.44	59	118.82	5.62	06:13:00	18:13:00	6
1	1475208322	9/29/2016 12:00:00 AM	18:05:22	10.96	54	30.44	59	154.16	4.50	06:13:00	18:13:00	6
2	1475208022	9/29/2016 12:00:00 AM	18:00:22	19.42	55	30.44	57	58.42	6.75	06:13:00	18:13:00	6
3	1475207722	9/29/2016 12:00:00 AM	17:55:22	27.14	55	30.44	53	47.86	4.50	06:13:00	18:13:00	6

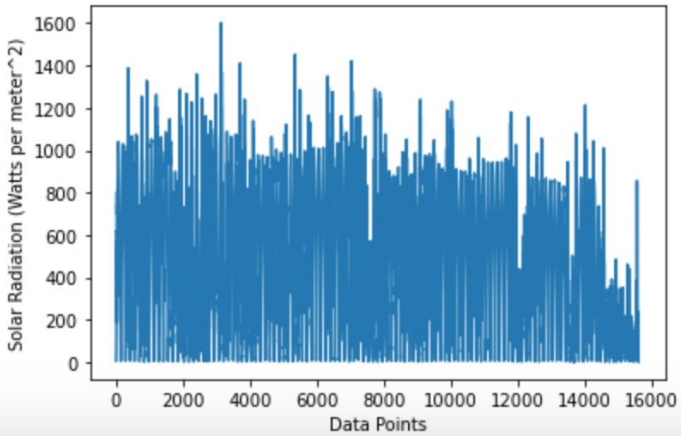
In [7]:
```
features = ['UNIXTime','Temperature', 'Pressure', 'Humidity','WindDirection(Degrees)',
            'Speed','SunRiseHour','SunRiseMin','SunSetHour','SunSetMin']
target = ['Radiation']

X = day_rows[features]
y = day_rows[target]
y.shape
```

Out[7]:
```
(15608, 1)
```

In [8]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

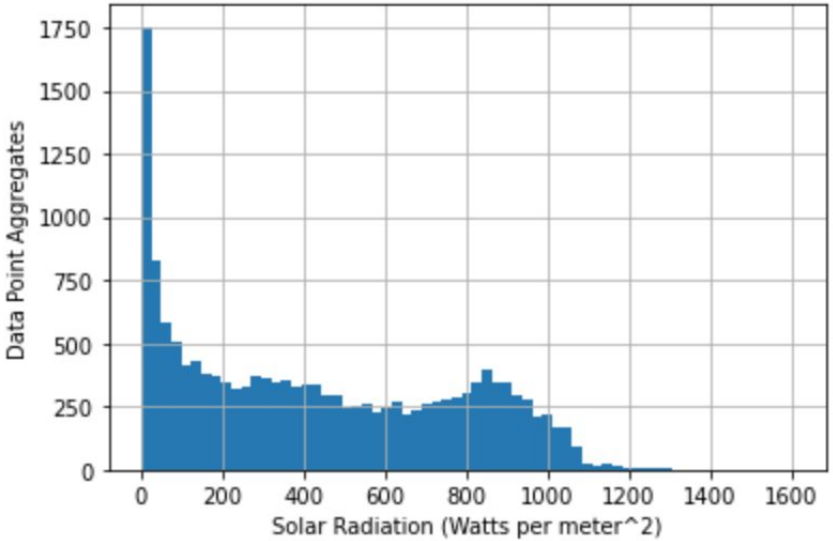
In [9]:
```
plt.plot(y)
plt.xlabel('Data Points')
plt.ylabel('Solar Radiation (Watts per meter^2)')
```

Out[9]:
```
Text(0, 0.5, 'Solar Radiation (Watts per meter^2)')
```



```
plt.hist(y,density=False, bins=65)
plt.grid()
plt.ylabel('Data Point Aggregates')
plt.xlabel('Solar Radiation (Watts per meter^2)')
```

```
Text(0.5, 0, 'Solar Radiation (Watts per meter^2)')
```




```
# Train/test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33,random_state=1)
print(y_train)
```

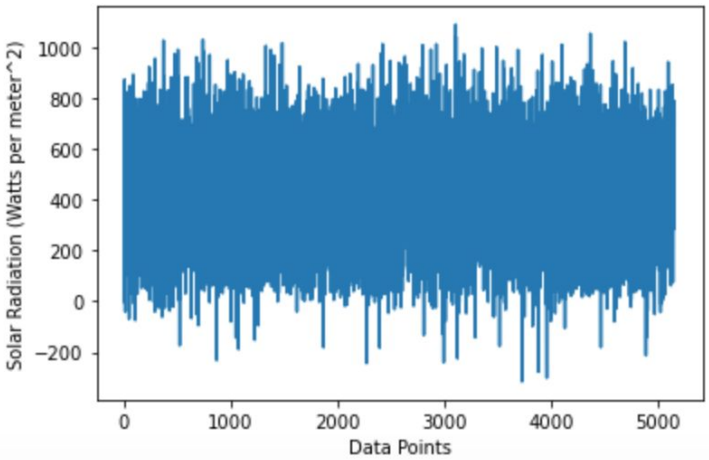
```
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```

```
LinearRegression()
```

```
y_prediction = regressor.predict(X_test)
```

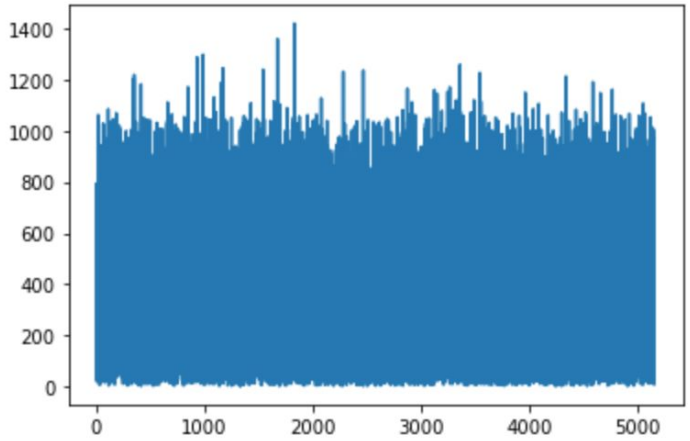
```
y_prediction
y_test =y_test.reset_index(drop=True)
plt.xlabel('Data Points')
plt.ylabel('Solar Radiation (Watts per meter^2)')
plt.plot(y_prediction)
#plt.plot(y_test)
```

```
[<matplotlib.lines.Line2D at 0x122dce100>]
```



```
plt.plot(y_test)
```

```
[<matplotlib.lines.Line2D at 0x123032490>]
```




```
# Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_prediction)))
```

```
Mean Absolute Error: 195.40384330203003
Mean Squared Error: 59704.3084841211
Root Mean Squared Error: 244.34465102416524
```

```
print(regressor.score(X_train,y_train))
```

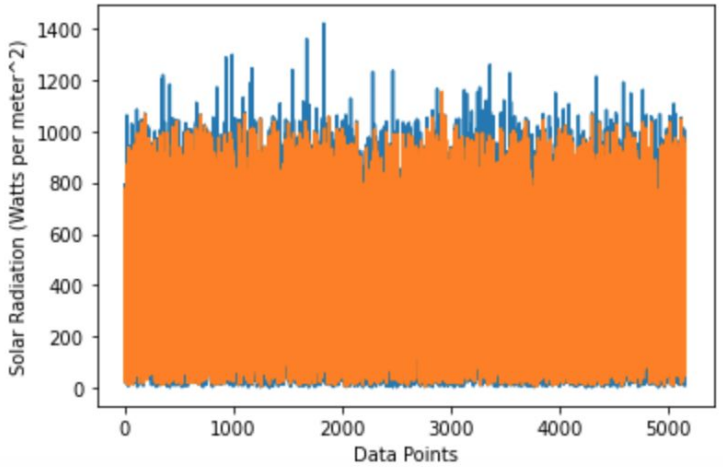
```
0.45774032803952736
```

```
foret_regressor = RandomForestRegressor(n_estimators = 1200 , random_state = 9 , min_weight_fraction_leaf = 0.00001)
foret_regressor.fit(X_train,y_train.values.ravel())
```

```
RandomForestRegressor(min_weight_fraction_leaf=1e-05, n_estimators=1200,
                      random_state=9)
```

```
plt.plot(y_test)
forest_prediction = foret_regressor.predict(X_test)
plt.plot(forest_prediction)
plt.xlabel('Data Points')
plt.ylabel('Solar Radiation (Watts per meter^2)')
#y_test.shape
forest_prediction.shape
```

```
(5151,)
```



```
# Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, forest_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, forest_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, forest_prediction)))
```

```
Mean Absolute Error: 80.30258250663293
Mean Squared Error: 16118.780000598572
Root Mean Squared Error: 126.9597574060323
```

```
print(foret_regressor.score(X_train,y_train))
```

```
0.9795346294470793
```

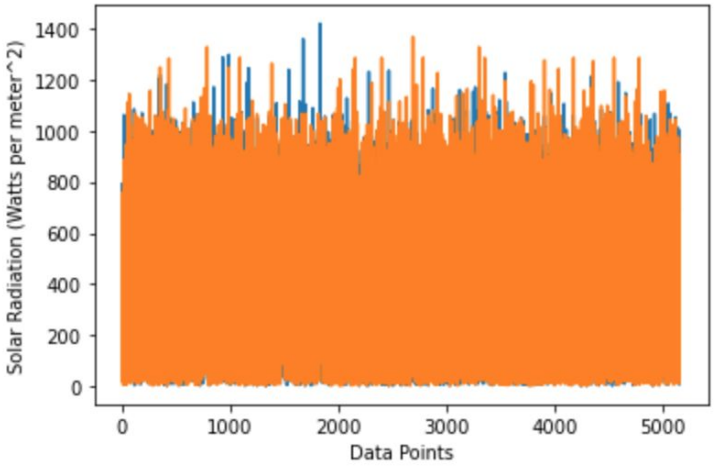
```
tree_regressor = DecisionTreeRegressor()
tree_regressor.fit(X_train,y_train)
```

```
DecisionTreeRegressor()
```

```
y_tree_prediction = tree_regressor.predict(X_test)
```

```
plt.xlabel('Data Points')
plt.ylabel('Solar Radiation (Watts per meter^2)')
plt.plot(y_test)
plt.plot(y_tree_prediction)
```

```
[<matplotlib.lines.Line2D at 0x12420a8b0>]
```




```
# Calculate accuracy
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_tree_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_tree_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_tree_prediction)))
```

```
Mean Absolute Error: 96.19657736361873
Mean Squared Error: 30056.70319598137
Root Mean Squared Error: 173.36869151026482
```

In [25]:
```
print(tree_regressor.score(X_train,y_train))
```

```
1.0
```

End