Detection of Analyte medium Refractive Index Using Machine Learning Techniques on Simulated Grating Structures

Tejasva Soni

Integrated MSc Physics, IIT Roorkee, Roorkee-247667

Guide:

Dr Rajib Biswas, Ph.D.

Professor(Assistant)

Department of physics, Tezpur University, Tezpur-784028

Abstract:

Numerous applications exist for material detection as transparent and light colored liquids, ranging from industrial production to pharmaceutical, environmental monitoring, and hazardous risk control. For many applications, infrared absorption spectroscopy is a favored technique, due to attributes like short response time, high specificity, minimal drift, negligible sample disruption, and reliability.
Here we perform an implementation study on a Grating structure, whose simulated transmission and reflection spectra will be analysed and in turn will be used to make a Machine Learning Regression Model that can predict the analyte material refractive index.

Keywords: Random Forest Regression, SciKit Learn, Decision Trees, Analyte Material

Abbreviations:

n0	Refractive Index of Analyte Material
RFR	Random Forest Regression
MSE	Mean Square Error
RMSE	Root Mean Square Error
ML	Machine Learning

1. INTRODUCTION

1.1Approach

The workflow of this study can be divided into the following structure to which each step will be discussed in detail.

1. Grating Simulation and software interfacing
2. Data generation and Cleaning
3. Literature review
4. Model implementation and optimization
5. Result analysis

2. Walkthrough

2.1Grating Simulation and Software interfacing

In this Project MATLAB based Grator 2003 was used to simulate Grating structure and properties. Its data flow can be understood by analysing the input and output parameters.

2.1.1 INPUTS

The input section is divided into three main sections:

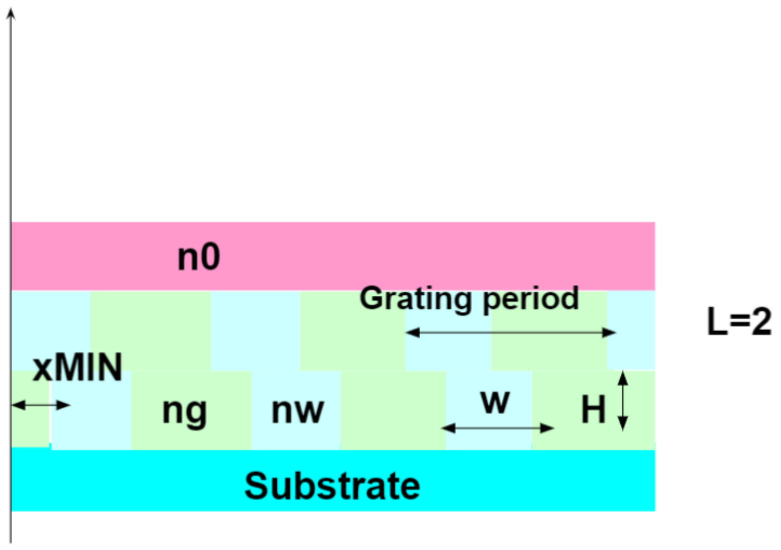
Independent parameters – :

- L - The number of layers in the grating
- M - Number of expanding modes
- Polarization
- Wavelength
- Grating period
- Incidence angle
- n0 - Entrance dielectric coefficient

The substrate parameter – this parameter might be dependent on the wavelength parameter.

The Layers parameters – these parameters are the structure and material types of the different layers in the grating. They are:

- ng - Dielectric coefficient g(Real)
- ng - Dielectric coefficient g(Imaginary)
- nw - Dielectric coefficient w(Real)
- nw - Dielectric coefficient w(Imaginary)
- xMIN - Shift from symmetry axis of grating
- H - Height of layer
- W - The breadth of w



A sample Grating Structure

A sample structure parameter is as follow:

1. **TOP** - ANALYTE MATERIAL (n0 ref index)
2. **LAYER 1** - cover layer of 22 nm of SiO2
3. **LAYER 2-**
   The gold (Au) nanoslit array of 25-nm height

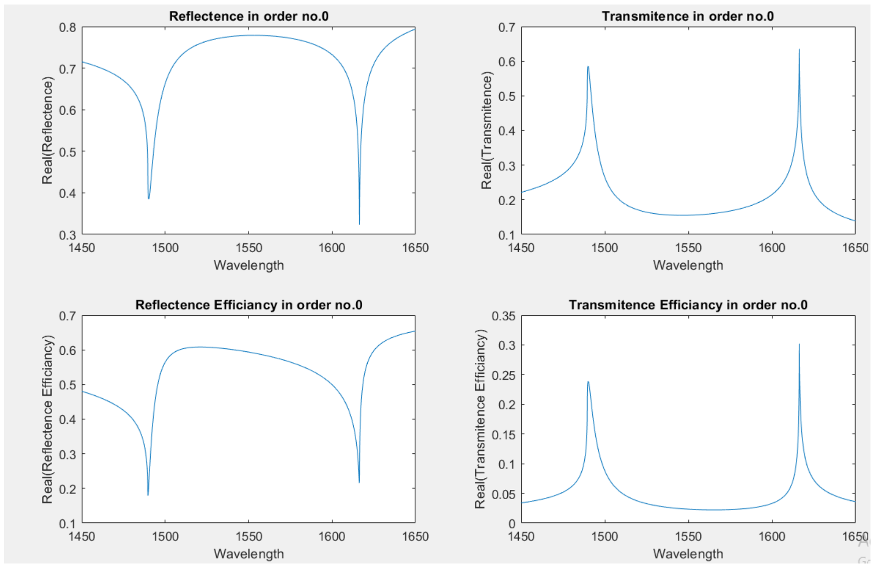
   72-nm groove width and period of 1120 nm
4. **SiO2 SUBSTRATE LAYER**

We use the 2-dimensional variation, that is, it allows us to choose two of the listed parameters and execute a simulation on a range of values of those certain parameters. This option will allow us to receive a three dimensional plot of the distribution of results according to the different values of the chosen parameters. In our case we choose these two parameters as No and wavelength.

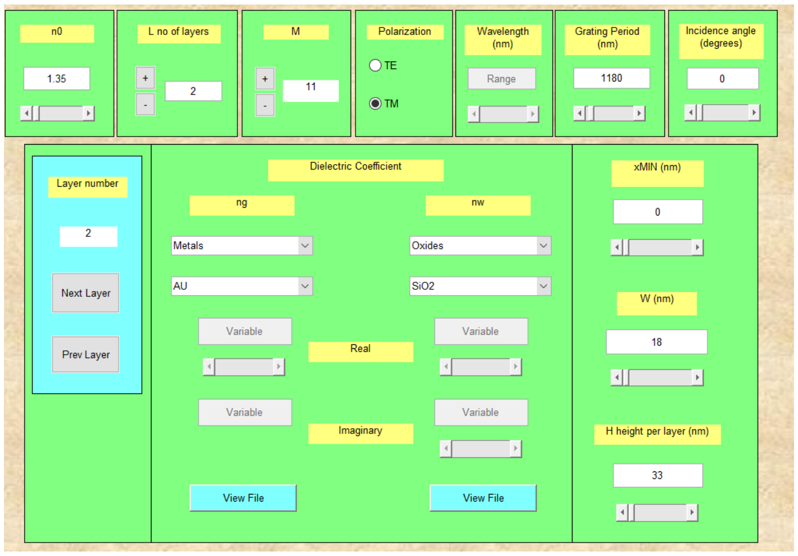
The program outputs four main numeric values:

- Transmittance (complex number)
- Reflectance (complex number)
- Transmittance efficiency (real number)
- Reflectance efficiency (real number)

We choose Transmission coefficient/Transmittance vs Wavelength in the range that we provided in input and then use this data as an input to ML models and try to predict The refractive Index of analyte material.



A sample output of the simulation that is output parameters Vs Wavelength


n0: 1.35
L no of layers: 2
M: 11
Polarization: TE [✓ TM]
Wavelength (nm): Range
Grating Period (nm): 1180
Incidence angle (degrees): 0
Layer number: 2
Dielectric Coefficient
ng: Metals, AU
nw: Oxides, SiO2
xMIN (nm): 0
W (nm): 18
H height per layer (nm): 33

Grator interface

2.2 Data generation and Cleaning

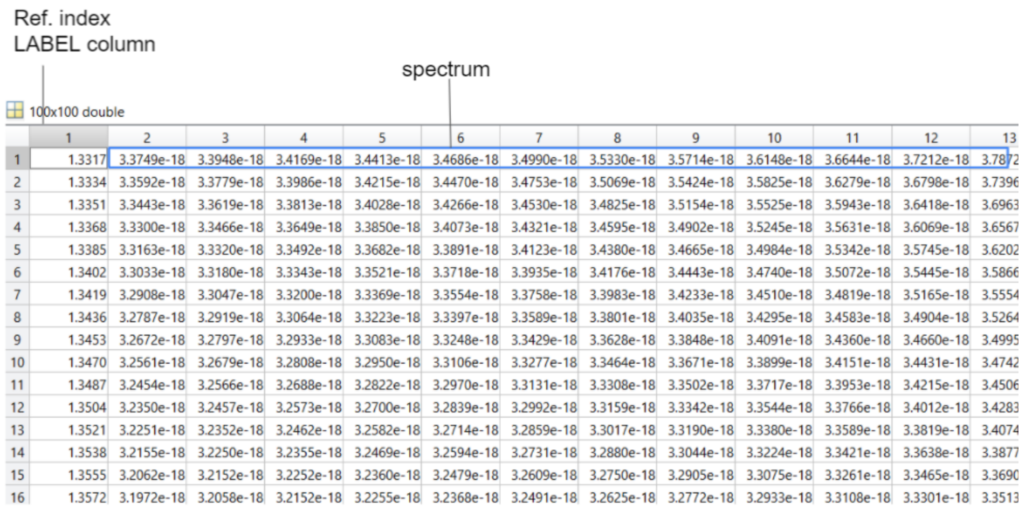
At this stage we build up a well structured and clean dataset to be further applied machine learning models upon efficiently.
Performing a construction of a small sample dataset, First we make a two dimensional varied output from the simulator and store it in a 2-D matrix tabular form.

Wavelength - 1500-1800 100 measurements
transparent and light colored liquids having a refractive index in the range from 1.33 to 1.50. Hence,
Analyte material ref. Index No 1.33 to 1.5 100 measurements
Hence total size = 100 x 100 = 10000 cells
All other parameters are kept constant.

Some points about construction:
From the generated data, out of all we picked out the 0th order transmission spectrum for further processing and analysis. Corresponding to each value of the refractive index we have a spectrum(a series). Hence making a 2-D matrix for the range of indexes.

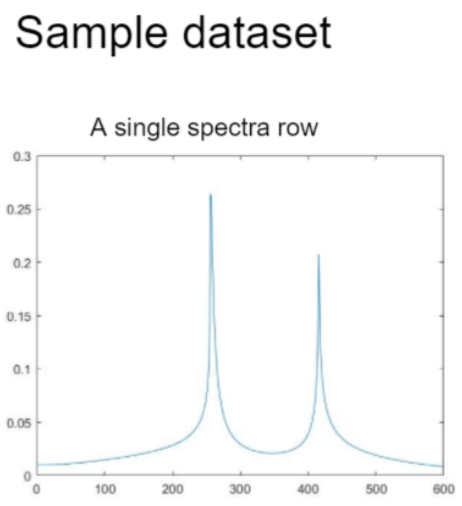
We note, 1st column as LABEL column corresponding to refractive index No and the rest of the row except the first cell is the Spectrum(transmission).
Similarly TEST dataset will be picked out as a sliced part of the whole dataset or further whole data will be splitted into TRAIN-TEST for validation.

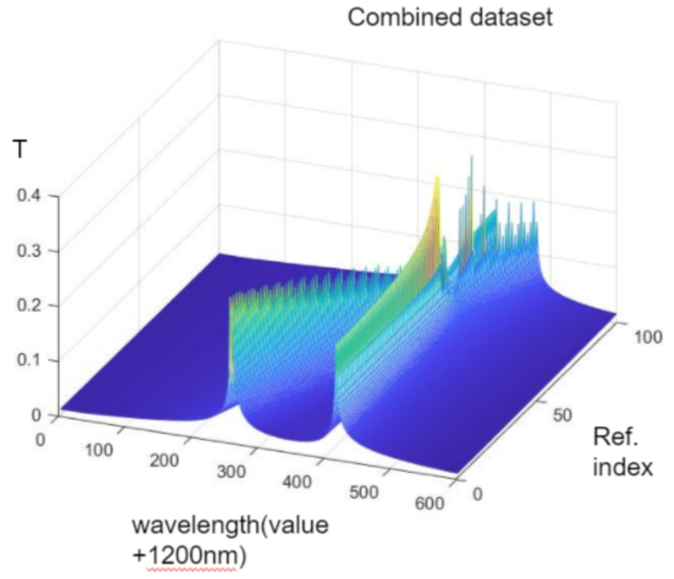
Sample dataset:


Ref. index
LABEL column
spectrum
100x100 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.3317	3.3749e-18	3.3948e-18	3.4169e-18	3.4413e-18	3.4686e-18	3.4990e-18	3.5330e-18	3.5714e-18	3.6148e-18	3.6644e-18	3.7212e-18	3.7872
2	1.3334	3.3592e-18	3.3779e-18	3.3986e-18	3.4215e-18	3.4470e-18	3.4753e-18	3.5069e-18	3.5424e-18	3.5825e-18	3.6279e-18	3.6798e-18	3.7396
3	1.3351	3.3443e-18	3.3619e-18	3.3813e-18	3.4028e-18	3.4266e-18	3.4530e-18	3.4825e-18	3.5154e-18	3.5525e-18	3.5943e-18	3.6418e-18	3.6963
4	1.3368	3.3300e-18	3.3466e-18	3.3649e-18	3.3850e-18	3.4073e-18	3.4321e-18	3.4595e-18	3.4902e-18	3.5245e-18	3.5631e-18	3.6069e-18	3.6567
5	1.3385	3.3163e-18	3.3320e-18	3.3492e-18	3.3682e-18	3.3891e-18	3.4123e-18	3.4380e-18	3.4665e-18	3.4984e-18	3.5342e-18	3.5745e-18	3.6202
6	1.3402	3.3033e-18	3.3180e-18	3.3343e-18	3.3521e-18	3.3718e-18	3.3935e-18	3.4176e-18	3.4443e-18	3.4740e-18	3.5072e-18	3.5445e-18	3.5866
7	1.3419	3.2908e-18	3.3047e-18	3.3200e-18	3.3369e-18	3.3554e-18	3.3758e-18	3.3983e-18	3.4233e-18	3.4510e-18	3.4819e-18	3.5165e-18	3.5554
8	1.3436	3.2787e-18	3.2919e-18	3.3064e-18	3.3223e-18	3.3397e-18	3.3589e-18	3.3801e-18	3.4035e-18	3.4295e-18	3.4583e-18	3.4904e-18	3.5264
9	1.3453	3.2672e-18	3.2797e-18	3.2933e-18	3.3083e-18	3.3248e-18	3.3429e-18	3.3628e-18	3.3848e-18	3.4091e-18	3.4360e-18	3.4660e-18	3.4995
10	1.3470	3.2561e-18	3.2679e-18	3.2808e-18	3.2950e-18	3.3106e-18	3.3277e-18	3.3464e-18	3.3671e-18	3.3899e-18	3.4151e-18	3.4431e-18	3.4742
11	1.3487	3.2454e-18	3.2566e-18	3.2688e-18	3.2822e-18	3.2970e-18	3.3131e-18	3.3308e-18	3.3502e-18	3.3717e-18	3.3953e-18	3.4215e-18	3.4506
12	1.3504	3.2350e-18	3.2457e-18	3.2573e-18	3.2700e-18	3.2839e-18	3.2992e-18	3.3159e-18	3.3342e-18	3.3544e-18	3.3766e-18	3.4012e-18	3.4283
13	1.3521	3.2251e-18	3.2352e-18	3.2462e-18	3.2582e-18	3.2714e-18	3.2859e-18	3.3017e-18	3.3190e-18	3.3380e-18	3.3589e-18	3.3819e-18	3.4074
14	1.3538	3.2155e-18	3.2250e-18	3.2355e-18	3.2469e-18	3.2594e-18	3.2731e-18	3.2880e-18	3.3044e-18	3.3224e-18	3.3421e-18	3.3638e-18	3.3877
15	1.3555	3.2062e-18	3.2152e-18	3.2252e-18	3.2360e-18	3.2479e-18	3.2609e-18	3.2750e-18	3.2905e-18	3.3075e-18	3.3261e-18	3.3465e-18	3.3690
16	1.3572	3.1972e-18	3.2058e-18	3.2152e-18	3.2255e-18	3.2368e-18	3.2491e-18	3.2625e-18	3.2772e-18	3.2933e-18	3.3108e-18	3.3301e-18	3.3513

Sample dataset Visualization in 3D:





MATLAB CODE:

```
avec=[];                    %dataset to be stored in avec
no = 100                    %range of analyte material
for i=1:no
b = [result(:,i).Teff];     %result is output file from simulation
d = b(8,:);                 %8 is to pick out the 0th order
avec = [avec;d];            %bd dummy variable
end
for i=1:no
avec(i,1) = 1.33+ (i/no)*0.17; %creating label column for analyte material
end
writematrix(avec,'final.csv');
```

2.3 Model implementation and optimization

Choosing a ML model is quite an iterative process which can be improved by having an intuition about the type of data and what we want to get out of it.

Our INPUT data is in the form of a spectrum(time series) plot. While OUTPUT is a single value ie Refractive index of analyte material.

ML algorithms can be classified into two groups based on their working about data to make predictions, that is, supervised and unsupervised learning.

Supervised Machine Learning: Supervised learning is where we have input variables (x) and output variables (Y) and we use ML algorithm to figure out the mapping function from the input to the output i.e. Y = f(X)
Supervised learning can further be classified into Regression and Classification problems.

A **regression** problem is when the output variable is real or a continuous value. Examples are market price prediction, weather temperature prediction.
Whereas a **classification** problem is where the output variable is categorized, such as “white” or “black” or “cancer” and “no cancer”, examples are differentiating subgroups in a larger population.

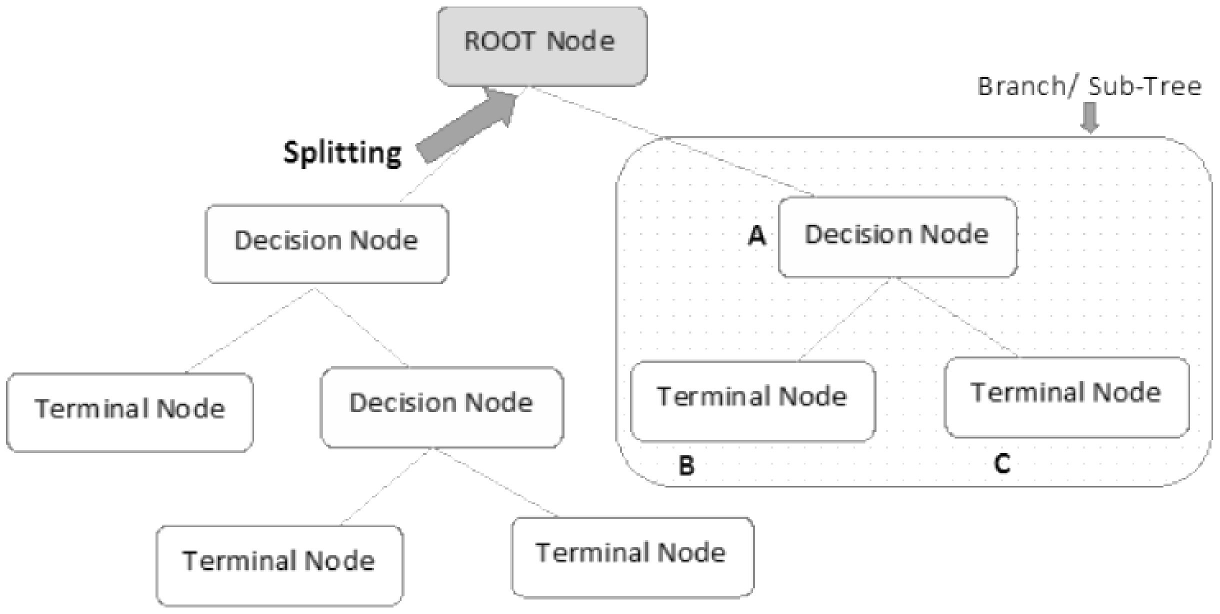
From the above discussion we understand that our problem statement of predicting the No from the input spectrum is a supervised learning problem of type regression.

2.3.1 Decision Trees and Random forest Regression

We choose Random Forest Regression model as it is a popular and robust regression model for the first iteration.

DECISION TREES:

A Decision tree observes features and trains the model in the structure of a tree to predict output in the future and produce meaningful continuous output. A decision tree arrives at an estimated prediction by asking a series of questions on the data, each question narrows the possible values until the model gets confident enough to make the final prediction.

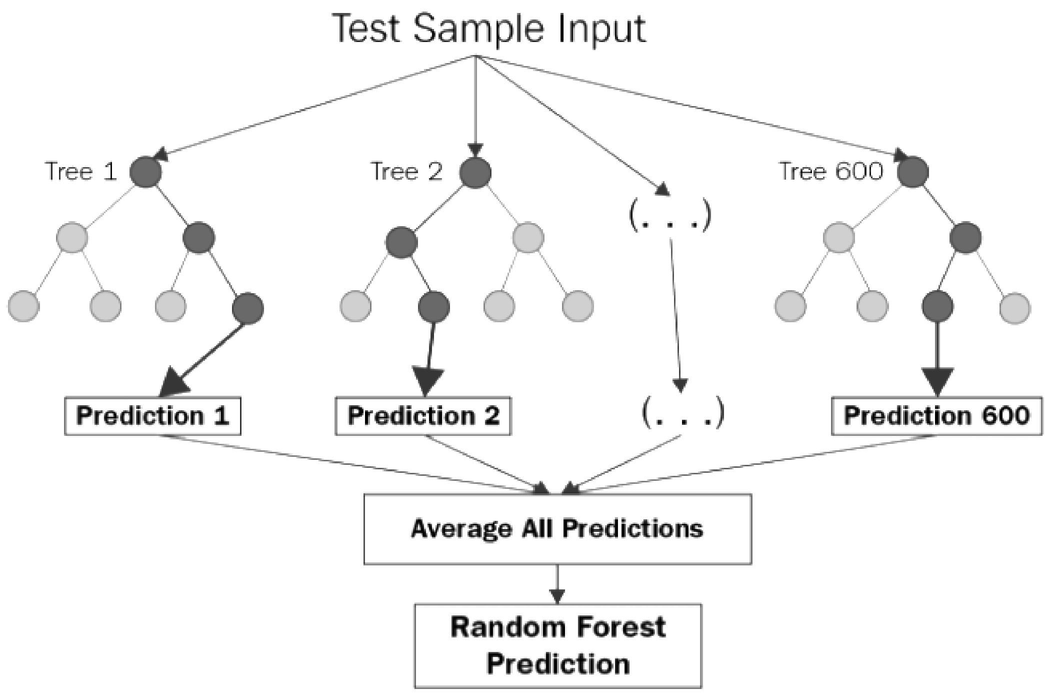


Source: 1.

The order of the question and their content are determined by training the model on previous data.
Decision tree regression normally uses (MSE) mean squared error for deciding to or when split a node in two or more sub-nodes.
When we want to make a prediction the same data format is provided to the model in order to make a prediction. The prediction will be an estimate or regression based on the train data that it has been trained on.

RANDOM FOREST REGRESSOR

Random Forest Regression is a supervised learning algorithm that uses ensemble learning with decision trees for regression.
Ensemble learning with decision trees method is a technique that combines several decision trees and outputs the mean of the prediction classes as the prediction of all the trees.



Source: 2.

Walkthrough steps of its working:

1. Picks random n data points from the training dataset.
2. Builds a decision tree associated with these n data points.
3. Choose the number K of trees we want to use and go back to step 1 and 2.
4. For a new data point, make each one of the K-tree predict the output value for the data point in the problem and assign the new data point to the mean across all of the predicted output values.

Some advantages of using RFR are:

- It reduces overfitting of decision trees and hence improves the accuracy.
- It is used for both classification and regression based problems..
- Normalization of data is not required.

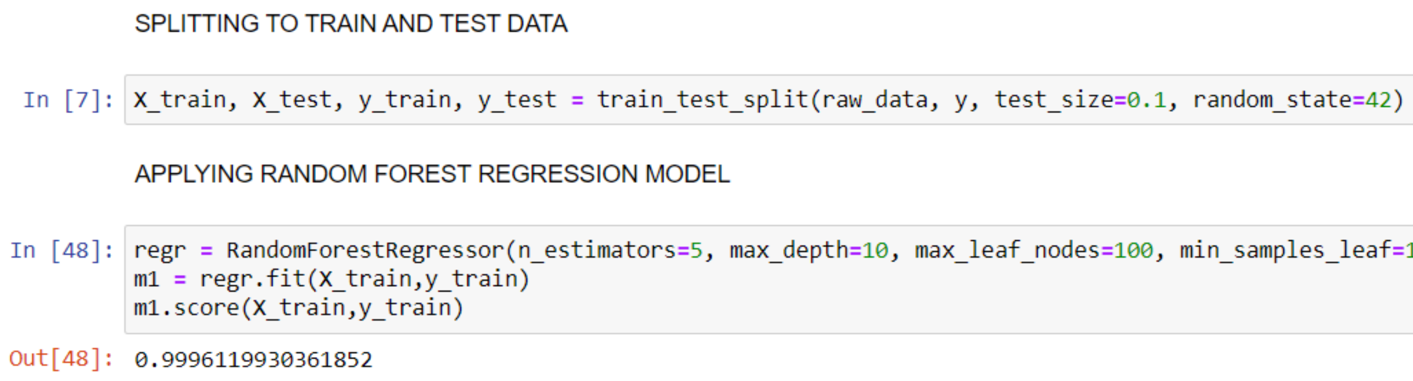
2.3.2 MODEL IMPLEMENTATION

After importing the data from simulator in .csv format, The implementation was done on jupyter notebook using Python language and following packages were taken use of:

Pandas - for data handling
Numpy- for linear Algebra
Matplotlib.pyplot- for data visualization
Sklearn- sci-kit for Machine Learning model implementation

After splitting our whole dataset into Train and test data , The test set then checks the model's predictions based on what it learned from the training set.

Jupyter code cell:



SPLITTING TO TRAIN AND TEST DATA

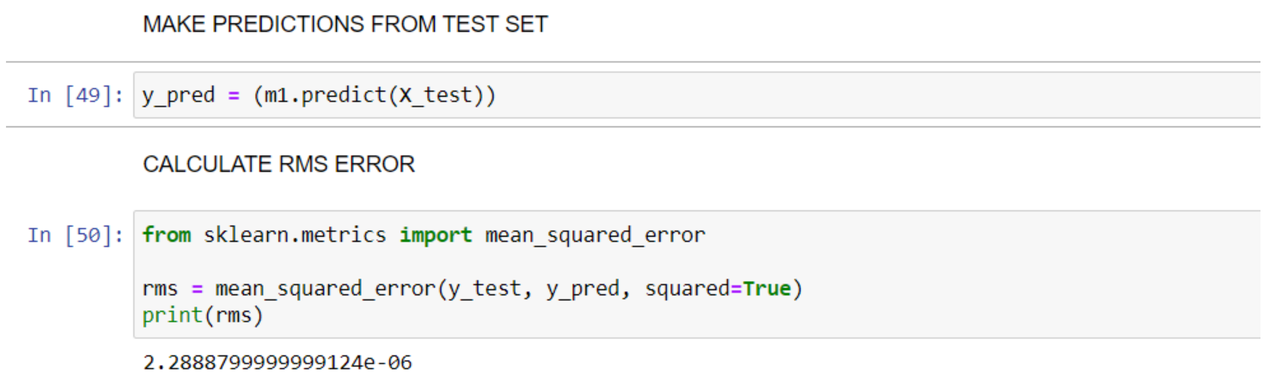
```
In [7]: X_train, X_test, y_train, y_test = train_test_split(raw_data, y, test_size=0.1, random_state=42)
```

APPLYING RANDOM FOREST REGRESSION MODEL

```
In [48]: regr = RandomForestRegressor(n_estimators=5, max_depth=10, max_leaf_nodes=100, min_samples_leaf=1
         m1 = regr.fit(X_train,y_train)
         m1.score(X_train,y_train)

Out[48]: 0.9996119930361852
```

2.3.3 RESULT ANALYSIS

R² score shows us how well the model is fitted to the test data by comparing it to the average of the dependent variable. Closer is the score to 1, then it indicates that the model performs well while further is the score from 1, indicates that model does not perform well.
Prediction Score for small dataset - 0.99



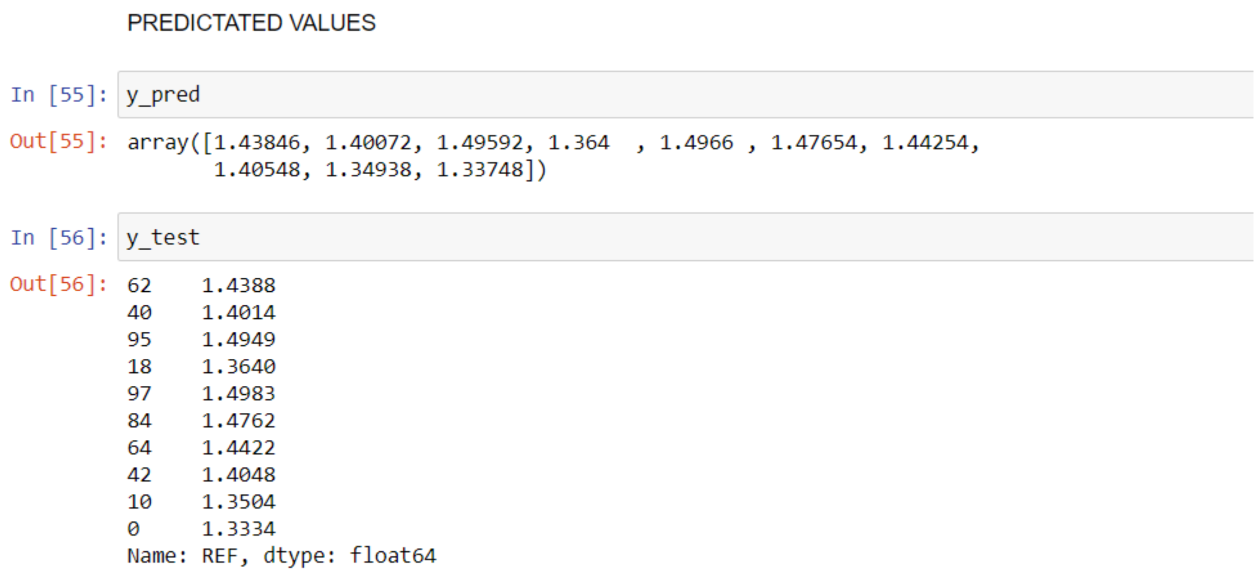
MAKE PREDICTIONS FROM TEST SET

```
In [49]: y_pred = (m1.predict(X_test))
```

CALCULATE RMS ERROR

```
In [50]: from sklearn.metrics import mean_squared_error

         rms = mean_squared_error(y_test, y_pred, squared=True)
         print(rms)

         2.2888799999999124e-06
```

-MSE(mean square error) is very low hence model is performing very well on small dataset

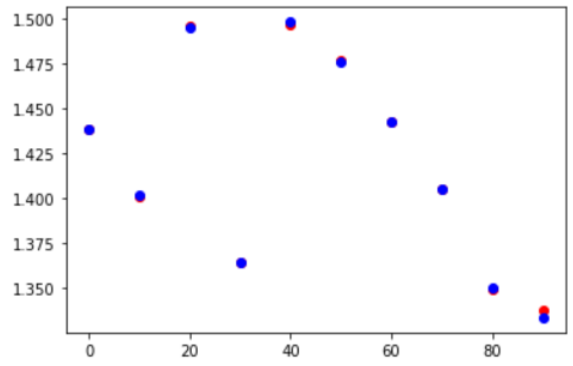


PREDICTATED VALUES

```
In [55]: y_pred

Out[55]: array([1.43846, 1.40072, 1.49592, 1.364  , 1.4966 , 1.47654, 1.44254,
                1.40548, 1.34938, 1.33748])

In [56]: y_test

Out[56]: 62    1.4388
         40    1.4014
         95    1.4949
         18    1.3640
         97    1.4983
         84    1.4762
         64    1.4422
         42    1.4048
         10    1.3504
         0     1.3334
         Name: REF, dtype: float64
```

-Predicted values(ypred) vs actual values(ytest)

```
In [44]: x = np.linspace(0,90,y_test.shape[0])
         plt.scatter(x, y_pred, color ='red') # red shows predicted
         plt.scatter(x, y_test, color ='blue')  #blue is actual
         plt.show()
         # hence perfect overlap shows good prediction
```



-Scatter Plot of prediction

Inference:

The Random Forest Regressor performed really well for this small scale dataset, while the approaches can be using 1-D Convolutional Neural Networks which are efficient at local pattern detection needed for spectral type time series data.

3 Extending Data

In an attempt to increase accuracy and aid the density of data, Keeping the range of variation of wavelength and No. same as before but taking more samples this time to improve the accuracy.

WAVE-LENGTH	1500 1800	*MEASUREMENTS* - 100
N0(ref. Index)	1.3 - 1.5	*MEASUREMENTS* - 500

4 Hyperparameter Tuning

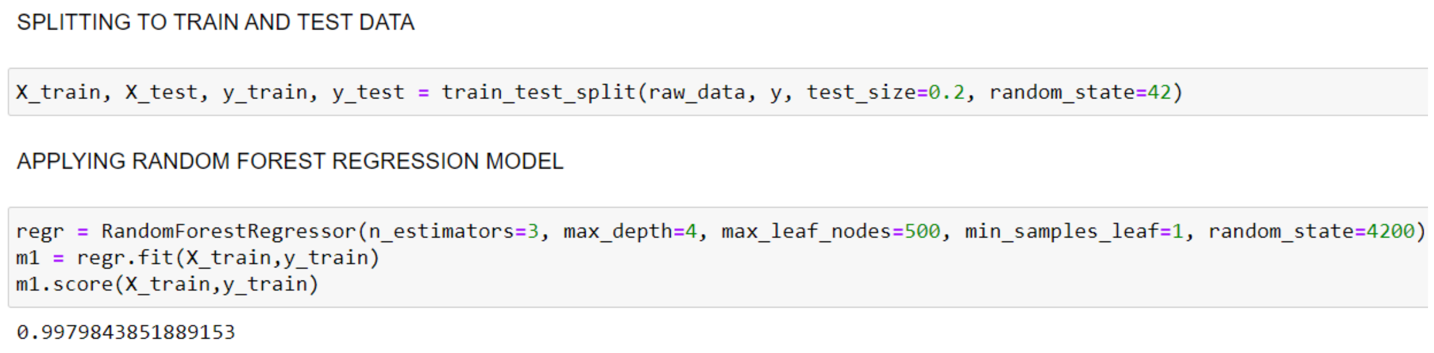
To get better insight on the behind the scenes of the ML model, we try to tweak with the hyperparameters associated with the model, Random forest Regressor in this case.

Some of The hyperparameters available in the package are:

- **N_estimators -** The number of trees in the forest.
- **Max_depth -** The maximum depth of the tree.
- **Criterion -** The function to measure the quality of a split.
- **Max_leaf_nodes -** Grow tree in best-first fashion.

Firstly we tweak the Max_depth parameter while keeping everything else the same, first we keep the max_depth = 4, and train the model and predict the results, then we visualize the result using the prediction vs actual plot and Scatter Plot.

-> max_depth = 4



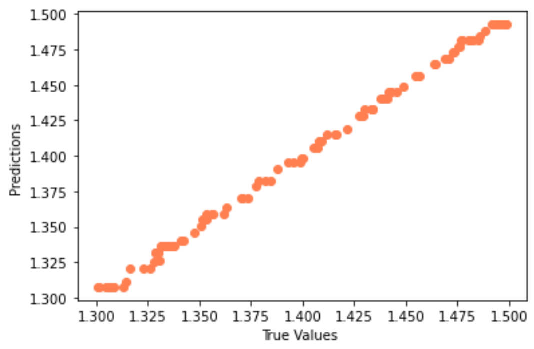
SPLITTING TO TRAIN AND TEST DATA

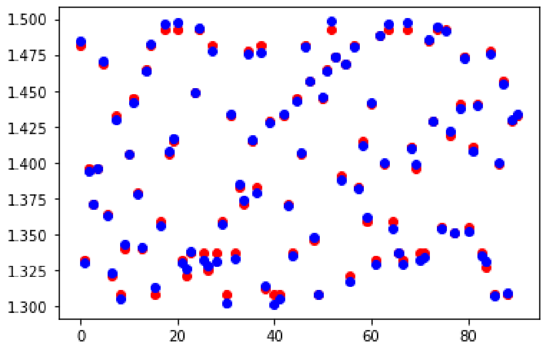
```
X_train, X_test, y_train, y_test = train_test_split(raw_data, y, test_size=0.2, random_state=42)
```

APPLYING RANDOM FOREST REGRESSION MODEL

```
regr = RandomForestRegressor(n_estimators=3, max_depth=4, max_leaf_nodes=500, min_samples_leaf=1, random_state=4200)
m1 = regr.fit(X_train,y_train)
m1.score(X_train,y_train)
```

```
0.9979843851889153
```





Now, we increase the max_depth to 8 and repeat the same steps to compare the result.

-> max_depth = 8

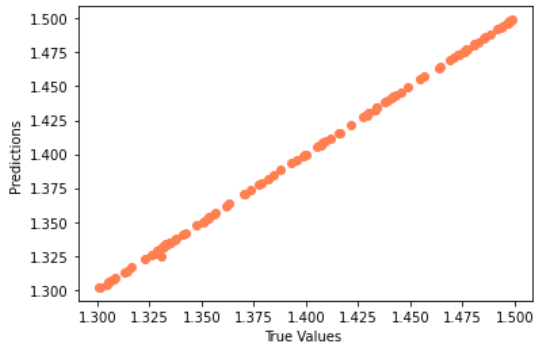
SPLITTING TO TRAIN AND TEST DATA

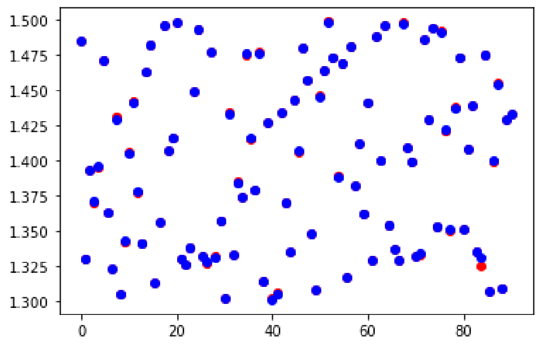
```
X_train, X_test, y_train, y_test = train_test_split(raw_data, y, test_size=0.2, random_state=42)
```

APPLYING RANDOM FOREST REGRESSION MODEL

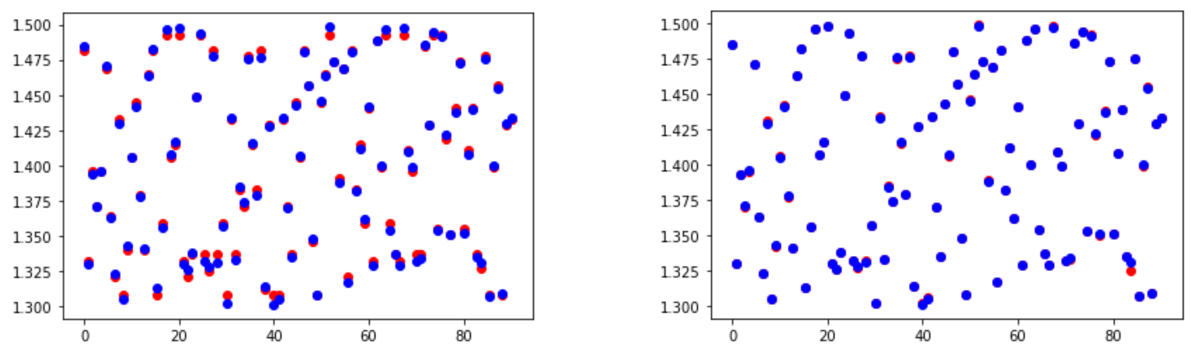
```
regr = RandomForestRegressor(n_estimators=3, max_depth=8, max_leaf_nodes=500, min_samples_leaf=1, random_state=4200)
m1 = regr.fit(X_train,y_train)
m1.score(X_train,y_train)
```

```
0.9999543993410732
```





For scatter plot, red shows predicted and blue is actual



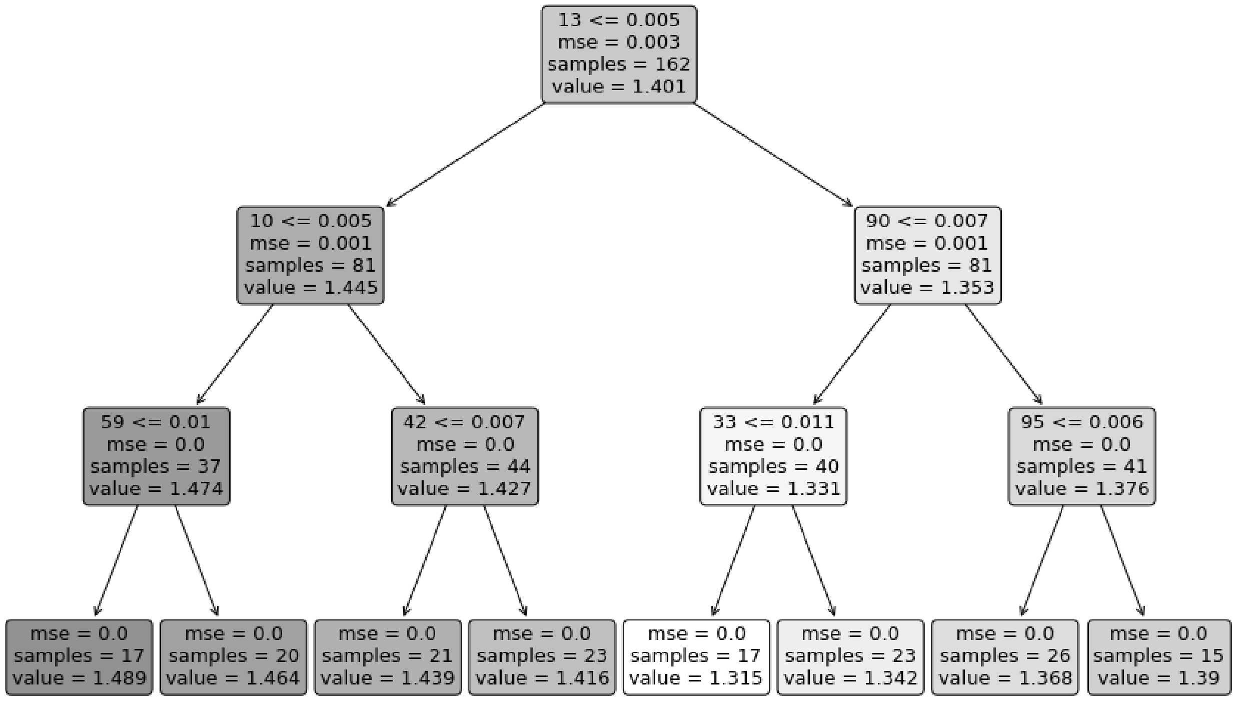
red shows predicted and blue is actual, on left is max_depth 4 and on right is max_depth=8

Accuracy score goes from 0.997 to 0.999.
This concludes increasing the maximum depth of the tree increases the accuracy but might also lead to overfitting if it's too large.

5 Tree analysis

Out of all the trees in the forest we check out any one tree to analyse and visualize the inner decision paths of the decision tree.The spectrum for which we have 99 data points acts as our features so that value at each point in range(1,100) is a feature and the target variable is refractive index.
Decision Tree tries to evenly split the data points such that at the leaf nodes(end) correct prediction is made. The prediction at leaf is the average of target values of samples at leaf and this is returned as the regressed value.

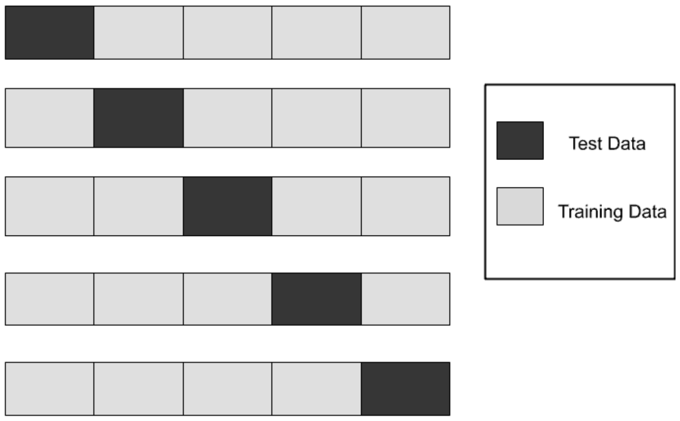
Note: the columns(corresponding to each wavelength) renumbered to 1-100 to label the features indicating their position.



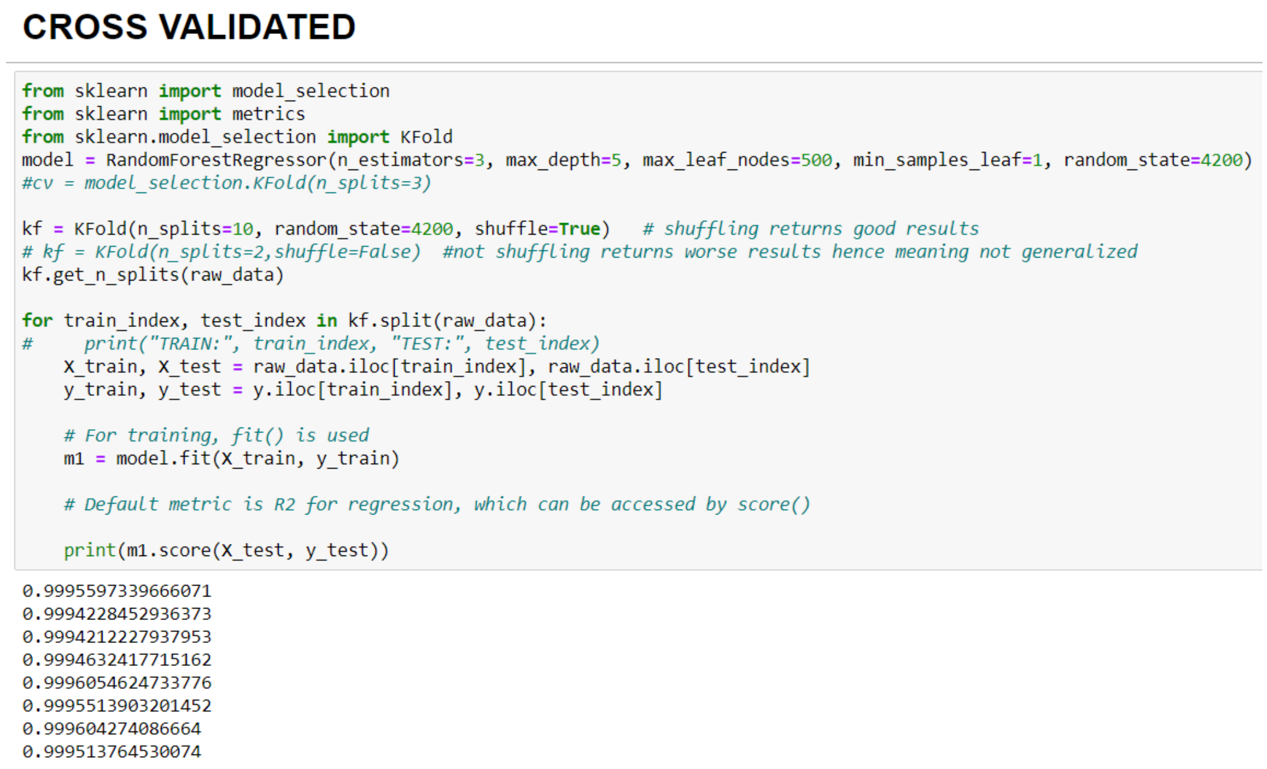
A tree of depth=3, we can observe the node splitting on values of Transmissivity at given wavelength

6 Cross Validation

Cross validation is a technique to prevent overfitting by splitting the data into certain combinations of slots/folds for assessing how the results will generalize to an independent data set. That is, it uses a limited sample to estimate how the model performs in general when it is used to make predictions on data not used/seen by the model during the training.



CODE:



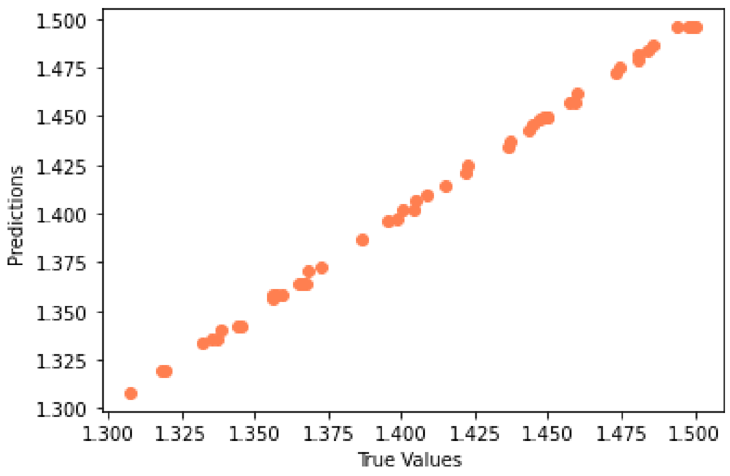
CROSS VALIDATED

```python
from sklearn import model_selection
from sklearn import metrics
from sklearn.model_selection import KFold
model = RandomForestRegressor(n_estimators=3, max_depth=5, max_leaf_nodes=500, min_samples_leaf=1, random_state=4200)
#cv = model_selection.KFold(n_splits=3)

kf = KFold(n_splits=10, random_state=4200, shuffle=True)   # shuffling returns good results
# kf = KFold(n_splits=2,shuffle=False)  #not shuffling returns worse results hence meaning not generalized
kf.get_n_splits(raw_data)

for train_index, test_index in kf.split(raw_data):
#     print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = raw_data.iloc[train_index], raw_data.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # For training, fit() is used
    m1 = model.fit(X_train, y_train)

    # Default metric is R2 for regression, which can be accessed by score()

    print(m1.score(X_test, y_test))
```

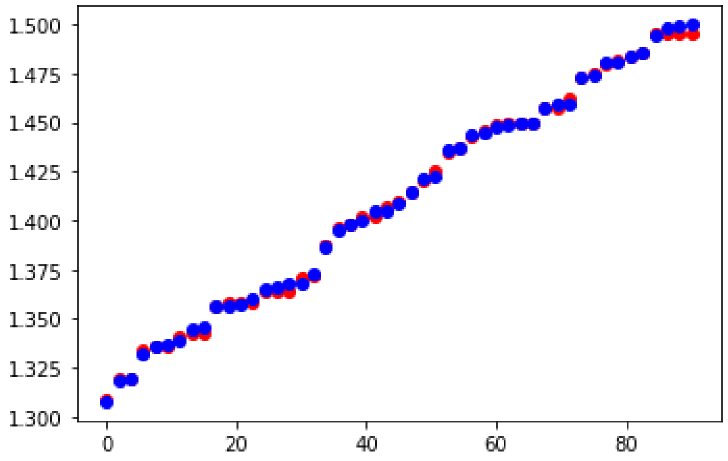
```
0.9995597339666071
0.9994228452936373
0.9994212227937953
0.9994632417715162
0.9996054624733776
0.9995513903201452
0.999604274086664
0.999513764530074
```

Observation:
observed 0.99 score for 10 fold cross validation, also observed that not shuffling data for Cross Validation returns worse results hence meaning model is not generalized. Shuffling the data solved this problem.

Results:



Predicted value vs Actual value



For scatter plot red shows predicted and blue is actual

REFERENCES:

1. M. Hamed Mozaffari and Li-Lin Tay, A Review of 1D Convolutional Neural Networks toward Unknown Substance Identification in Portable Raman Spectrometer, Arxiv June (2020)
2. Sachin Kumar Srivastava, Ibrahim Abdulhalim,
Self-referenced sensor utilizing extraordinary optical transmission from metal nanoslits array, Optics Letters Vol. 40, No. 10 (2015)
3. Jahan B. Ghasemi* and Hossein Tavakoli Application of random forest regression to spectral multivariate calibration, Analytical Methods, issue 7 (2013)
4. Georgios Drakos, Decision Tree Regressor explained in depth,Article GdCoder, (2019)
5. Chaya Bakshi, Random Forest Regression, Article GitConnected (2020)
6. Will Koehrsen, Random Forest in Python, towards data science, dec (2017)