In this project, we predict the Indoor Room Temperature by correlation and machine learning from the data of sun irradiation and outdoor temperature.

Units of data used:

1. Date: in UTC.
2. Time: in UTC.
3. Indoor temperature (room), in ºC.
4. Lighting (room), in Lux.
5. Sun irradiance, in W/m2.
6. Outdoor temperature, in ºC.
7. Outdoor relative humidity, in %.

First of all, Let us import required modules!

In [26]:
```
import os,numpy as np,pandas as pd,matplotlib.pyplot as plt,seaborn as sns
```

Now changing the working directory

In [27]:
```
os.chdir("C:/Users/Eternal/Desktop")
```

We also supress the copy warning

In [28]:
```
pd.set_option('mode.chained_assignment', None)
```

Lets read the Initial data from the .csv file and convert it into a dataframe.We also change the format of index to Date Time format.

From the data, we selected days data from 22/03/2012 to 31/03/2012.

In [29]:

```
DF_initialDataset=pd.read_csv("BuildingDataSet.csv",sep=",",index_col=0)
NewparsedIndex = pd.to_datetime(DF_initialDataset.index)  #Parsed into DateTim
DF_initialDataset.index=NewparsedIndex
DF_targetDataset=DF_initialDataset["22-03-2012":"31-03-2012"]
DF_targetDataset.describe()  #Describing targeted dataset
```

Out[29]:

	Temperature_Comedor_Sensor	Temperature_Habitacion_Sensor	Weather_Temp
count	960.000000	960.000000	960.000000
mean	19.042787	18.545009	13.570001
std	3.045278	2.974534	4.934843
min	11.352000	11.076000	5.000000
25%	16.889150	16.415525	9.000000
50%	18.995350	18.545650	14.000000
75%	21.429675	20.846975	17.733300
max	25.540000	24.944000	26.000000

8 rows × 22 columns

We select specific columns from data so to corelate target and features. Here, we will use Indoor temperature, Sun Irradiance,Outdoor Temperature,Lighting of room, Outdoor Humidity.

In [30]:

```
DF_selected=DF_targetDataset[['Temperature_Habitacion_Sensor','Meteo_Exterior_
DF_selected.rename(columns={"Temperature_Habitacion_Sensor":"Indoor Room Tempe
DF_selected["Sun Irradiance"][DF_selected["Sun Irradiance"]<0.0]=0  #Setting t
```

For giving lagged features, we first create the copy of our selected data frame. Then defining and using the function,we will create the lagged features for Sun Irradiation and Outdoor Temperature.

NOTE: The function used here only gives lagged values for Sun Irradiation(3rd,4th,5th and 6th hour) and for Outdoor temperature(1st,2nd,3rd and 4th hour) and for Indoor room temperature(30 and 60 mins before)

In [31]:
```
DF_lagged=DF_selected.copy()

'''A function to create lag columns of selected columns passed as arguments'''
def lag_column(df,column_names,lag_period):    #df> pandas dataframe,column_na
    for column_name in column_names:
        if(column_name=="Sun Irradiance"):
            for i in range(3,lag_period+1):
                new_column_name = column_name+" -"+str(i)+"hr"
                df[new_column_name]=(df[column_name]).shift(i*4)
        elif(column_name=="Outdoor Temperature"):
            for i in range(1,lag_period-1):
                new_column_name = column_name+" -"+str(i)+"hr"
                df[new_column_name]=(df[column_name]).shift(i*4)
        else:
            for i in range(2,lag_period-1,2):
                new_column_name = column_name+" -"+str(i*15)+"mins before"
                df[new_column_name]=(df[column_name]).shift(i)

    return df

DF_lagged=lag_column(DF_lagged,["Sun Irradiance","Outdoor Temperature","Indoor
DF_lagged.dropna(inplace=True)
```

Now to corelate the data, we create a heat map which provide visiual insight to correlation.

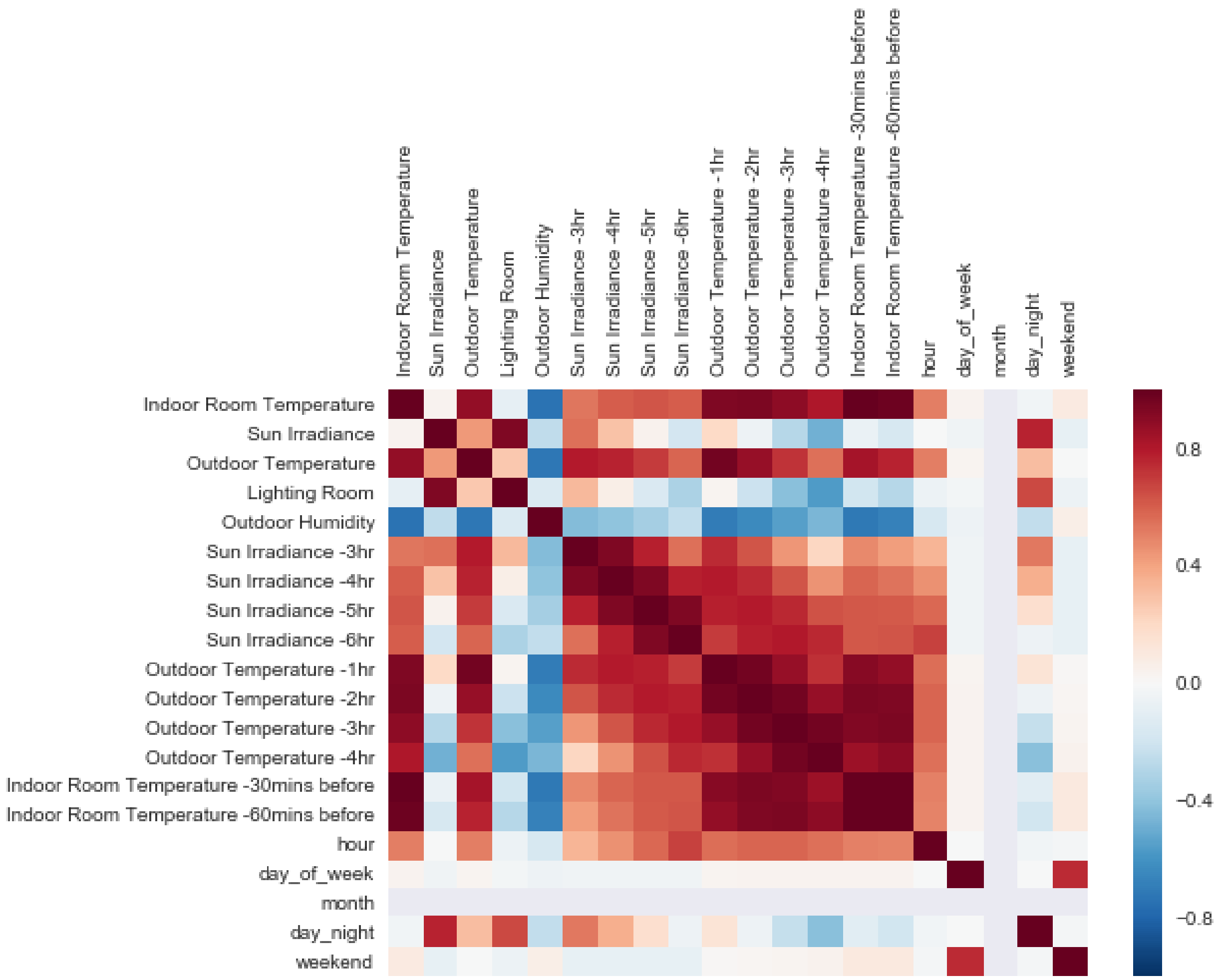
From the image below,

1)We can see that Indoor temperature of the room depends on outdoor temperature at that same hour but the sun irradiance takes time to heat up the room.

2)Lighting of the room and Outdoor Humidity doesn't contribute to Inddor temperature at all

NOTE:DARKER color means the correlation is strong and the darker blue color indicates poor coorelation.

In [54]:
```
#Creating a plot using heatmap functionality to provide correlations between c
fig = plt.figure("Figure for proving insight about Corelations")
plot = fig.add_axes()
plot = sns.heatmap(DF_lagged.corr(), annot=False)
plot.xaxis.tick_top()
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```



Plotting Indoor temperature,Sun Irradiation and Outdoor Temperature will give us greater understanding on how one depends on another.

To do so we introduce a function to normalize the data and eliminate the problem of different unit.

In the normalized graph below,

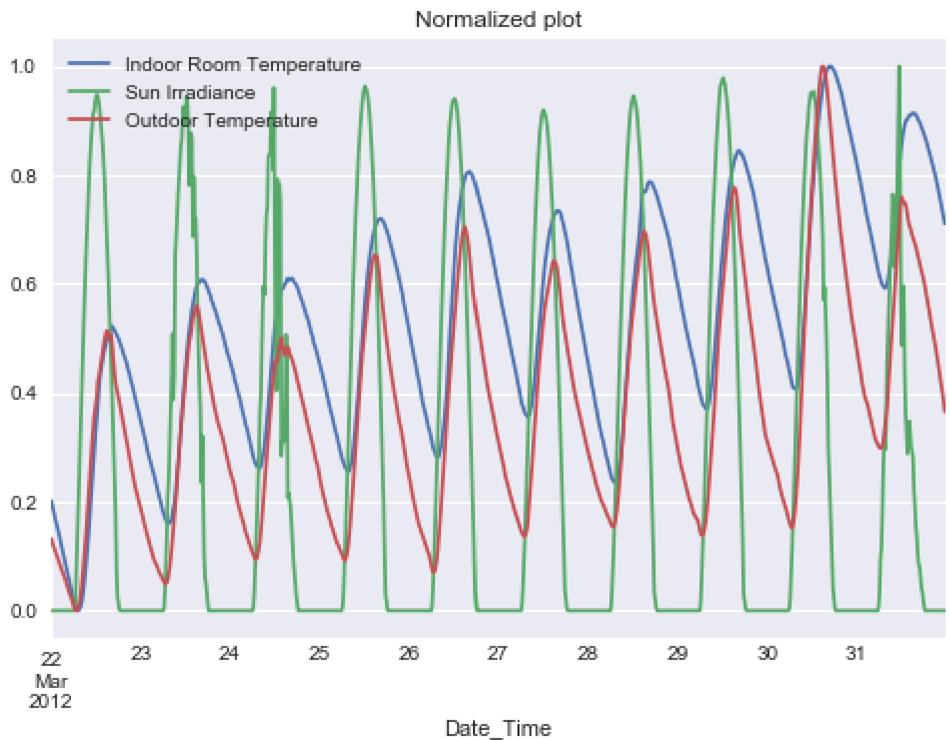
1)We can see that the Indoor temperature is increasing w.r.t. outdoor temperature in real time but it takes time for sun irradiation to heat up the room walls/roof,so the Sun Irradiation will spike up before room indoor temperature.

2)At some instances,Sun Irradiance is irregular.We can assume that sun Irradiance is irregular due to rainy/cloudy environment.

3)During night time, Sun Irradiation is zero.

In [33]:
```
'''A function to normalize dataset's columns values to provide a ranged insigh
def normalize(df):
    return (df-df.min())/(df.max()-df.min())

#Plotting the normalized dataframe with selected columns
normalizedDF=normalize(DF_selected)
PlotDF=normalizedDF[["Indoor Room Temperature","Sun Irradiance","Outdoor Tempe
PlotDF.plot()
plt.title("Normalized plot")
plt.show()
```



Let's add some time related features now like hour,Day of Week,Month, Day or night,Weekend or not etc.

We use a function to detect weekend and drop all the NaN values.

In [34]:
```
#Adding time-related features
DF_lagged['hour'] = DF_lagged.index.hour
DF_lagged['day_of_week'] = DF_lagged.index.dayofweek
DF_lagged['month'] = DF_lagged.index.month
DF_lagged['day_night'] = np.where((DF_lagged['hour']>=6)&(DF_lagged['hour']<=1

'''A function to label a day of week as weekend or not'''
def weekend_detector(day):
    if (day==5 or day==6):
        weekend = 1
    else:
        weekend = 0
    return weekend
DF_lagged['weekend'] = [weekend_detector(s) for s in DF_lagged['day_of_week']]
DF_lagged.dropna(inplace=True) #Dropping nan values

DF_lagged.head()
```

Out[34]:

	Indoor Room Temperature	Sun Irradiance	Outdoor Temperature	Lighting Room	Outdoor Humidity	Sun Irradiance -3hr	Sun Irra -4hr
Date_Time							
2012-03-22 06:00:00	11.3200	0.0000	9.38667	13.2653	64.0933	0.0	0.0
2012-03-22 06:15:00	11.2247	2.7460	9.30667	17.2207	63.9813	0.0	0.0
2012-03-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Now we have our dataframe with target and features(Time and Lagged):

In [35]:
```
target = DF_lagged[['Indoor Room Temperature']]
features = DF_lagged[[c for c in DF_lagged.columns if c not in ["Indoor Room T
```

Creating a training dataset for machine learning and providing an independent testset which follows same probabilistic distribution of training!

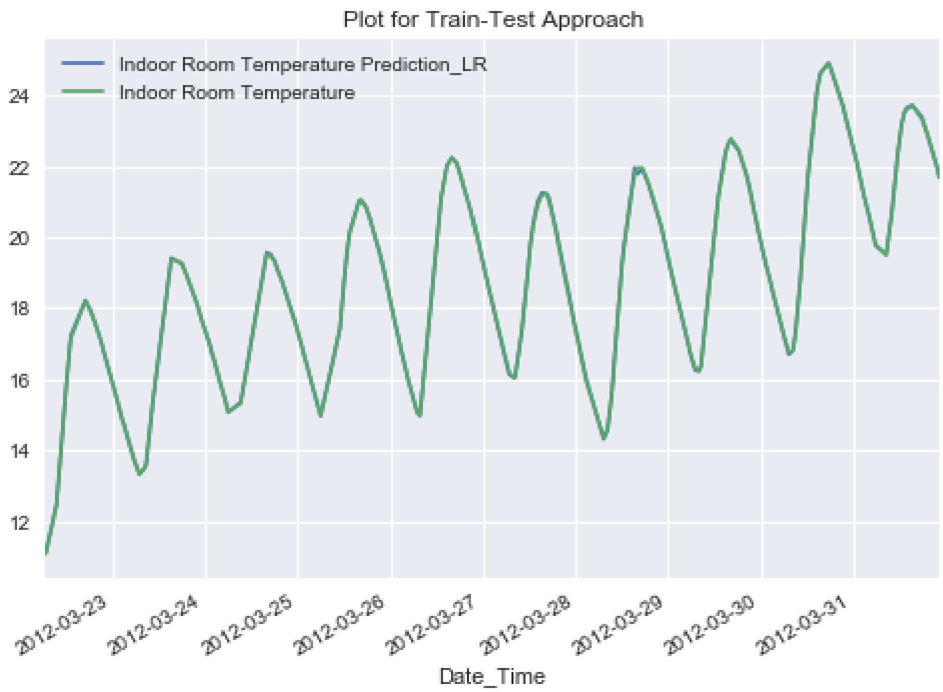
In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_siz
```

Implementing Linear regression technique of scikit-learn machine learning module to make

In [37]:
```
from sklearn import linear_model
linear_reg = linear_model.LinearRegression()
linear_reg.fit(X_train,y_train)
prediction = linear_reg.predict(X_test)
predict_series = pd.Series(prediction.ravel(),index=y_test.index).rename('Indo
LearnedDataset = pd.DataFrame(predict_series).join(y_test).dropna()
```

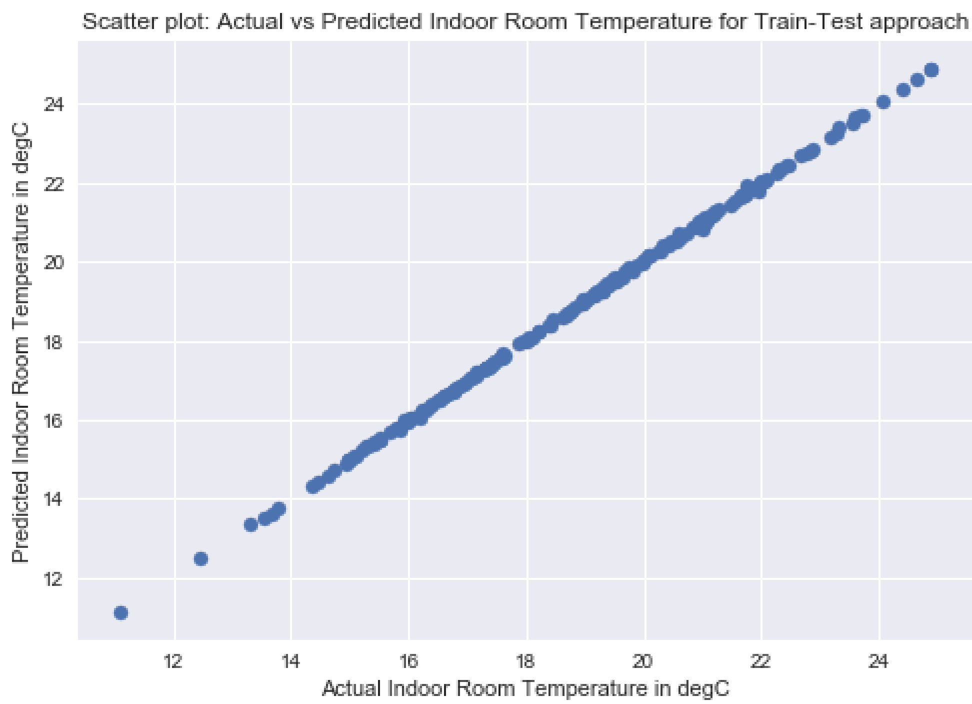
Plotting the learned dataset and verifying the predicted values with actual ones

In [38]:
```
LearnedDataset.plot()
plt.title("Plot for Train-Test Approach")
plt.show()
```



In [39]:

```
plt.figure()
plt.scatter(LearnedDataset['Indoor Room Temperature'],LearnedDataset['Indoor R
plt.xlabel("Actual Indoor Room Temperature in degC")
plt.ylabel("Predicted Indoor Room Temperature in degC")
plt.title("Scatter plot: Actual vs Predicted Indoor Room Temperature for Train
plt.show()
```



Calculating the accuracy metrics of implemented machine learning model

In [40]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
metric_R2_score = r2_score(y_test,prediction) #Perfectly accurate model gives
metric_mean_absolute_error = mean_absolute_error(y_test,prediction)
metric_mean_squared_error = mean_squared_error(y_test,prediction)
coeff_variation = np.sqrt(metric_mean_squared_error)/float(y_test.mean())

print "The R2_score is "+str(metric_R2_score)
print "The mean absoulute error is "+str(metric_mean_absolute_error)
print "The mean squared error is "+str(metric_mean_squared_error)
print "Coefficient variation : "+str(coeff_variation)
```

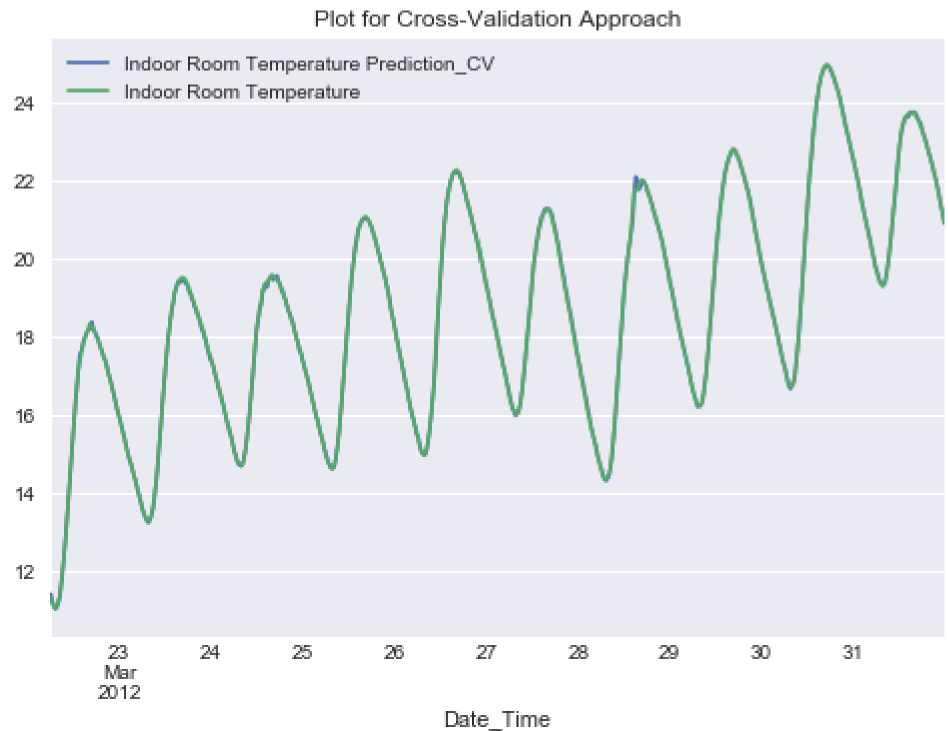
```
The R2_score is 0.999706187861
The mean absoulute error is 0.0332309372913
The mean squared error is 0.00220430612676
Coefficient variation : 0.00247818617935
```

Implementing cross-validation approach to test the performance of the machine learning model

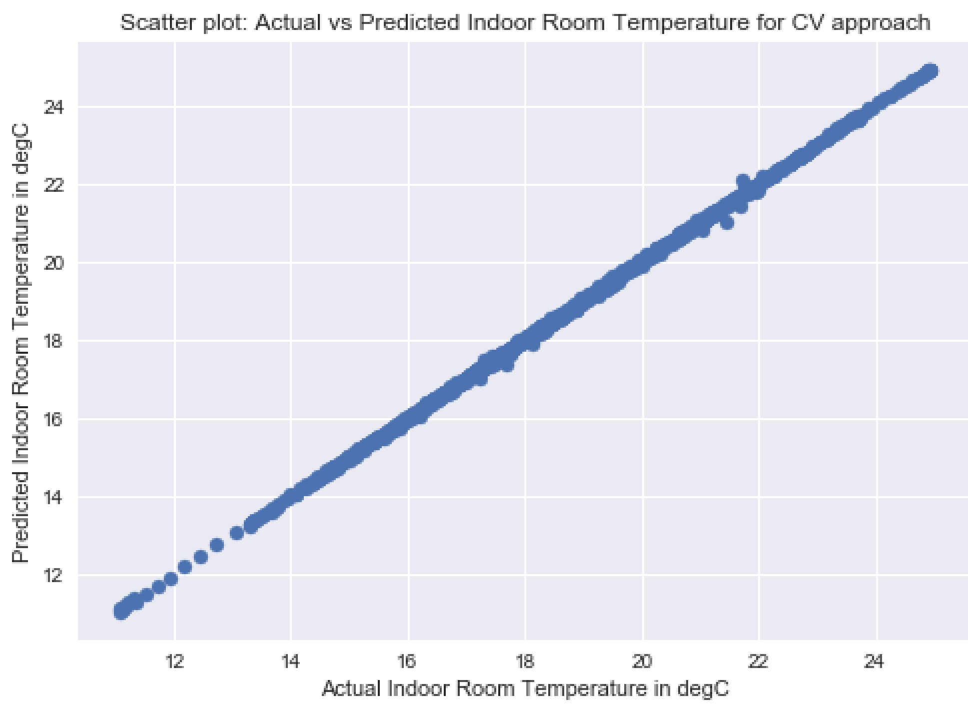
In [41]:
```
from sklearn.model_selection import cross_val_predict
predict_linearReg_CV = cross_val_predict(linear_reg,features,target,cv=10)
predict_DF_linearReg_CV=pd.DataFrame(predict_linearReg_CV, index = target.inde
predict_DF_linearReg_CV = predict_DF_linearReg_CV.join(target)
```

Plotting the learned dataset and verifying the predicted values with actual ones

In [42]:
```
predict_DF_linearReg_CV.plot()
plt.title("Plot for Cross-Validation Approach")
plt.show()
```



In [43]:
```
plt.figure()
plt.scatter(predict_DF_linearReg_CV['Indoor Room Temperature'],predict_DF_line
plt.xlabel("Actual Indoor Room Temperature in degC")
plt.ylabel("Predicted Indoor Room Temperature in degC")
plt.title("Scatter plot: Actual vs Predicted Indoor Room Temperature for CV ap
plt.show()
```



Implementing cross-validation approach to test the performance of the machine learning model

Implementing Random Forest regression technique of scikit-learn machine learning module to make

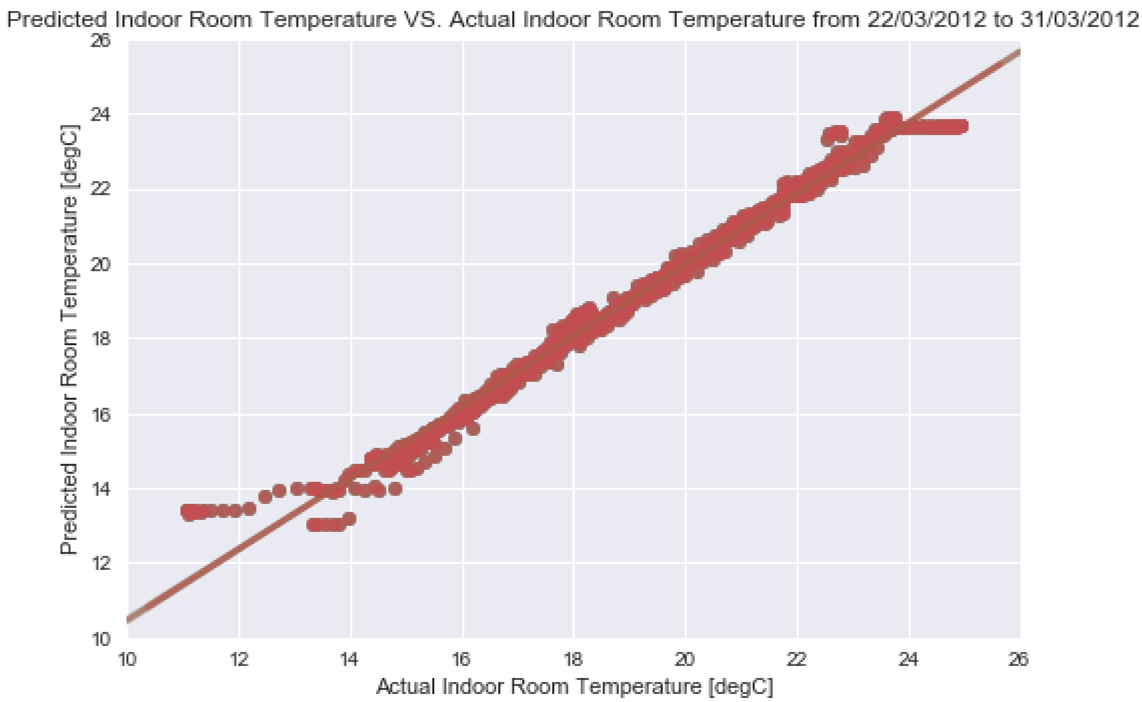
In [52]:
```
from sklearn.model_selection import cross_val_predict #Plotting the learned da

fig
fig = plt.figure("Actual Vs Prediction by Random Forest")
ax1 = fig.add_subplot(111)

from sklearn.ensemble import RandomForestRegressor
reg_RF = RandomForestRegressor()
predict_RF_CV = cross_val_predict(reg_RF,features,target,cv=10)
predict_DF_RF_CV=pd.DataFrame(predict_RF_CV, index = target.index,columns=["In
predict_DF_RF_CV = predict_DF_RF_CV.join(target).dropna()
```

Plotting the learned dataset and verifying the predicted values with actual ones

In [48]:
```
fig = plt.figure("Actual Vs Prediction by Random Forest")
ax1 = fig.add_subplot(111)
plot = sns.regplot(x="Indoor Room Temperature", y="Indoor Room Temperature Pre
                   data=predict_DF_RF_CV,ax=ax1,
                   line_kws={"lw":3,"alpha":0.5})
plt.title('Predicted Indoor Room Temperature VS. Actual Indoor Room Temperatur
plot.set_xlim([10,26])
plot.set_ylim([10,26])
plot.set_xlabel('Actual Indoor Room Temperature [degC]')
plot.set_ylabel('Predicted Indoor Room Temperature [degC]')
regline = plot.get_lines()[0];
regline.set_color('red')
plt.show()
```



Calculating the accuracy metrics of implemented machine learning model

In [46]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
R2_score_DF_RF_CV = r2_score(predict_DF_RF_CV["Indoor Room Temperature"],predi
mean_absolute_error_DF_CV = mean_absolute_error(predict_DF_RF_CV["Indoor Room
mean_squared_error_DF_CV = mean_squared_error(predict_DF_RF_CV["Indoor Room Te
coeff_variation_DF_CV = np.sqrt(mean_squared_error_DF_CV)/predict_DF_RF_CV["In
print "\nThe R2_score is: "+str(R2_score_DF_RF_CV)
print "The Mean absoulute error is: "+str(mean_absolute_error_DF_CV)
print "The Mean squared error is: "+str(mean_squared_error_DF_CV)
print "The Coefficient of variation is: "+str(coeff_variation_DF_CV)
```

```
The R2_score is: 0.983795990273
The Mean absoulute error is: 0.18619442646
The Mean squared error is: 0.13187533598
The Coefficient of variation is: 0.0194236562328
```

In [ ]:

```
```