IOT Project Report

Component # 1: Data Collection

Connecting ESP32 to Cloud

For connecting ESP32 to google sheets, we used IFTTT(if this then that). We used DHT11 sensor to collect the temperature and humidity readings for five days in different location and published it in google sheet with the help of IFTTT. We collected data at a sampling rate of 2 minutes for 10 hours (**continuous two hours on five different days**).

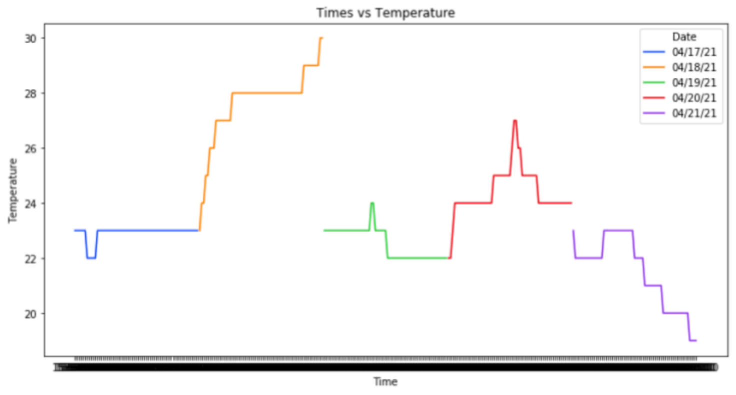
For using IFTTT to integrate with google sheets, we created IFTTT account which is free. Then, we created google sheets named Temp_Humidity Readings and connect it to ESP32. The unique API key was provided which gives access to the google sheet. We wrote ESP32 code in Thonny IDE for reading temperature and humidity sensor data.In the code, DHT.measure() measures the values from sensor whereas DHT.temperature() and DHT.humidity() reads the temperature and humidity values respectively and stored the sensor readings in value 1 and value 2 in json format. We used post method to send the data in google sheet creating the request url. sleep_ms(120000) reads the values in every 2 minutes from sensor. The data is stored in google sheet Temp_Humidity Readings with columns Timestamp, Temperature and Humidity.
The url link of our google sheet is https://docs.google.com/spreadsheets/d/1OQhfSJclKgIxRU3H56LTIzGReKjiU1iDY5NPgRjgT78/edit?ts=60818346#gid=0 . I downloaded this sheet and save file as temp_humidity.csv and used it for further data plotting and analysis.I have attached the fully commented code in Dropbox.

Component # 2: Data Plotting

After the collection of data, I used jupyter notebook for data plotting and analysis part.
Firstly, I imported all the necessary libraries and read the csv file using pandas Dataframe.
I splitted the timestamp of my data to Date and Time for data plotting with the lambda function.

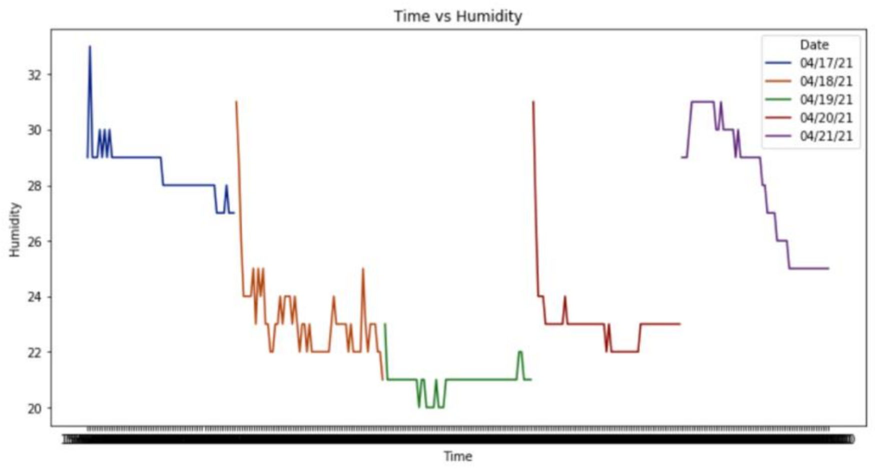
- **Line Plot**
  For line plot, seaborn is used which is a data visualization library in python based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
  For line plot between Time and temperature following code is used.
  sns.lineplot(data=data, x="Time", y="Temperature", hue="Date",palette="bright")

  In line plot, time is x-axis and temperature is y-axis. hue determines which column in the dataframe should be used for color encoding. In our case date is the hue. palette is the color for the lines.



In above figure, the lines with different colors shows the temperature recorded in different dates for two hours of time. The highest temperature recorded is on 18th April and lowest temperature recorded is on 21th April.
For line plot between Time and Humidity following code is used.

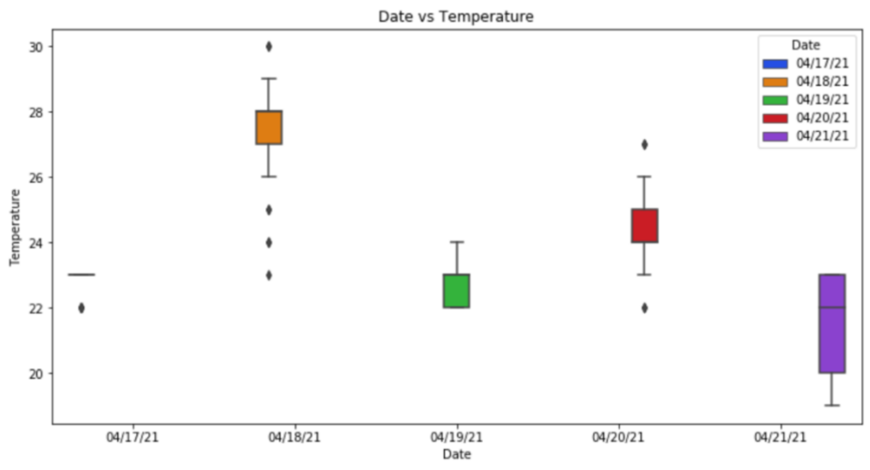
```
sns.lineplot(data=data, x="Time", y="Humidity", hue="Date",palette="dark")
```



In above figure, the lines with different colors shows the humidity recorded in different dates for two hours of time. The highest humidity recorded is on 17th April and lowest humidity recorded is on 19th April.

- **Box plot**
  For box plot between date and temperature, following code is used.

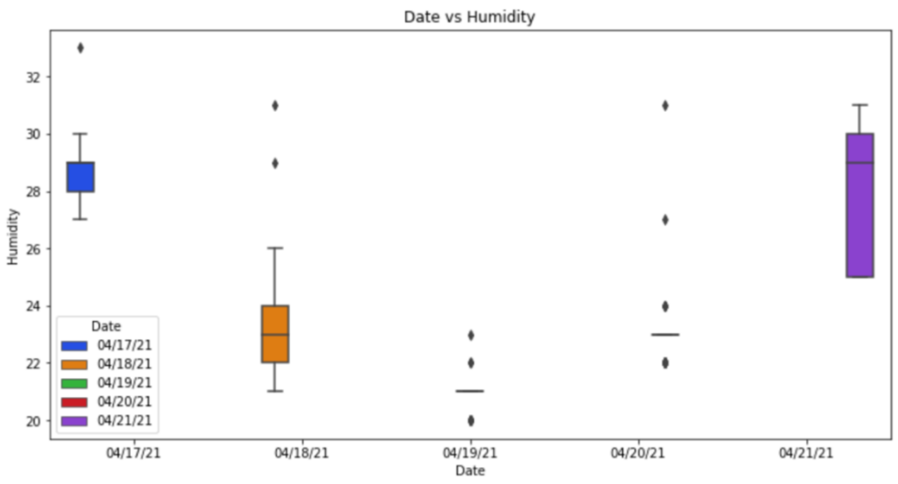
```
sns.boxplot(data=data, x="Date", y="Temperature",hue='Date',palette="bright")
```



In above figure, each box shows the temperature recorded in different five dates. On 17th April the temperature is constant 23 degree Celsius. On boxplot of 18th April, dots are seen which mean outliers i.e. temperature that differs significantly from other rest of the data so located outside the whiskers of boxplot.The whiskers show the temperature data that are mostly recorded.

For box plot between date and humidity, following code is used.

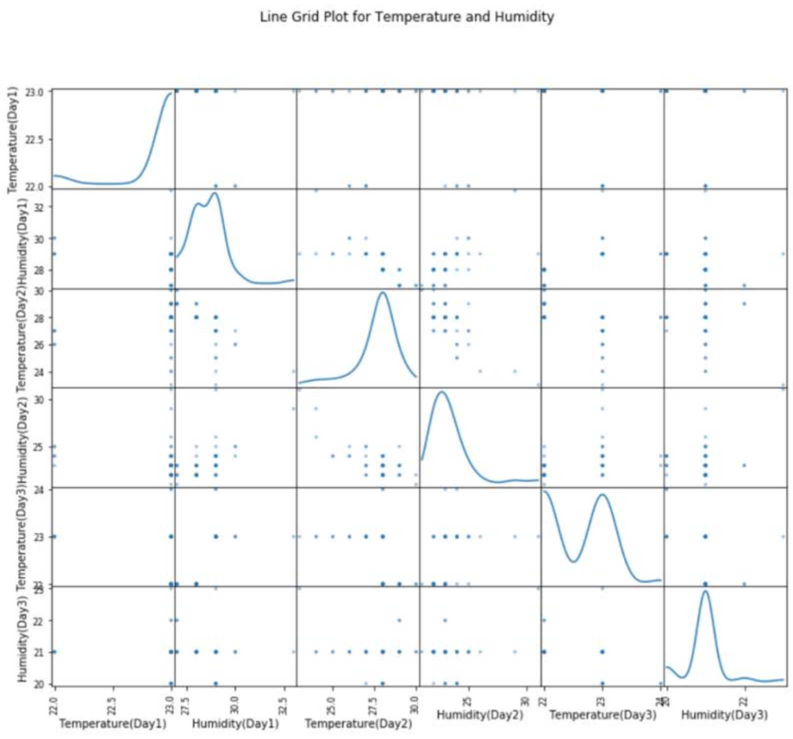
```
sns.boxplot(data=data, x="Date", y="Humidity",hue='Date',palette="bright")
```

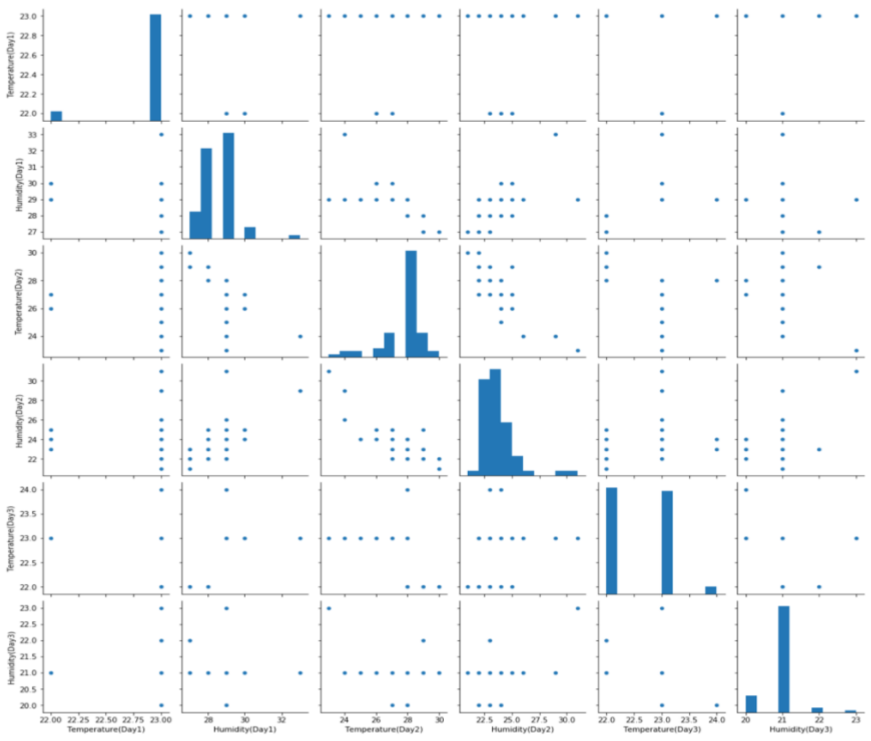


In above figure, different boxes shows the humidity recorded in different five dates. On 19th and 20th April, the Humidity values are mostly constant and outliers are also present which shows humidity values significantly different from rest of the data.

Line Grid Plot

For line grid plot of temperature and humidity of three days, I made new excel file splitting the temperature and humidity of three different days. Then, I used sns.pairplot for plotting.





Component # 3: Data Analysis

Data pre-processing:

The collected dataset has the following variables: timestamp, temperature, and humidity. For extracting time from the timestamp column and representing it in minutes form, I used the following python code.

```
def minutes(Time):
   hours,minutes,seconds = tuple(Time.split(':'))     #Split the hours, minutes and seconds
   return int(hours)*60+int(minutes)                    #Returning time in minutes form
data['Minutes']=data['Time'].apply(lambda x: minutes(x))
```

Creating Linear regression Model

I created a simple Linear regression model M1 for between temperature and minutes, M2 between humidity and minutes and multiple linear regression model M3 between temperature, humidity and minutes.

Linear regression is a linear approach to modelling the relationship between dependent and independent variables. It is used when we want to predict the value of a variable based on the value of another variable. The variable we want to predict is called the dependent variable (or sometimes, the outcome variable). The variable we are using to predict the other variable's value is called the independent variable (or sometimes, the predictor variable).

The linear regression equation has the form y= a + bX, where y is the dependent variable (that's the variable that goes on the Y axis), X is the independent variable (i.e. it is plotted on the X axis), b is the slope of the line and a is the y-intercept.

I created the regression model M1 and fit it using the code:

```
X1 = data['Temperature'].values.reshape(-1,1)     #Taking temperature as independent variable i.e. X axis
y1 = data['Minutes'].values.reshape(-1,1)           #Taking minutes as dependent variable i.e. Y axis
M1 = LinearRegression()                #creating model M1
M1.fit(X1, y1)                         #Fitting the model
print(M1.coef_[0][0])                  #printing the coefficient value
print(M1.intercept_[0])                #printing the intercept value
print("The linear model is: Y = {:.5} + {:.5}X".format(M1.intercept_[0], M1.coef_[0][0]))
predictions1 = M1.predict(X1)         #Predicting the model
```

Similar code is used for creating model M3.

Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable. In model M3, we have Humidity and minutes as independent variables and temperature as dependent variable. So, I used the following code for creating the model.

```
X3 = data[['Humidity','Minutes']]
y3 = data['Temperature']
M3 = LinearRegression()
```

Computing Mean Squared Error and AIC for M1, M2,and M3.

Mean Squared Error measures the average of error squares i.e. the average squared difference between the estimated values and actual values. The **mean squared error** tells how close a regression line is to a set of points. It is always non – negative and values close to zero are better.
It is calculated with the following codes.

```
error1=mean_squared_error(y1,predictions1)
print('Mean Squared Error for model M1:',error1)
```

In my case, Mean Squared Error for model M1 is **25836.515725445217**, M2 is **28708.963877620474** and M3 is **3.673273366417648.** So, comparing these results , Model M3 is better than other models as its value is close to zero.

AIC(Akaike information criterion) is a mathematical method for evaluating how well a model fits the data it was generated from. AIC is used to compare different possible models and determine which one is the best fit for the data. Negative AIC indicates less information loss than a positive AIC and therefore a better model.
AIC is calculated with the following codes.

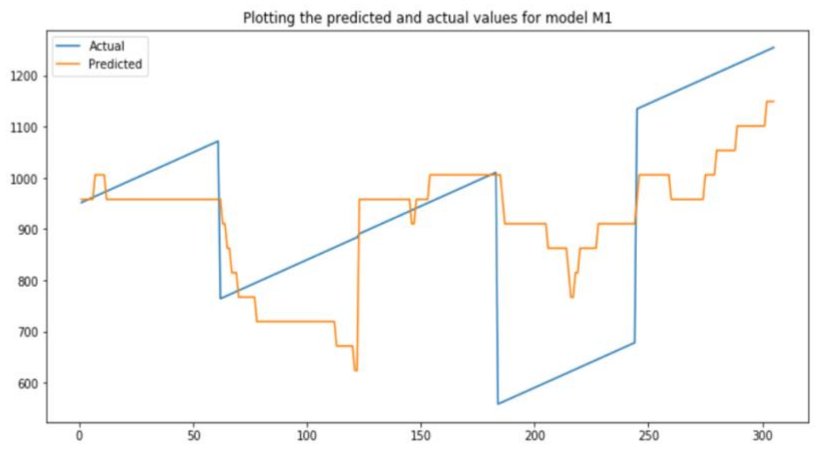
```
resid1 = y1 - predictions1
sse_M1 = sum(resid1**2) # sum of squares
AIC_M1 = 2*1 - 2*np.log(sse_M1)
print('AIC of model 1',AIC_M1)
```

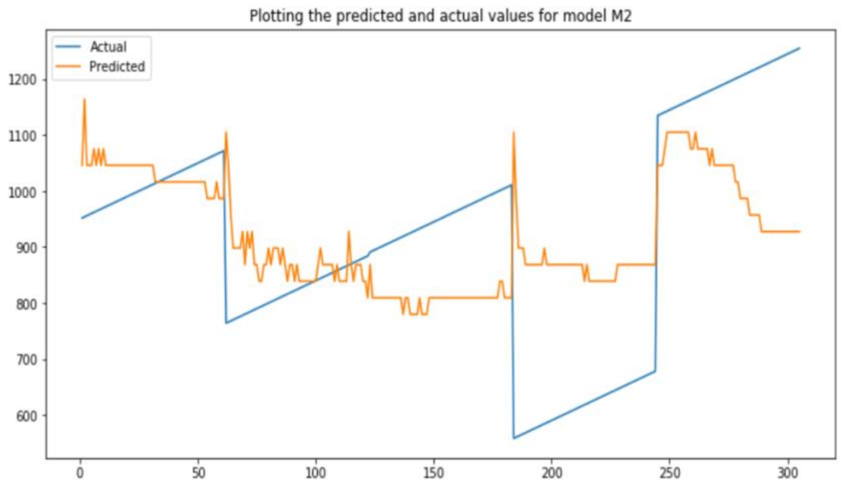
AIC for my model M1, M2 and M3 are **-29.75971177** , **-29.97055292** and **-12.042789933198286** respectively. Comparing the results, Model M2 is better as it has more negative value which means less information loss.

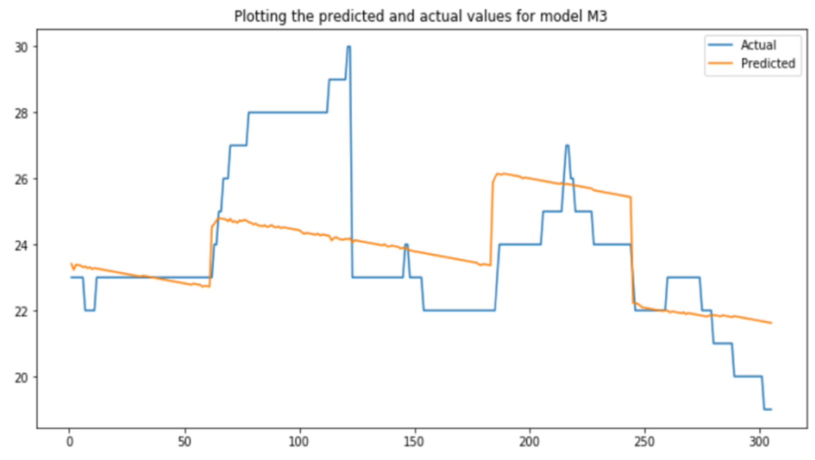
For plotting predicted and actual values of the Model M1, following code is used.

```
df1 = pd.DataFrame({'Actual':y1.flatten(),'Predicted':predictions1.flatten()})
df1.index = range(1,len(y1)+1)
df1.plot(title="Plotting the predicted and actual values for model M1",figsize=(12,6))
```

Similar code is used for model M2 and M3.


Plotting the predicted and actual values for model M1
Actual
Predicted


Plotting the predicted and actual values for model M2
Actual
Predicted


Plotting the predicted and actual values for model M3
Actual
Predicted