National Taipei University of Technology
Computer Science and Information Engineering

Principles and Applications of Data Science

Spring 2022

Semester Group Project Report

The relationship between temperature, humidity and human stress

Name: 顏翊純、黃彥文

Sid: 110599002、109598113

Date:06/22/2022

Content(目錄)

Abstract

With the progress of society and the development of science and technology, people under the great pressure at the same time. We realized that there are much people suffer from Depression or Bipolar Disorder in our daily life, even elementary students. These mental illnesses are not only caused by personal or society reason, but also depend on the nature environment [1]. Just like humid weather will make people feel depress than sunny. We hoped that we can create a model which can predict the stress, by this way, people can do something relaxing at the beginning to make these stress don't grow larger and larger.

In this project we proposed a tool to predict the stress level by temperature and humidity. We used the data provided from papers[1] written by L. Rachakonda, S. P. Mohanty, E. Kougianos, and P. Sundaravadivel [2][3] to do the analysis. We do the linear analysis and try several algorithms to create the predicting model, e.g. K-NN, SVM, Adaboost. In this prediction, the stress level are classifier to 0, 1, 2, higher number means higher stress. If the prediction get stress level 2, the tool will provide the ten most restful activities to the user.

We also used the Taiwan weather data provided by Taiwan Ministry of Transportation and Communications[2]. We combine the temperature and humidity data each month from year 108 to year 110 to do the stress level prediction. The result also shows on the tool.

[1] https://www.kaggle.com/datasets/laavanya/stress-level-detection
[2] https://stat.motc.gov.tw/mocdb/stmain.jsp?sys=100&funid=a8101

1. Introduction

Motivation:

New York Times has written the article, << His College Knew of His Despair. His Parents Didn't, Until It Was Too Late.>>, using "despair" to talking about the university students may go through in America. And as we realized that there are much people suffer from Depression or Bipolar Disorder in our daily life. We want to found that if there is something we can do before these happens. We start to search for human stress, and two papers written by L. Rachakonda, S. P. Mohanty, E. Kougianos, and P. Sundaravadivel were found. They create an IoT device which can detect the data from human body that can help to detect the stress immediately. We used the data provided by this project to create a model which can predict the human stress by temperature and humidity.

We found that most of them have some common personality, they suffer a big stress at the end, but they are not conscious of when these stresses started. We hoped that we can create a model which can predict the stress, by this way, people can do something relaxing at the beginning to make these stress don't grow larger and larger.

Objectives:

Create a model which can predict the human stress by temperature and humidity data.

Although the temperature and humidity are the body temperature and humidity in these papers, we've known that the weather will also influence the frequency of symptoms, which called "Seasonal affective disorder (SAD)".

Application and Contributions:

These mental illnesses somehow caused by the nature environment, by this tool, it may help people to predict the stress, and it can do some restful activities to release these stresses than suffer large stresses at the end and caused the mental illnesses. The point above is not only for the normal person but also the person who was suffered by mental illnesses before and get well now. For people who are suffered by mental illnesses before, it's easy for them to get mental illnesses again if they don't have great control. It's important for them to find out if there is stress in their mind or not and this tool may help.

2. Literature review and related works

Literature review:

L. Rachakonda, S. P. Mohanty, E. Kougianos, and P. Sundaravadivel wrote paper in 2018, "A Smart Sensor in the IoMT (Internet of Medical Things) for Stress Level Detection"[2]. In this paper, it's talking about a device created as a band which has sensors in it to detect temperature, heart rate, accelerometer…etc. These data will be the input of the Deep Learning or Deep Neural Networks (DNN) models, and the output will be the stress level. The paper they presented in 2019, "Stress-Lysis: A DNN-integrated edge device for stress level detection in the IoMT"[3] is talking about the novel contribution, like using Mamdani fuzzy logic for accurate stress detection, combines not only one data to do the prediction, quickly to detect stress level is presented and a novel IoMT-enabled system for stress analysis at the edge and not at the cloud is proposed, thus advancing the state-of-the-art in the IoMT.

For choosing the machine learning algorithms, especially SVM, we've searched for the using of different kernels, but at the end we still decide to do 4 kernels to have the comparison. Like RBF is better to using on the non-linear data, linear SVC is different from the linear kernel, first one using the square to do the equation, last one using the absolute.

Related works:

There is someone on Kaggle use this data to do the prediction three months ago, after we decided to do this project one month later. Their code[3] is using Logistic Regression to do the experiment. Logistic Regression[4] is also a kind of classifier algorithms, which is used for predicting the categorical dependent variable using a given set of independent variables.

Algorithm	Accuarcy
Logistic Regression (On Kaggle)	**0.997506**
K-NN (Our Project)	**0.9975**
SVM Linear SVC (Our Project)	**0.9975**
Decision Tree (Our Project)	**0.997506**

[3] https://www.kaggle.com/code/souravbhandari/predicting-stress-level/notebook
[4] https://www.ibm.com/topics/logistic-regression

3. Problem statement

Weather impacts person's mood, especially rainy day. Nowadays, with the progress of the society, many people don't have time to conscious that they are suffered by huge stresses. That's also one of the reasons that more and more people suffer from the mental illness these days. In addition, conscious the status of oneself can also help to control the temper which can avoid the argue situation and improve the communication.

Scenario 1:

Rose is a university student who suffered by depression. It's not easy for her to detect emotions herself. She hopes that if there is any tool can help her to find herself in the stressful way. By this way, she can release some of the stress earlier than accumulated pressure at the end and feel depress without doing anything.

Scenario 2:

Burt is a manger in the company, sometimes he felt that he's easily get anger without any reason. By the stress predict, he can know that today will get a more stressful mood or more relax mood. It can help him to control the temper more easily.

4. Proposed models (approaches)

We do two kinds of linear analysis first, and check their relationship (See Experiment Section). We choose different kinds of algorithm to do the comparison, KNN, SVM and Decision Tree. The data source classify stress to three level, 0, 1, 2, higher number means higher stress, so we choose classifier algorithms instead of regression algorithms.

We take the temperature and humidity column as the input data, and the stress level column as the output data. For each algorithm we use 60% data as training data, 20 % as verification data, 20% as test data. As the figure below, we split 80% data as training data first, and 20% data as test data, and for the 80% train data, we split 25% as verification data. By this way, we get 60% data as training data, 20 % as verification data, and 20% as test data. The reason we use one more verification data, not only training data and test data, is that if we always use the same test data, we may try to adjust the parameters to make the accuracy higher, and the model may become overfitting at the end.

```
In [1]: import pandas as pd

        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
```

```
In [2]: lysis_data=pd.read_csv("./Stress-Lysis.csv")
        print(lysis_data.head())
```

```
   Humidity  Temperature  Step count  Stress Level
0     21.33        90.33         123             1
1     21.41        90.41          93             1
2     27.12        96.12         196             2
3     27.64        96.64         177             2
4     10.87        79.87          87             0
```

```
In [3]: # print(lysis_data[['Humidity', 'Temperature']])
        hum_temp_data = lysis_data[['Humidity', 'Temperature']]
        str_lev_data = lysis_data['Stress Level']
```

```
In [4]: train_data , test_data , train_label , test_label = train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
        train_data , ver_data , train_label , ver_label = train_test_split(train_data, train_label, test_size=0.25)
```

K-NN:

The KNN algorithm [5]assumes that similar things exist in close proximity. In other words, similar things are near to each other. It works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

```
In [5]: model = KNeighborsClassifier(n_neighbors=3)

        # Train the model using the training sets
        model.fit(train_data, train_label)

        # Predict Output
        predicted = model.predict(ver_data) # 0:Overcast, 2:Mild

        print("accuracy: ", accuracy_score(ver_label, predicted))
```

```
accuracy:  0.9975
```

```
In [6]: predicted = model.predict(test_data)
        print("accuracy: ", accuracy_score(test_label, predicted))
```

```
accuracy:  1.0
```

[5] https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

SVM:

SVM[6] works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

SVM has different kernel for different data situation[7], our data is great to use the linear one. But we want to compare different kernels and get different results.

Linear SVC is the best one for our situation, for our data is linear, classify into three level, and the difference between linear kernel and linear SVC kernel[8] is that linear SVC use square hinge loss, and SVC use absolute hinge loss.

In [6]:
```
C = 2 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=C).fit(train_data, train_label)

pred_ver = svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 0.9975
Test Accuracy: 0.9925187032418953
```

In [10]:
```
C = 2 # SVM regularization parameter

rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(train_data, train_label)

pred_ver = rbf_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = rbf_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 0.9975
Test Accuracy: 0.9950124688279302
```

[6] https://www.ibm.com/docs/it/spss-modeler/SaaS?topic=models-how-svm-works
[7] https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
[8] https://blog.csdn.net/qq_23069955/article/details/80961186

In [11]:
```
C = 2 # SVM regularization parameter

poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(train_data, train_label)

pred_ver = poly_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = poly_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 1.0
Test Accuracy: 0.9950124688279302
```

In [12]:
```
C = 2 # SVM regularization parameter

lin_svc = svm.LinearSVC(C=C, dual=False).fit(train_data, train_label)

pred_ver = lin_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = lin_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 0.9975
Test Accuracy: 0.9975062344139651
```

Decision Tree:

Decision Tree [9]creates a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from training data. In Decision Trees, for predicting a class label for a record, it starts from the root of the tree. It compares the values of the root attribute with the record's attribute. On the basis of comparison, it follows the branch corresponding to that value and jump to the next node.

We do Adaboost Classifier[10] and Adaboost Decision Tree[11] at the same time to do the comparison[12], and found that they get the same result.

[9] https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html
[10] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
[11] https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_regression.html
[12] https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

In [5]:
```
model =  AdaBoostClassifier(DecisionTreeClassifier(max_depth=2), n_estimators=50)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 1.0
Test Accuracy: 0.9975062344139651
```

In [8]:
```
model =  AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 1.0
Test Accuracy: 0.9975062344139651
```

In [9]:
```
model = AdaBoostClassifier(n_estimators=50, random_state=0)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train, acc_test))
```

```
Verification Accuracy: 1.0
Test Accuracy: 0.9975062344139651
```

5. Experiments

We do the linear analysis first as the pictures below. We do two kinds of analysis, first one is using stress level with two label to do the analysis, red: stress 0, blue: stress 1, yellow: stress 2. Second, we using any of two label to do the linear analysis. Through the analysis, we found:

- the higher temperature, the higher pressure
- the higher humidity, the higher pressure
- the higher the pressure, the more steps you take.

After doing linear analysis, we use different algorithms to create model by using Jupyter notebook (in src folder) to see the accuracy.

Then we transfer the Jupyter notebook code to python code, for we want to create a window application which can enter the humidity and temperature to predict the stress level, if user get stress level 2, it may show the ten of most restful activities. User can also choose which model they want to use.

We also collect Taiwan's temperature and Humidity each month from year 108 to year 110, the stress level predict result is also be shown on the window application.

All analysis and different model accuracy result is shown on window application, too.

Tools:

1. Jupyter notebook

2. Python Library: sklearn, pandas, tkinter

Data Sources:

From paper "A Smart Sensor in the IoMT for Stress Level Detection"[2] and "Stress-Lysis: A DNN-integrated edge device for stress level detection in the IoMT"[3] written by L. Rachakonda, S. P. Mohanty, E. Kougianos, and P. Sundaravadivel. They share the data on Kaggle[13].

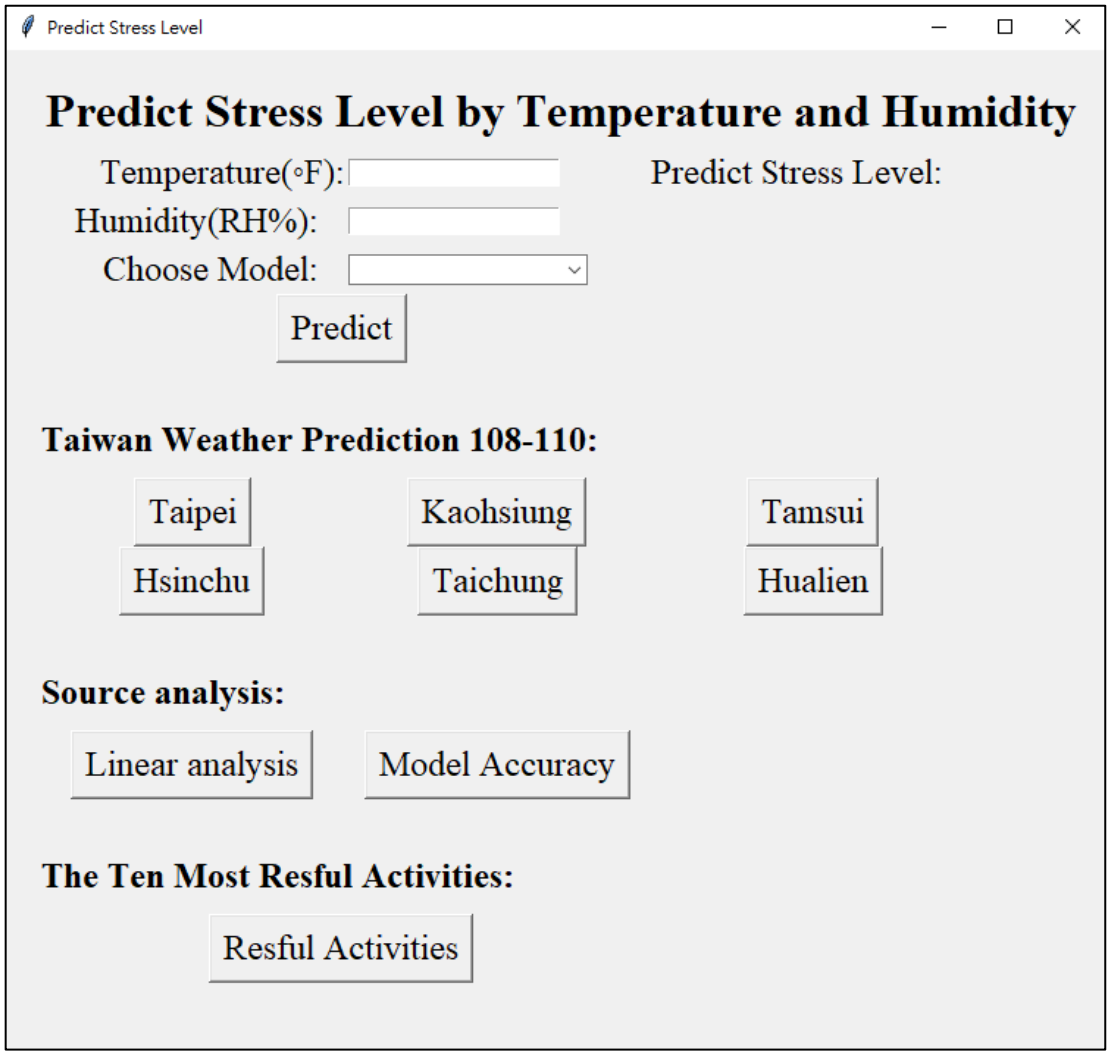
[13] https://www.kaggle.com/datasets/laavanya/stress-level-detection

Window Application Stress Level Predict (Result):

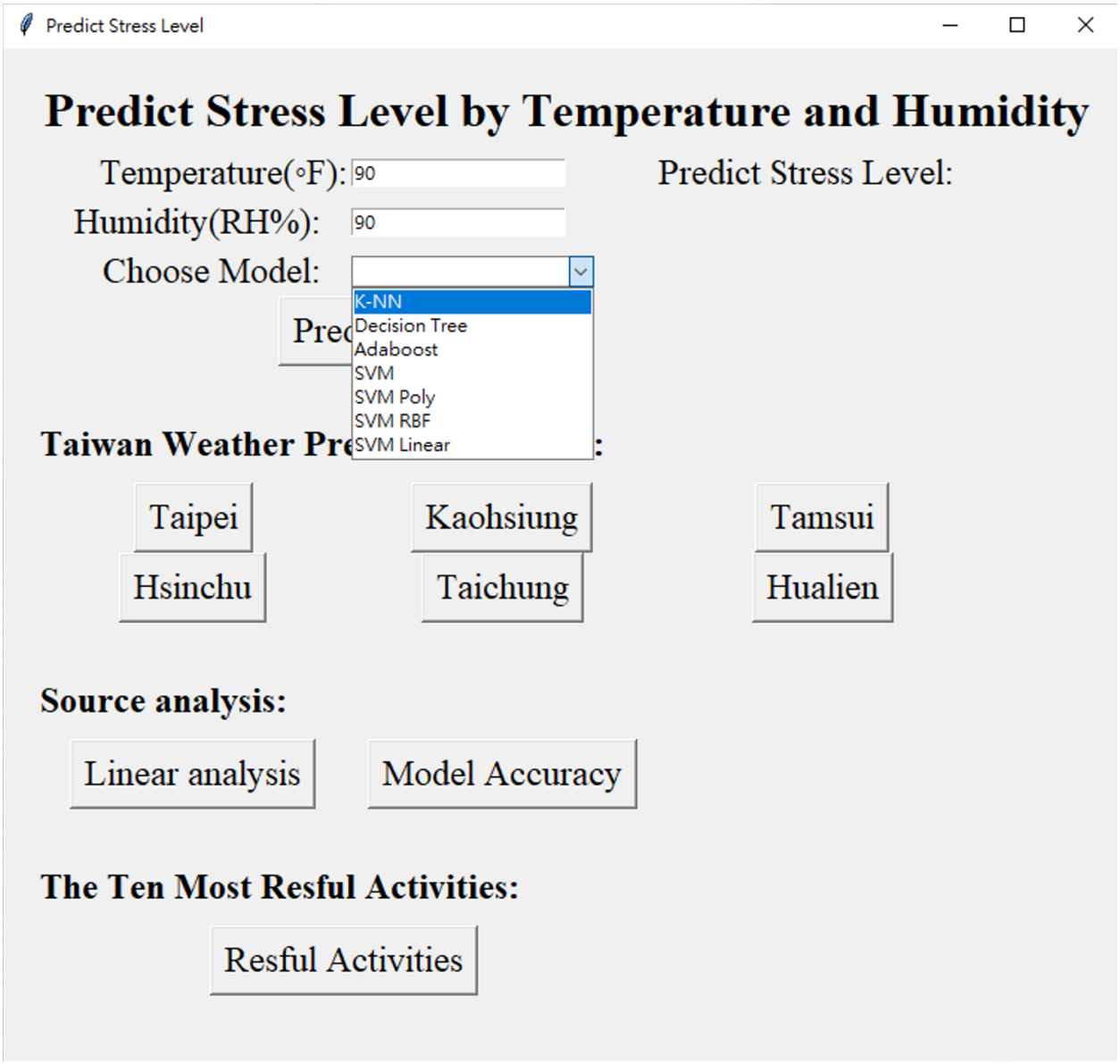
Our goal is to create a model which can predict the human stress by temperature and humidity. The operational window and steps as follow:

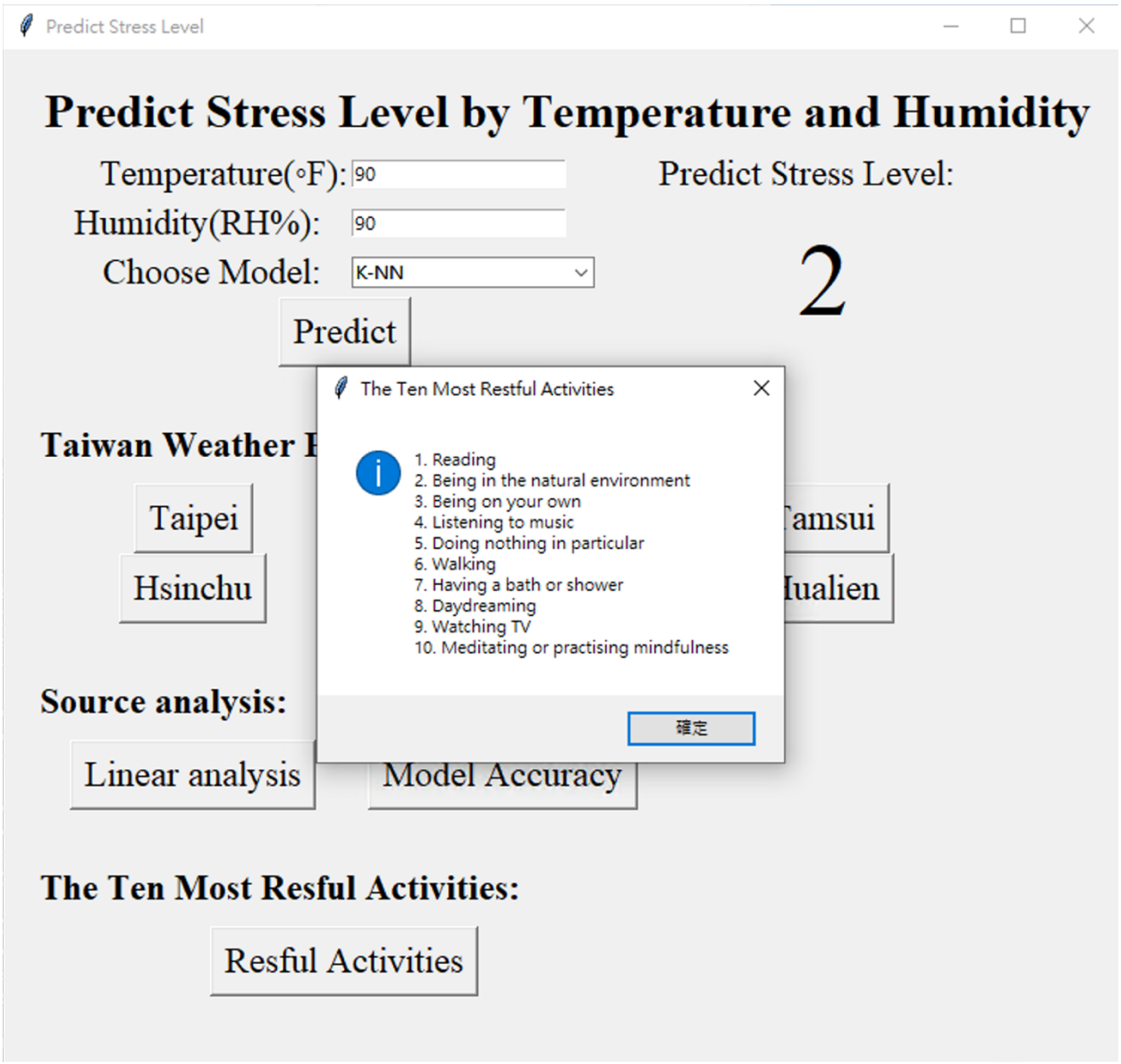
- input temperature (°F)
- input humidity (RH%)
- choose predict model
- click 'Predict' button
- get 'Predict Stress Level' (value: 0-2)
- if 'Predict Stress Level' is 2, the window will show 'The Ten Most Resful Activities'

We analyzed the stress levels from 108 to 110 represented by Taipei, Kaohsiung, Tamsui, Hsinchu, Taichung and Hualien, using data from Central Weather Bureau. Also, we provide source analysis including Linear analysis and Model Accuracy.

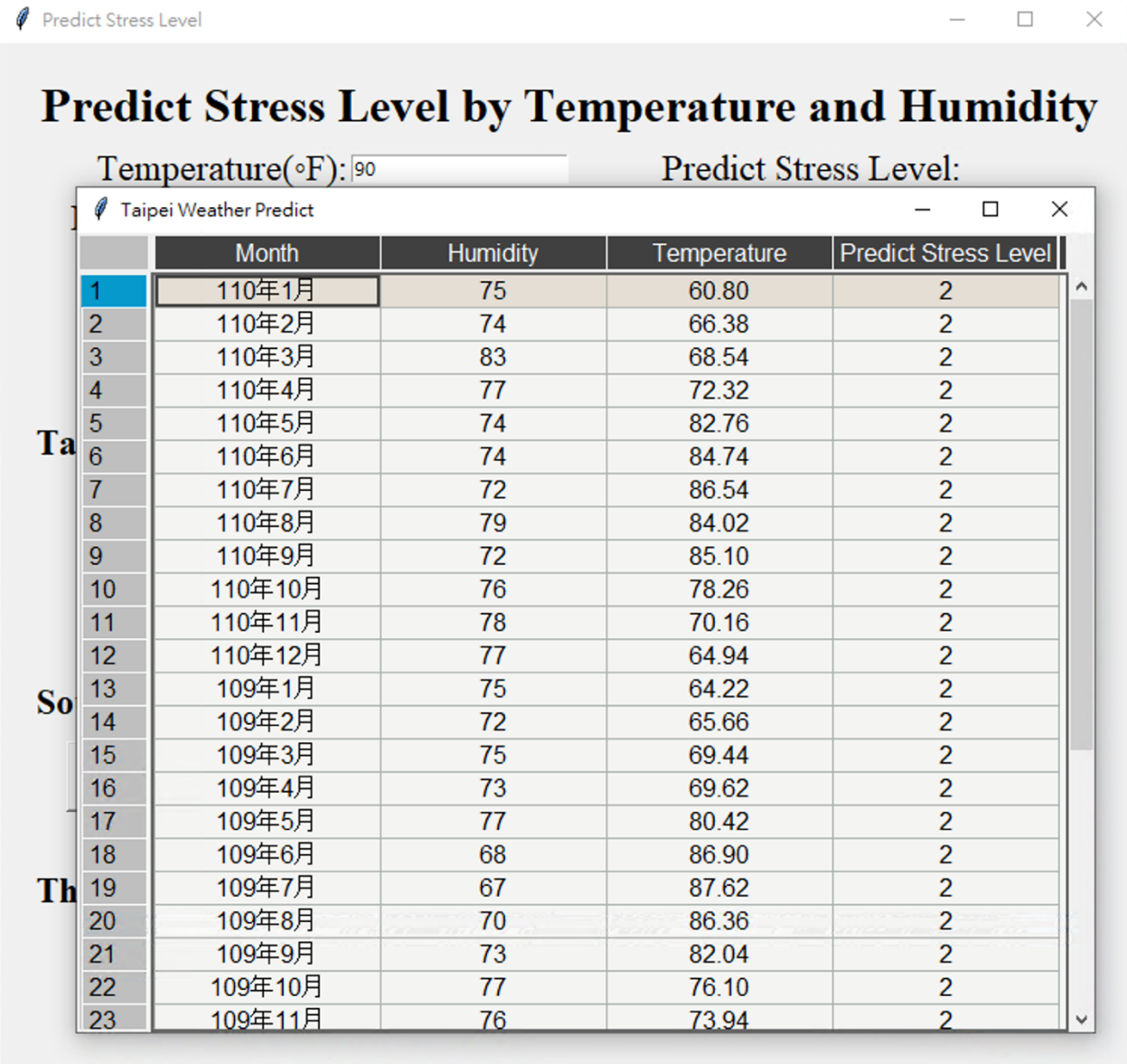


Ex: Enter humidity and temperature, and can choose algorithm models, if get stress level 2, it'll show recommend activities




Predict Stress Level
Predict Stress Level by Temperature and Humidity
Temperature(◦F): 90
Humidity(RH%): 90
Choose Model: K-NN
Predict
Predict Stress Level:
2
The Ten Most Restful Activities
1. Reading
2. Being in the natural environment
3. Being on your own
4. Listening to music
5. Doing nothing in particular
6. Walking
7. Having a bath or shower
8. Daydreaming
9. Watching TV
10. Meditating or practising mindfulness
確定
Taiwan Weather F
Taipei
Hsinchu
Tamsui
Hualien
Source analysis:
Linear analysis
Model Accuracy
The Ten Most Resful Activities:
Resful Activities

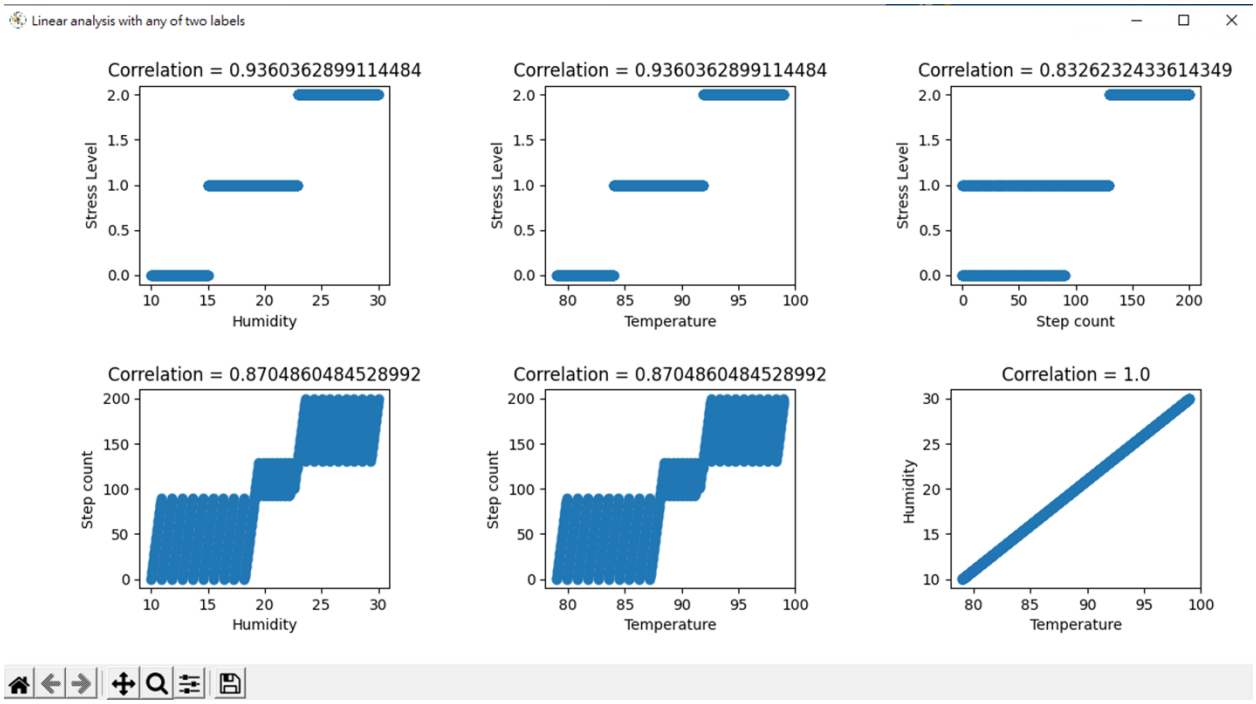
Ex: Click Taipei weather prediction

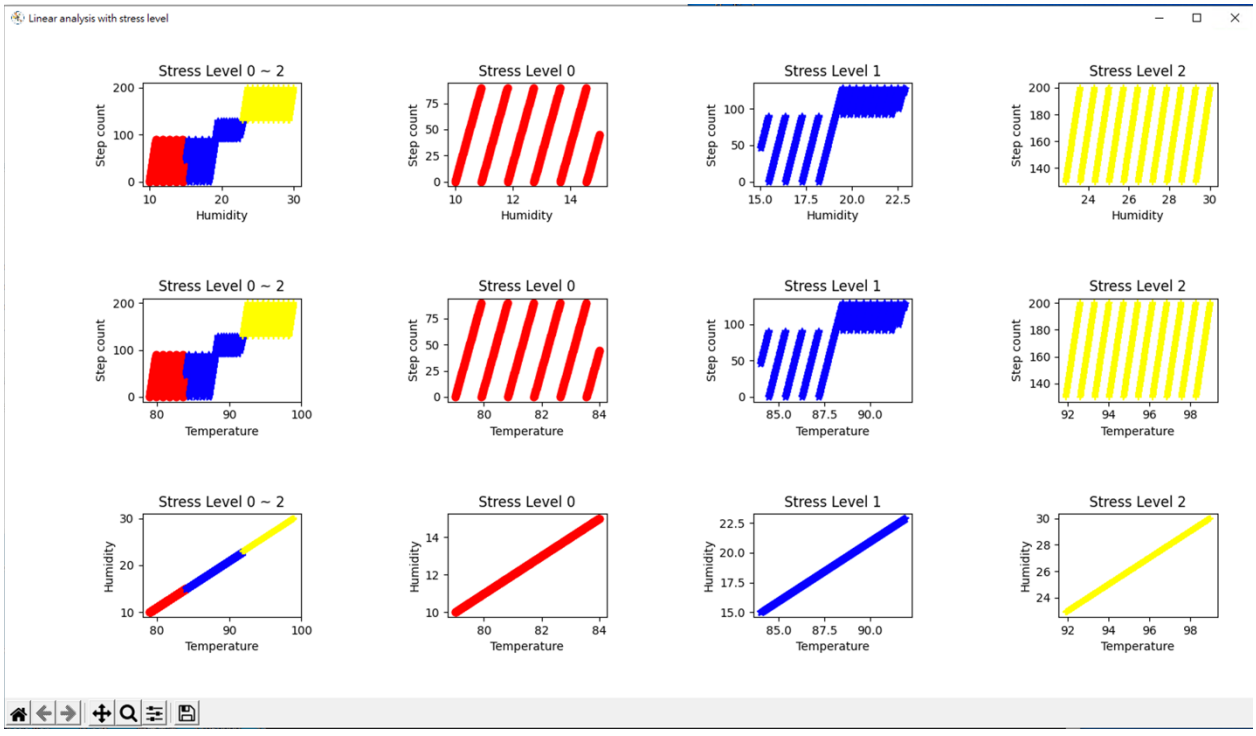


	Month	Humidity	Temperature	Predict Stress Level
1	110年1月	75	60.80	2
2	110年2月	74	66.38	2
3	110年3月	83	68.54	2
4	110年4月	77	72.32	2
5	110年5月	74	82.76	2
6	110年6月	74	84.74	2
7	110年7月	72	86.54	2
8	110年8月	79	84.02	2
9	110年9月	72	85.10	2
10	110年10月	76	78.26	2
11	110年11月	78	70.16	2
12	110年12月	77	64.94	2
13	109年1月	75	64.22	2
14	109年2月	72	65.66	2
15	109年3月	75	69.44	2
16	109年4月	73	69.62	2
17	109年5月	77	80.42	2
18	109年6月	68	86.90	2
19	109年7月	67	87.62	2
20	109年8月	70	86.36	2
21	109年9月	73	82.04	2
22	109年10月	77	76.10	2
23	109年11月	76	73.94	2

Ex: Click accuracy for each algorithm:

Ex: Click Linear analysis:





6. Conclusion

Our project used several kinds of algorithms to create the predict model and user can compare the accuracy. By using the predicting model, user can insert the temperature and humidity to predict the stress level. We also use Taiwan's temperature and humidity from year 108 to year 110 per month to do the prediction. It also shows the linear analysis on the tool. We found that high humidity and high temperature will get higher stress level. When user gets stress level 2(highest level), the tool will show ten of most restful activities to suggest user. After finished this project, we learned several algorithms which can train the model to classify the data. We hope that this can help people conscious of their mental status.

In the future, this predicting model can combine other data to do the analysis, e.g., when get higher predicted stress level, more people go travel at the same time. This can be analyzed by getting traffic data or how much tickets were sold at tourist spots. It can also be used to combine the IoT device to have warning message to user.

7. Others

We use mobbing to discuss, develop and write documentations together, and the total time efforts are 162 hours(81 hours/person). Detailed records are as follows:

- 3/10 19:00-23:30 discuss project plan
- 3/17 20:00-22:30 environment setting
- 3/27 22:00-00:00 plots data directly and analyze(linear analysis)
- 4/02 21:00-00:10 k-nn done
- 4/07 19:30-22:30 midterm report
- 4/13 23:00-02:00 read papers
- 4/14 11:00-13:30 write introduction
- 4/14 18:00-19:00 draw timeline
- 4/14 21:30-00:00 midterm report and presentation practice
- 4/29 10:00-11:50 correlation coefficient and part of SVM
- 4/29 19:00-01:10 SVM, Decision Tree, Transfer all code to python
- 4/30 21:30-23:30 modified the code and discuss what to do next
- 5/19 21:00-23:30 GUI_try to present title, label and entry
- 5/27 10:00-12:30 GUI_predict part done
- 6/02 21:00-00:30 GUI_taiwan weather done

- 6/08 13:00-15:30 almost finish GUI

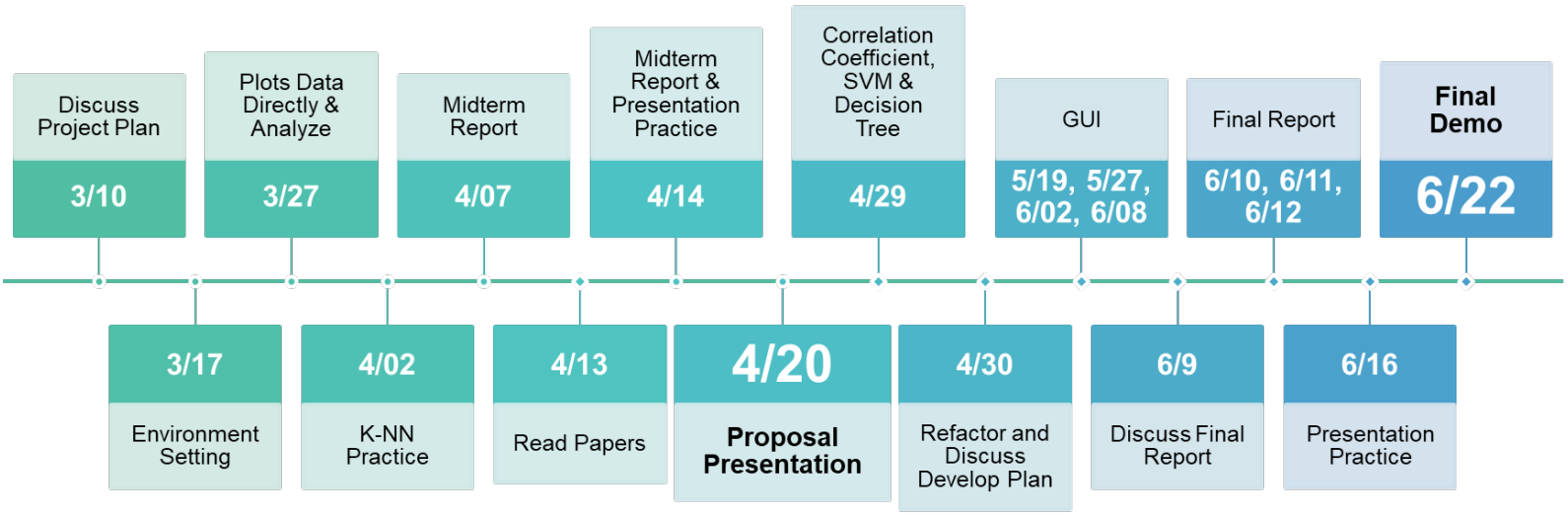
- 6/09 21:00-22:10 finish GUI and discuss document

- 6/10 16:00-4:00 document Abstract, Introduction, Problem Statement, Conclusion

- 6/11 20:00-4:00 document Proposed Models

- 6/12 12:50-23:50 document appendix, reference, literature review and related works, experiments, demo video, README file

- 6/16 21:30-01:00 PPT & presentation practice



Reference

[1] Shukla, J. (2013). Extreme weather events and mental health: Tackling the psychosocial challenge. International Scholarly Research Notices, 2013.

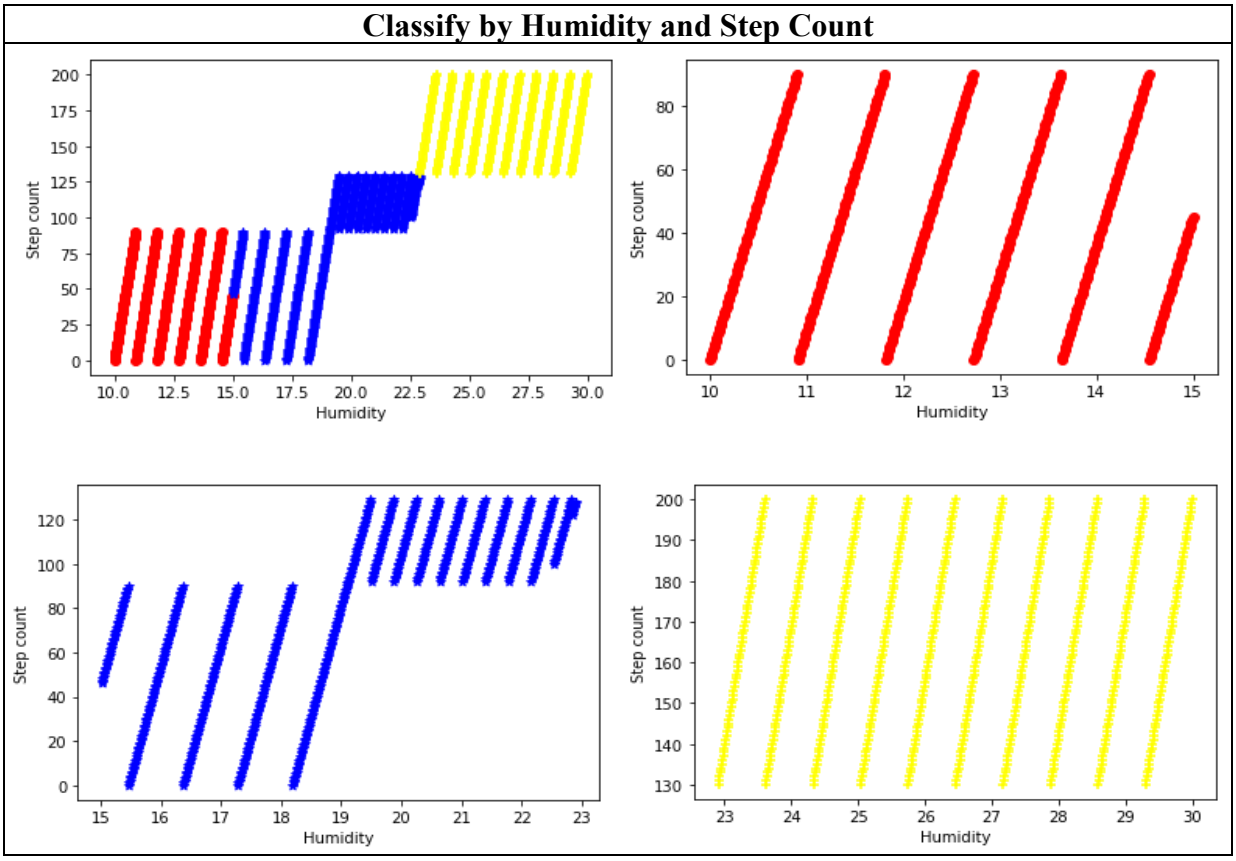
[2] Rachakonda, L., Sundaravadivel, P., Mohanty, S. P., Kougianos, E., & Ganapathiraju, M. (2018, December). A smart sensor in the IoMT for stress level detection. In 2018 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS) (pp. 141-145). IEEE.

[3] Rachakonda, L., Mohanty, S. P., Kougianos, E., & Sundaravadivel, P. (2019). Stress-Lysis: A DNN-integrated edge device for stress level detection in the IoMT. IEEE Transactions on Consumer Electronics, 65(4), 474-483.

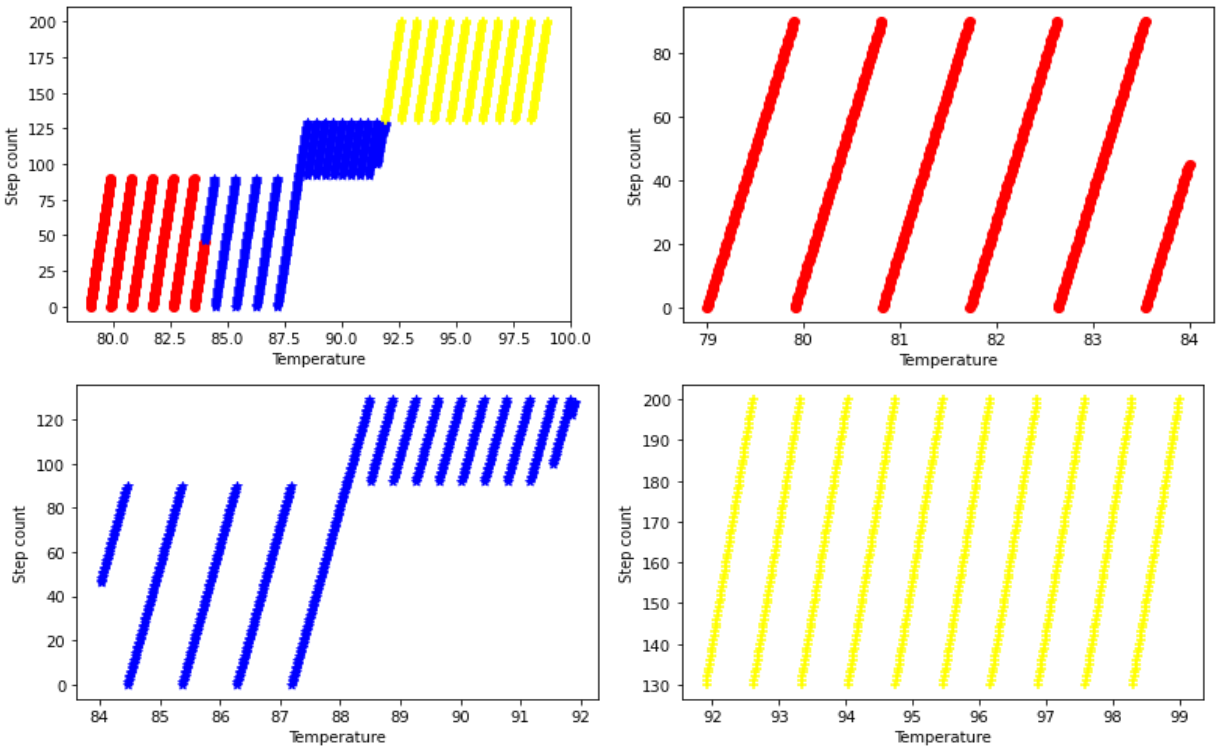
Appendix

Linear analysis:

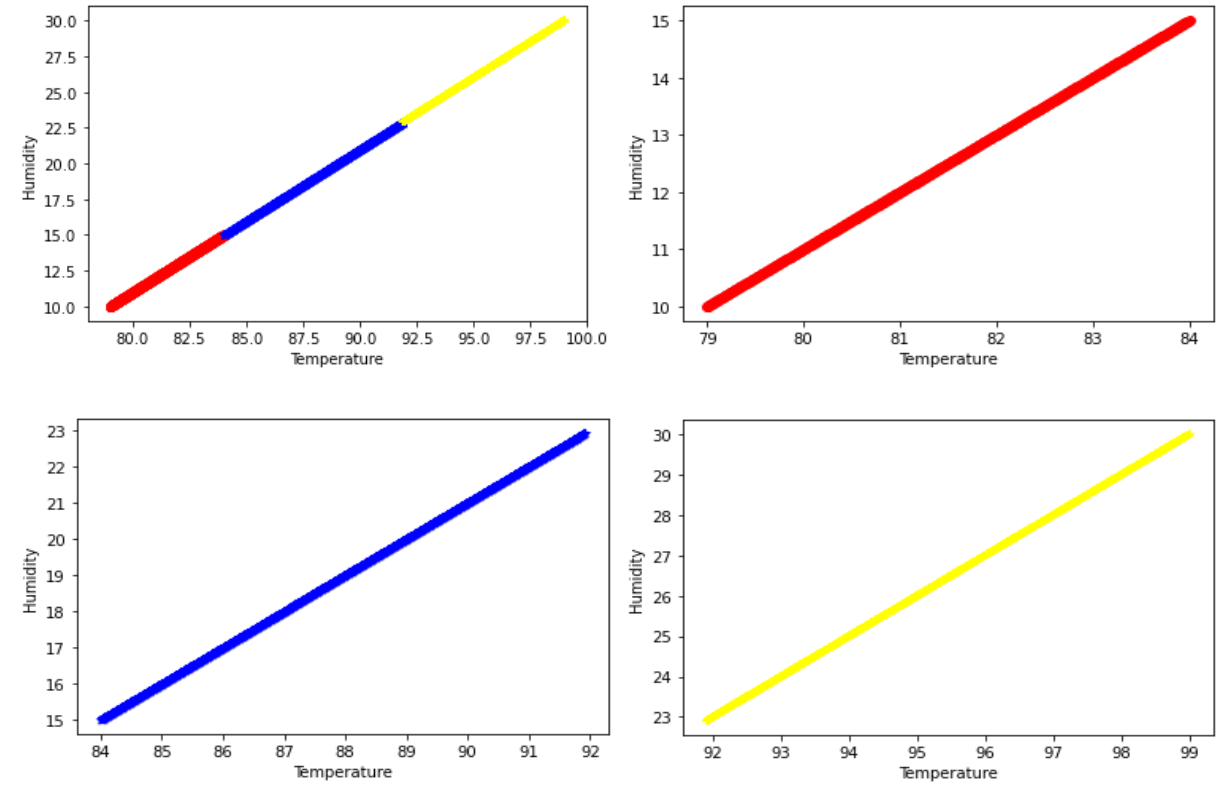
a. by stress level (red: stress 0, blue: stress 1, yellow: stress 2):



Classify by Temperature and Step Count

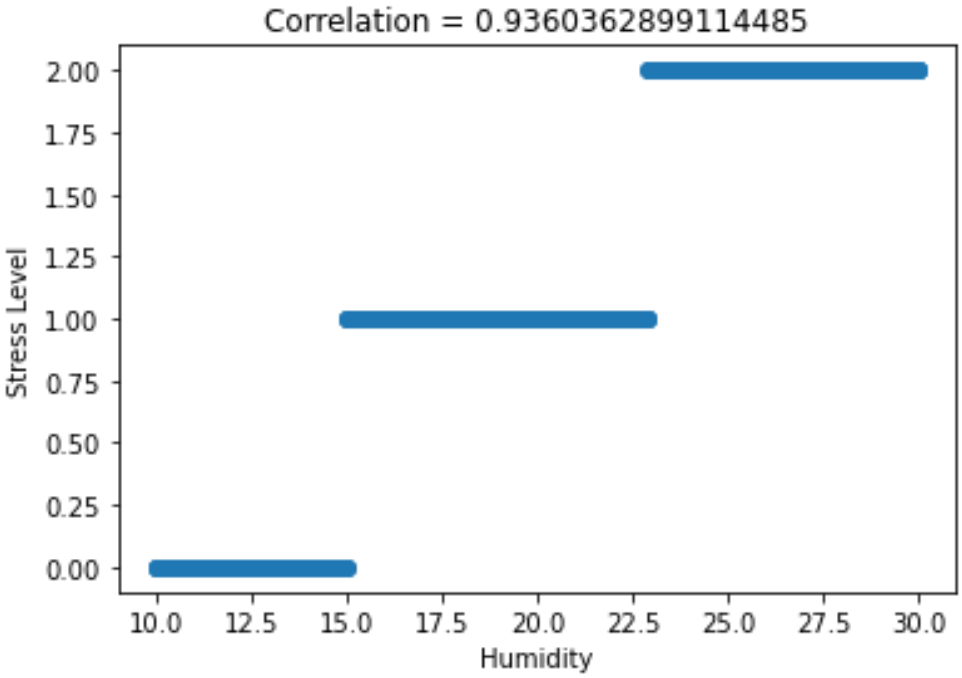


Classify by Temperature and Humidity

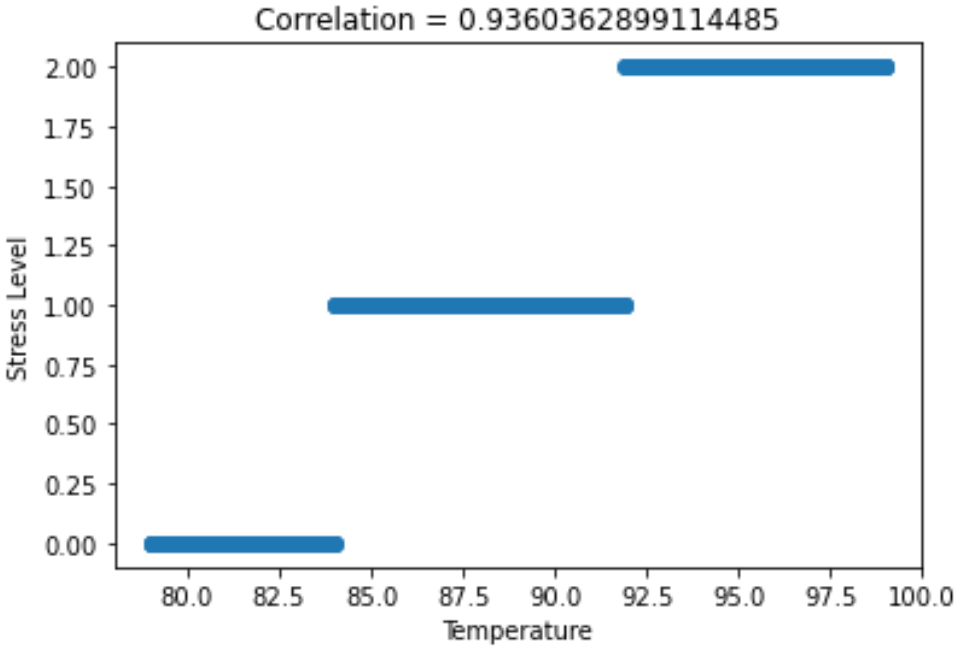


b. any of two labels:

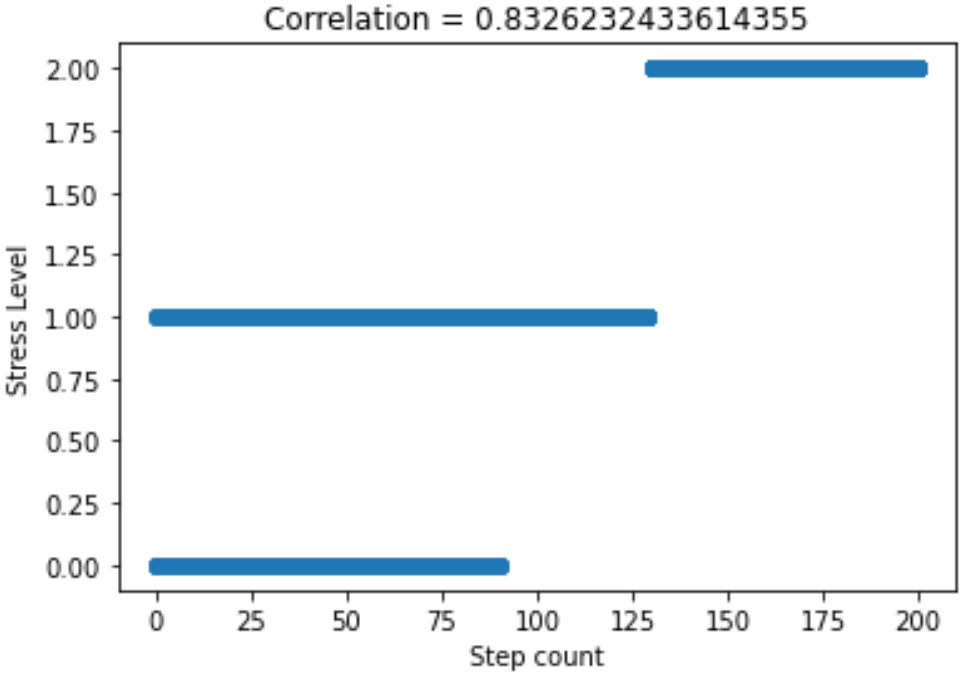
Correlation coefficient between humidity and stress level is 0.93



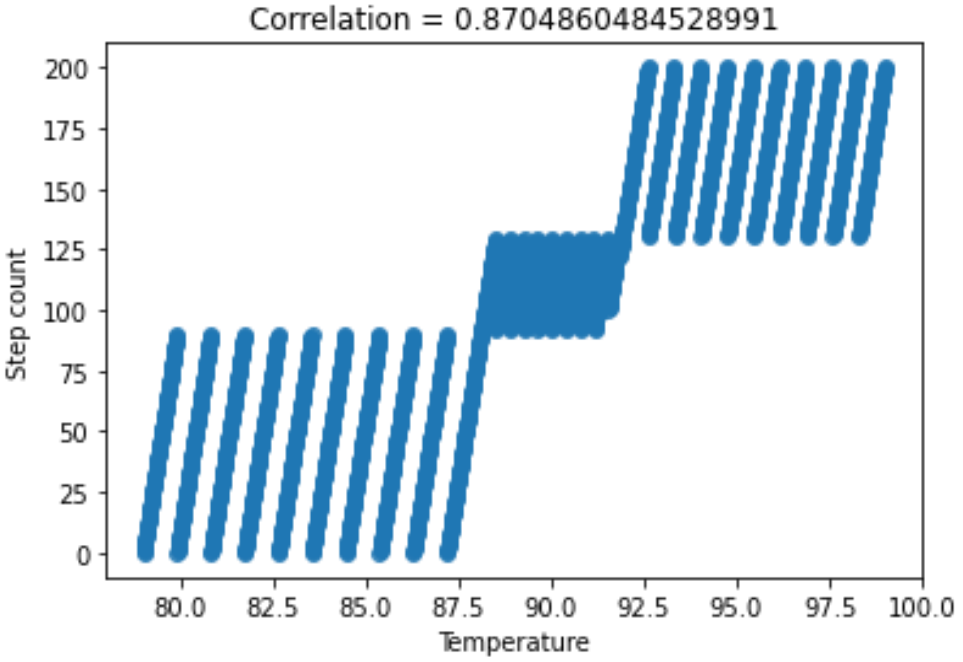
Correlation coefficient between temperature and stress level is 0.93



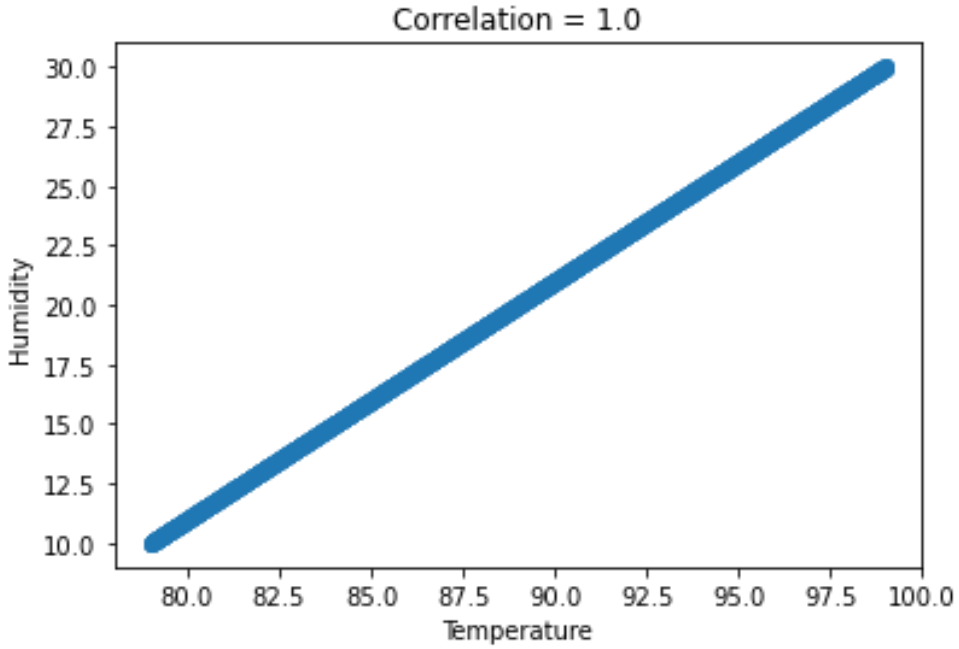
Correlation coefficient between step count and stress level is 0.83



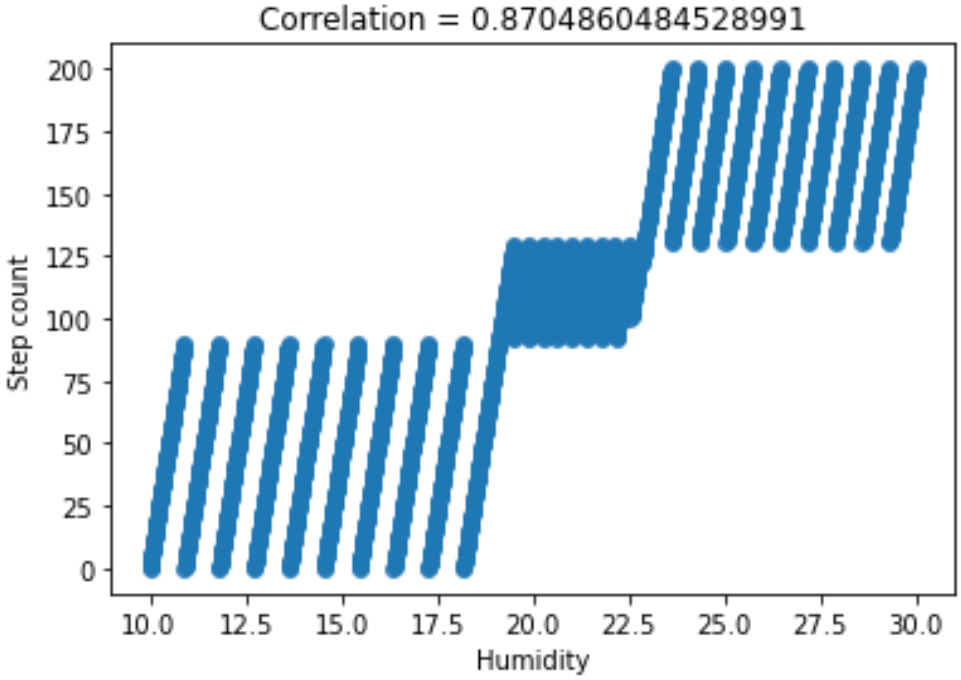
Correlation coefficient between temperature and step count is 0.87



Correlation coefficient between temperature and humidity is 1.0



Correlation coefficient between humidity and step count is 0.87



Python algorithm code:

K-NN

```
hum_temp_data = lysis_data[['Humidity', 'Temperature']]
str_lev_data = lysis_data['Stress Level']

train_data , test_data , train_label , test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
train_data , ver_data , train_label , ver_label = train_test_split(train_data,
train_label, test_size=0.25)

model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(train_data, train_label)

# Predict Output
predicted = model.predict(ver_data) # 0:Overcast, 2:Mild

print("accuracy: ", accuracy_score(ver_label, predicted))
```

accuracy: 0.9975

```
predicted = model.predict(test_data)
print("accuracy: ", accuracy_score(test_label, predicted))
```

accuracy: 1.0

AdaBoost Decision Tree and AdaBoost Classifier

```
hum_temp_data = lysis_data[['Humidity', 'Temperature']]
str_lev_data = lysis_data['Stress Level']

train_data , test_data , train_label , test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
train_data , ver_data , train_label , ver_label = train_test_split(train_data,
train_label, test_size=0.25)

model =  AdaBoostClassifier(DecisionTreeClassifier(max_depth=2), n_estimators=50)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 1.0

Test Accuracy: 0.9975062344139651

```
model =  AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 1.0

Test Accuracy: 0.9975062344139651

```
model = AdaBoostClassifier(n_estimators=50, random_state=0)
model.fit(train_data, train_label)

pred_ver = model.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = model.predict(test_data)
```

```
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 1.0

Test Accuracy: 0.9975062344139651

SVM

```
hum_temp_data = lysis_data[['Humidity', 'Temperature']]
str_lev_data = lysis_data['Stress Level']

train_data , test_data , train_label , test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
train_data , ver_data , train_label , ver_label = train_test_split(train_data,
train_label, test_size=0.25)

C = 2 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=C).fit(train_data, train_label)

pred_ver = svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 0.9975

Test Accuracy: 0.9925187032418953

```
C = 2 # SVM regularization parameter

rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(train_data, train_label)

pred_ver = rbf_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = rbf_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 0.9975

Test Accuracy: 0.9950124688279302

```
C = 2 # SVM regularization parameter

poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(train_data, train_label)

pred_ver = poly_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = poly_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 1.0

Test Accuracy: 0.9950124688279302

```
C = 2 # SVM regularization parameter

lin_svc = svm.LinearSVC(C=C, dual=False).fit(train_data, train_label)

pred_ver = lin_svc.predict(ver_data)
acc_train = accuracy_score(ver_label, pred_ver)

pred_test = lin_svc.predict(test_data)
acc_test = accuracy_score(test_label, pred_test)

print("Verification Accuracy: {} \nTest Accuracy: {}".format(acc_train,
acc_test))
```

Verification Accuracy: 0.9975

Test Accuracy: 0.9975062344139651

Stress Level Predict Model Code: GUI

```
def show_model_accuracy():
    classifier_model = ClassifierModel()

    acc_result_window = tk.Tk()
    acc_result_window.title('Model Accuracy')

    result_list =
[acc_result.knn(classifier_model),  acc_result.decision_tree(classifier_model),
acc_result.adaboost(classifier_model), acc_result.svm(classifier_model),
acc_result.svm_rbf(classifier_model), acc_result.svm_poly(classifier_model),
acc_result.svm_linear(classifier_model)]

    acc_string = ""

    for acc in result_list:
        acc_string += acc
        acc_string += "\n\n"

    size = len(acc_string)
    acc_string = acc_string[:size-2]

    acc_result_label = tk.Label(acc_result_window, text = acc_string,
font=("Times New Roman", 18),  justify=tk.LEFT)
    acc_result_label.grid(sticky = tk.W)


def show_linear_analysis():
    linear_analysis = LinearAnaiysis()

    linear_analysis.linear_with_correlation_all()
    linear_analysis.linear_show_with_stress_level_all()


def taipei_pred():

    mon_data = ["110 年 1 月", "110 年 2 月","110 年 3 月","110 年 4 月","110 年 5 月
","110 年 6 月","110 年 7 月","110 年 8 月","110 年 9 月","110 年 10 月","110 年 11 月","110
年 12 月","109 年 1 月", "109 年 2 月","109 年 3 月","109 年 4 月","109 年 5 月","109 年 6 月
","109 年 7 月","109 年 8 月","109 年 9 月","109 年 10 月","109 年 11 月","109 年 12 月",
"108 年 1 月", "108 年 2 月","108 年 3 月","108 年 4 月","108 年 5 月","108 年 6 月","108 年
7 月","108 年 8 月","108 年 9 月","108 年 10 月","108 年 11 月","108 年 12 月"]
```

```
    humi_data =
[75,74,83,77,74,74,72,79,72,76,78,77,75,72,75,73,77,68,67,70,73,77,76,88,74,79,75
,76,77,77,73,72,79,74,75,77]
    temp_data =
[60.8,66.38,68.54,72.32,82.76,84.74,86.54,84.02,85.1,78.26,70.16,64.94,64.22,65.6
6,69.44,69.62,80.42,86.9,87.62,86.36,82.04,76.1,73.94,64.58,65.3,65.84,67.64,75.5
6,77,83.3,86.54,86.9,81.14,77.54,71.6,66.38]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()

    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)

    weather_result = tk.Tk()
    weather_result.title('Taipei Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)

    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

    # weather_result.mainloop()

def kaohsiung_pred():
    mon_data = ["110年1月", "110年2月","110年3月","110年4月","110年5月
","110年6月","110年7月","110年8月","110年9月","110年10月","110年11月","110
年12月","109年1月", "109年2月","109年3月","109年4月","109年5月","109年6月
","109年7月","109年8月","109年9月","109年10月","109年11月","109年12月",
"108年1月", "108年2月","108年3月","108年4月","108年5月","108年6月","108年
7月","108年8月","108年9月","108年10月","108年11月","108年12月"]

    humi_data =
[69,72,73,75,76,84,81,84,80,77,71,71,70,69,71,69,79,75,76,82,75,74,74,71,71,71,73
,74,79,79,81,84,76,72,70,71]
    temp_data =
[64.94,69.44,74.48,77.72,84.38,82.76,84.38,83.12,84.38,81.68,76.28,70.16,69.62,71
```

```
.24,76.46,77,83.48,86.54,86.9,84.38,85.46,82.22,78.44,72.14,71.42,74.84,75.38,80.
42,81.5,85.1,84.74,83.84,83.66,81.86,77.18,71.24]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()

    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)

    weather_result = tk.Tk()
    weather_result.title('Kaohsiung Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)

    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

def tamsui_pred():
    mon_data = ["110 年 1 月", "110 年 2 月","110 年 3 月","110 年 4 月","110 年 5 月
","110 年 6 月","110 年 7 月","110 年 8 月","110 年 9 月","110 年 10 月","110 年 11 月","110
年 12 月","109 年 1 月", "109 年 2 月","109 年 3 月","109 年 4 月","109 年 5 月","109 年 6 月
","109 年 7 月","109 年 8 月","109 年 9 月","109 年 10 月","109 年 11 月","109 年 12 月",
"108 年 1 月", "108 年 2 月","108 年 3 月","108 年 4 月","108 年 5 月","108 年 6 月","108 年
7 月","108 年 8 月","108 年 9 月","108 年 10 月","108 年 11 月","108 年 12 月"]

    humi_data =
[79,80,88,78,82,82,79,84,77,77,80,82,82,85,86,79,84,73,62,72,78,76,76,88,83,87,81
,84,82,85,79,77,83,84,80,82]
    temp_data =
[59,64.04,65.3,70.7,80.6,83.12,85.1,83.12,83.84,77.36,68.9,63.14,62.24,63.14,67.2
8,68.18,78.62,84.38,85.82,84.56,79.88,75.2,72.68,63.32,63.68,63.86,66.38,73.58,75
.56,81.32,85.28,84.92,79.34,75.74,70.34,64.4]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()
```

```
    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)

    weather_result = tk.Tk()
    weather_result.title('Tamsui Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)

    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

def hsinchu_pred():
    mon_data = ["110年1月", "110年2月","110年3月","110年4月","110年5月
","110年6月","110年7月","110年8月","110年9月","110年10月","110年11月","110
年12月","109年1月", "109年2月","109年3月","109年4月","109年5月","109年6月
","109年7月","109年8月","109年9月","109年10月","109年11月","109年12月",
"108年1月", "108年2月","108年3月","108年4月","108年5月","108年6月","108年
7月","108年8月","108年9月","108年10月","108年11月","108年12月"]

    humi_data =
[74,76,84,76,78,77,75,79,73,71,76,75,78,74,79,74,78,70,65,72,73,71,73,82,76,80,78
,77,77,74,68,74,71,71,70,75]
    temp_data =
[59.54,64.22,66.56,71.78,81.5,83.66,85.64,83.48,84.74,78.8,69.98,64.22,62.06,63.6
8,68,68.9,79.88,85.46,87.44,85.28,81.5,76.82,73.94,64.4,63.86,64.4,66.74,74.48,76
.46,82.94,86.36,85.1,80.96,77.36,71.24,64.94]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()

    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)
```

```
    weather_result = tk.Tk()
    weather_result.title('Hsinchu Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)

    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

def taichung_pred():
    mon_data = ["110 年 1 月", "110 年 2 月","110 年 3 月","110 年 4 月","110 年 5 月
","110 年 6 月","110 年 7 月","110 年 8 月","110 年 9 月","110 年 10 月","110 年 11 月","110
年 12 月","109 年 1 月", "109 年 2 月","109 年 3 月","109 年 4 月","109 年 5 月","109 年 6 月
","109 年 7 月","109 年 8 月","109 年 9 月","109 年 10 月","109 年 11 月","109 年 12 月",
"108 年 1 月", "108 年 2 月","108 年 3 月","108 年 4 月","108 年 5 月","108 年 6 月","108 年
7 月","108 年 8 月","108 年 9 月","108 年 10 月","108 年 11 月","108 年 12 月"]

    humi_data =
[71,70,73,70,70,81,77,81,74,71,72,71,76,72,70,70,75,70,74,75,71,69,72,77,75,75,79
,79,82,79,79,86,78,71,69,75]
    temp_data =
[60.8,66.2,70.7,75.02,83.84,82.04,84.2,82.04,84.2,80.06,72.32,65.84,64.4,66.2,72.
14,71.96,81.5,84.38,85.28,83.3,82.58,78.98,75.02,67.1,66.74,68.9,69.98,76.46,77.7
2,82.76,84.92,83.12,81.86,78.98,73.4,66.74]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()

    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)

    weather_result = tk.Tk()
    weather_result.title('Taichung Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)
```

```
    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

def hualien_pred():
    mon_data = ["110年1月", "110年2月","110年3月","110年4月","110年5月
","110年6月","110年7月","110年8月","110年9月","110年10月","110年11月","110
年12月","109年1月", "109年2月","109年3月","109年4月","109年5月","109年6月
","109年7月","109年8月","109年9月","109年10月","109年11月","109年12月",
"108年1月", "108年2月","108年3月","108年4月","108年5月","108年6月","108年
7月","108年8月","108年9月","108年10月","108年11月","108年12月"]

    humi_data =
[79,78,84,82,84,84,77,82,80,80,81,75,77,74,80,79,83,75,76,75,76,74,79,81,77,80,75
,76,80,81,77,78,75,73,71,75]
    temp_data =
[61.88,67.1,70.7,72.86,80.42,82.4,84.56,82.76,82.4,77.36,70.88,67.1,67.1,67.28,70
.52,70.7,78.98,84.74,85.1,84.38,81.32,77.54,74.3,68.36,68,69.8,69.62,76.1,75.92,8
1.86,84.74,84.02,81.32,77.54,73.22,68.36]

    data = {'Humidity': humi_data, 'Temperature': temp_data}
    test_data = pd.DataFrame(data)

    class_model = ClassifierModel()

    predict_val = class_model.knn_prediction(test_data)

    show_data = {'Month': mon_data, 'Humidity': humi_data, 'Temperature':
temp_data, 'Predict Stress Level':predict_val}
    show_data_frame = pd.DataFrame(show_data)

    weather_result = tk.Tk()
    weather_result.title('Hualien Weather Predict')

    frame = tk.Frame(weather_result)
    frame.pack(fill='both', expand=True)

    pt = Table(frame, dataframe=show_data_frame, width=605, height=500,
cellwidth=150, align='center')

    pt.show()

def show_resful_activities():
```

```
    tk.messagebox.showinfo('The Ten Most Restful Activities', '1. Reading\n2.
Being in the natural environment\n3. Being on your own\n4. Listening to music\n5.
Doing nothing in particular\n6. Walking\n7. Having a bath or shower\n8.
Daydreaming\n9. Watching TV\n10. Meditating or practising mindfulness')

def predict_result():
    humi = None
    temp = None
    try:
        humi = float(entry_humi.get())
    except:
        tk.messagebox.showerror(title=None, message="Invlid Humidity Data.")
        return 0

    try:
        temp = float(entry_temp.get())
    except:
        tk.messagebox.showerror(title=None, message="Invlid Temperature Data.")
        return 0

    data = {'Humidity': [humi], 'Temperature': [temp]}

    test_data = pd.DataFrame(data)
    model = combo_mod.get()
    class_model = ClassifierModel()
    predict_val = -1

    try:
        if (model == "K-NN"):
            predict_val = class_model.knn_prediction(test_data)[0]
        elif (model == "Decision Tree"):
            predict_val = class_model.decision_tree_prediction(test_data)[0]
        elif (model == "Adaboost"):
            predict_val = class_model.adaboost_prediction(test_data)[0]
        elif (model == "SVM"):
            predict_val = class_model.svc_prediction(test_data)[0]
        elif (model == "SVM Poly"):
            predict_val = class_model.poly_svc_prediction(test_data)[0]
        elif (model == "SVM RBF"):
            predict_val = class_model.rbf_svc_prediction(test_data)[0]
        elif (model == "SVM Linear"):
            predict_val = class_model.lin_svc_prediction(test_data)[0]
        else:
            tk.messagebox.showerror(title=None, message="Please choose a predict
model.")
```

```
            return 0
    except:
        tk.messagebox.showerror(title=None, message="Please enter valid data")
        return 0

    res_val.config(text = str(predict_val))

    if predict_val == 2:
        show_resful_activities()


if __name__ == "__main__":

    # 建立主視窗 Frame
    window = tk.Tk()
    window.geometry('750x680')

    # 設定視窗標題
    window.title('Predict Stress Level')

    # 標示文字
    title_label_font_style = tkFont.Font(family="Times New Roman", size=24,
weight="bold")
    title = tk.Label(window, text = 'Predict Stress Level by Temperature and
Humidity',
                     font = title_label_font_style)
    title.grid(row = 1, column = 0, columnspan = 7, padx = (25, 0), pady = (20,
5))

    # create a label widget
    label_temp = tk.Label(window, text = "Temperature(◦F):", font=("Times New
Roman", 18))
    label_humi = tk.Label(window, text = "Humidity(RH%):   ", font=("Times New
Roman", 18))
    label_mod = tk.Label(window, text = "Choose Model:   ", font=("Times New
Roman", 18))


    # 建立按鈕
    predict_button = tk.Button(window,   # 按鈕所在視窗
                    text = 'Predict',  # 顯示文字
                    command = predict_result, # 按下按鈕所執行的函數
                    font=("Times New Roman", 18))
```

```
    # rows and columns as specified
    label_temp.grid(row = 2, column = 1, sticky=tk.E)
    label_humi.grid(row = 3, column = 1, sticky=tk.E)
    label_mod.grid(row = 4, column = 1, sticky=tk.E)
    predict_button.grid(row = 5, column = 1, columnspan = 2)

    # entry widgets, used to take entry from user
    entry_temp = tk.Entry(window, width = 20)
    entry_humi = tk.Entry(window, width = 20)

    combo_mod = ttk.Combobox(window,
                             values=[
                                     "K-NN",
                                     "Decision Tree",
                                     "Adaboost",
                                     "SVM",
                                     "SVM Poly",
                                     "SVM RBF",
                                     "SVM Linear"])

    # arrange entry widgets
    entry_temp.grid(row = 2, column = 2, sticky=tk.W)
    entry_humi.grid(row = 3, column = 2, sticky=tk.W)
    combo_mod.grid(row = 4, column = 2, sticky=tk.W)

    label_res = tk.Label(window, text = "Predict Stress Level:", font=("Times New
Roman", 18))
    res_val = tk.Label(window, text = "", font=("Times New Roman", 52))

    label_res.grid(row = 2, column = 3, sticky=tk.W)
    res_val.grid(row = 3, column = 3, rowspan = 3)

    tk.Label(window, text="\n").grid(row = 6, column = 1, columnspan = 3)

    label_taiwan_pred = tk.Label(window, text = "Taiwan Weather Prediction 108-
110:",
                                 font=("Times New Roman", 18, "bold"))
    taipei_predict_button = tk.Button(window,
                    text = 'Taipei',
                    command = taipei_pred,
                    font=("Times New Roman", 18))
    kaohsiung_predict_button = tk.Button(window,
                    text = 'Kaohsiung',
                    command = kaohsiung_pred,
                    font=("Times New Roman", 18))
```

```
    tamsui_predict_button = tk.Button(window,
                    text = 'Tamsui',
                    command = tamsui_pred,
                    font=("Times New Roman", 18))
    hsinchu_predict_button = tk.Button(window,
                    text = 'Hsinchu',
                    command = hsinchu_pred,
                    font=("Times New Roman", 18))
    taichung_predict_button = tk.Button(window,
                    text = 'Taichung',
                    command = taichung_pred,
                    font=("Times New Roman", 18))
    hualien_predict_button = tk.Button(window,
                    text = 'Hualien',
                    command = hualien_pred,
                    font=("Times New Roman", 18))

    label_taiwan_pred.grid(row = 7, column = 1, columnspan = 3, sticky=tk.W, padx
= (0, 0), pady = (0, 10))
    taipei_predict_button.grid(row = 8, column = 1)
    kaohsiung_predict_button.grid(row = 8, column = 2)
    tamsui_predict_button.grid(row = 8, column = 3)
    hsinchu_predict_button.grid(row = 9, column = 1)
    taichung_predict_button.grid(row = 9, column = 2)
    hualien_predict_button.grid(row = 9, column = 3)

    tk.Label(window, text="\n").grid(row = 10, column = 1, columnspan = 3)

    label_source_analysis = tk.Label(window, text = "Source analysis: ",
                                font=("Times New Roman", 18, "bold"))

    linear_analysis_button = tk.Button(window,
                    text = 'Linear analysis',
                    command = show_linear_analysis,
                    font=("Times New Roman", 18))

    model_accuaracy_button = tk.Button(window,
                    text = 'Model Accuracy',
                    command = show_model_accuracy,
                    font=("Times New Roman", 18))


    label_source_analysis.grid(row = 11, column = 1, sticky=tk.W, padx = (0, 0),
pady = (0, 10))
    linear_analysis_button.grid(row = 12, column = 1)
```

```
    model_accuaracy_button.grid(row = 12, column = 2)

    tk.Label(window, text="\n").grid(row = 13, column = 1, columnspan = 3)

    label_reful_activities = tk.Label(window, text = "The Ten Most Resful
Activities: ",
                                font=("Times New Roman", 18, "bold"))

    show_resful_activities_button = tk.Button(window,
                    text = 'Resful Activities',
                    command = show_resful_activities,
                    font=("Times New Roman", 18))

    label_reful_activities.grid(row = 14, column = 1, columnspan = 3,
sticky=tk.W, padx = (0, 0), pady = (0, 10))
    show_resful_activities_button.grid(row = 15, column = 1, columnspan = 2)

    window.mainloop()
```

Stress Level Predict Model Code: classifier model

```
class ClassifierModel:
    def __init__(self):
        lysis_data=pd.read_csv("../Stress-Lysis.csv")
        hum_temp_data = lysis_data[['Humidity', 'Temperature']]
        str_lev_data = lysis_data['Stress Level']

        train_data , self.test_data , train_label , self.test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
        self.train_data , self.ver_data , self.train_label , self.ver_label =
train_test_split(train_data, train_label, test_size=0.25)

        self.knn_model = KNeighborsClassifier(n_neighbors=3).fit(self.train_data,
self.train_label)

        self.svc_model = svm.SVC(kernel='linear', C=2).fit(self.train_data,
self.train_label)
        self.rbf_svc_model = svm.SVC(kernel='rbf', gamma=0.7,
C=2).fit(self.train_data, self.train_label)
        self.poly_svc_model = svm.SVC(kernel='poly', degree=3,
C=2).fit(self.train_data, self.train_label)
        self.lin_svc_model = svm.LinearSVC(C=2, dual=False).fit(self.train_data,
self.train_label)

        self.decision_tree_model =
AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),
n_estimators=50).fit(self.train_data, self.train_label)
        self.adaboost_model = AdaBoostClassifier(n_estimators=50,
random_state=0).fit(self.train_data, self.train_label)


    def svc_prediction(self, predict_data):
        return self.svc_model.predict(predict_data)

    def rbf_svc_prediction(self, predict_data):
        return self.rbf_svc_model.predict(predict_data)

    def poly_svc_prediction(self, predict_data):
        return self.poly_svc_model.predict(predict_data)

    def lin_svc_prediction(self, predict_data):
        return self.lin_svc_model.predict(predict_data)

    def knn_prediction(self, predict_data):
```

```
        return self.knn_model.predict(predict_data)

    def decision_tree_prediction(self, predict_data):
        return self.decision_tree_model.predict(predict_data)

    def adaboost_prediction(self, predict_data):
        return self.adaboost_model.predict(predict_data)
```

Stress Level Predict Model Code: classifier model

```
class DecisionTreeModel:
    def __init__(self):
        lysis_data=pd.read_csv("../Stress-Lysis.csv")
        hum_temp_data = lysis_data[['Humidity', 'Temperature']]
        str_lev_data = lysis_data['Stress Level']

        train_data , self.test_data , train_label , self.test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
        self.train_data , self.ver_data , self.train_label , self.ver_label =
train_test_split(train_data, train_label, test_size=0.25)

        self.decision_tree_model =
AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),
n_estimators=50).fit(self.train_data, self.train_label)
        self.adaboost_model = AdaBoostClassifier(n_estimators=50,
random_state=0).fit(self.train_data, self.train_label)

    def decision_tree_prediction(self, predict_data):
        return self.decision_tree_model.predict(predict_data)

    def adaboost_prediction(self, predict_data):
        return self.adaboost_model.predict(predict_data)


if __name__ == '__main__':

    DecisionTreeModel = DecisionTreeModel()

    predicted =
DecisionTreeModel.decision_tree_prediction(DecisionTreeModel.ver_data) #
0:Overcast, 2:Mild
    acc_train = accuracy_score(DecisionTreeModel.ver_label, predicted)

    predicted =
DecisionTreeModel.decision_tree_prediction(DecisionTreeModel.test_data)
    acc_test = accuracy_score(DecisionTreeModel.test_label, predicted)

    print("Decision Tree Verification Accuracy: {} \nDecision Tree Test Accuracy:
{}".format(acc_train, acc_test))

    predicted = DecisionTreeModel.adaboost_prediction(DecisionTreeModel.ver_data)
# 0:Overcast, 2:Mild
    acc_train = accuracy_score(DecisionTreeModel.ver_label, predicted)
```

```
    predicted =
DecisionTreeModel.adaboost_prediction(DecisionTreeModel.test_data)
    acc_test = accuracy_score(DecisionTreeModel.test_label, predicted)


    print("Adaboost Verification Accuracy: {} \nAdaboost Test Accuracy:
{}".format(acc_train, acc_test))
```

Stress Level Predict Model Code: K-NN

```
class KNNModel:
    def __init__(self):
        lysis_data=pd.read_csv("../Stress-Lysis.csv")
        hum_temp_data = lysis_data[['Humidity', 'Temperature']]
        str_lev_data = lysis_data['Stress Level']

        train_data , self.test_data , train_label , self.test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
        self.train_data , self.ver_data , self.train_label , self.ver_label =
train_test_split(train_data, train_label, test_size=0.25)

        self.knn_model = KNeighborsClassifier(n_neighbors=3).fit(self.train_data,
self.train_label)

    def knn_prediction(self, predict_data):
        return self.knn_model.predict(predict_data)


if __name__ == '__main__':

    KNNModel = KNNModel()

    predicted = KNNModel.knn_prediction(KNNModel.ver_data) # 0:Overcast, 2:Mild
    acc_train = accuracy_score(KNNModel.ver_label, predicted)

    predicted = KNNModel.knn_prediction(KNNModel.test_data)
    acc_test = accuracy_score(KNNModel.test_label, predicted)

    print("KNN Verification Accuracy: {} \nKNN Test Accuracy:
{}".format(acc_train, acc_test))
```

Stress Level Predict Model Code: linear

```
class LinearAnaiysis:

    def __init__(self):
        lysis_data=pd.read_csv("../Stress-Lysis.csv")

        self.temp_data = lysis_data['Temperature']
        self.humi_data = lysis_data['Humidity']
        self.step_data = lysis_data['Step count']
        self.str_lev_data = lysis_data['Stress Level']

        self.stress_level_classifier_data = [lysis_data[self.str_lev_data == 0],
lysis_data[self.str_lev_data == 1], lysis_data[self.str_lev_data == 2]]

        self.linear_name_x = ['Humidity', 'Temperature', 'Step count',
'Humidity', 'Temperature', 'Temperature']
        self.linear_name_y = ['Stress Level', 'Stress Level', 'Stress Level',
'Step count', 'Step count', 'Humidity']

    def get_data_by_name(self, name):
        if (name == 'Humidity'):
            return self.humi_data
        elif(name == 'Temperature'):
            return self.temp_data
        elif(name == 'Step count'):
            return self.step_data
        elif (name == 'Stress Level'):
            return self.str_lev_data
        else:
            return None

    def linear_with_correlation(self, x_name, y_name):
        x_data = self.get_data_by_name(x_name)
        y_data = self.get_data_by_name(y_name)
        correlation = x_data.corr(y_data)
        print("correlation coefficient between", x_name, "and", y_name, "is",
correlation)

        plt.scatter(x_data, y_data, alpha=0.8)
        plt.xlabel(x_name)
        plt.ylabel(y_name)
        plt.title("Correlation = {}".format(correlation))
        plt.show()
```

```
    def linear_with_correlation_all(self):
        plot_row = 0
        plot_column = 0

        fig, axs = plt.subplots(2, 3)

        for i in range(6):
            linear_data_x = self.get_data_by_name(self.linear_name_x[i])
            linear_data_y = self.get_data_by_name(self.linear_name_y[i])

            correlation = linear_data_x.corr(linear_data_y)
            # print("correlation coefficient between", self.linear_name_x[i],
"and", self.linear_name_y[i], "is", correlation)

            axs[plot_column, plot_row].scatter(linear_data_x, linear_data_y,
alpha=0.8)
            axs[plot_column, plot_row].set(xlabel=self.linear_name_x[i],
ylabel=self.linear_name_y[i])
            axs[plot_column, plot_row].set_title("Correlation =
{}".format(correlation))
            if (plot_row == 2):
                plot_column += 1
                plot_row = 0
            else:
                plot_row += 1


        fig.tight_layout()
        fig.set_figheight(6)
        fig.set_figwidth(12)
        plt.gcf().canvas.set_window_title('Linear analysis with any of two
labels')
        plt.show()



    def set_column_num(self, label):
        if (label == 'Humidity'):
            return 0
        elif(label == 'Temperature'):
            return 1
        elif(label == 'Step count'):
            return 2
        else:
            return None
```

```
    def linear_show_with_stress_level_all_in_one(self, x_name, y_name):
        x_num = self.set_column_num(x_name)
        y_num = self.set_column_num(y_name)

        stress_0 = self.stress_level_classifier_data[0]
        stress_1 = self.stress_level_classifier_data[1]
        stress_2 = self.stress_level_classifier_data[2]

        plt.scatter(stress_0.iloc[:, x_num], stress_0.iloc[:, y_num], c="red",
marker='o', label='Stress Level 0')
        plt.scatter(stress_1.iloc[:, x_num], stress_1.iloc[:, y_num], c="blue",
marker='*', label='Stress Level 1')
        plt.scatter(stress_2.iloc[:, x_num], stress_2.iloc[:, y_num], c="yellow",
marker='+', label='Stress Level 2')

        plt.xlabel(x_name)
        plt.ylabel(y_name)

        plt.show()

    def linear_show_with_stress_level_split_to_three(self, x_name, y_name):

        x_num = self.set_column_num(x_name)
        y_num = self.set_column_num(y_name)
        color = ["red", "blue", "yellow"]
        marker = ['o', '*', '+']
        num = 0

        for stress_level in self.stress_level_classifier_data:
            label_name = "Stress Level" + str(num)
            plt.scatter(stress_level.iloc[:, x_num], stress_level.iloc[:, y_num],
c=color[num], marker=marker[num], label=label_name)
            plt.xlabel(x_name)
            plt.ylabel(y_name)

            plt.show()
            num += 1

    def linear_show_with_stress_level_all(self):
        linear_name_x = ['Humidity', 'Temperature', 'Temperature']
        linear_name_y = ['Step count', 'Step count', 'Humidity']
        color = ["red", "blue", "yellow"]
        marker = ['o', '*', '+']
```

```
        stress_0 = self.stress_level_classifier_data[0]
        stress_1 = self.stress_level_classifier_data[1]
        stress_2 = self.stress_level_classifier_data[2]

        plot_row = 0
        plot_column = 0

        fig, axs = plt.subplots(3, 4)

        for i in range(3):
            x_num = self.set_column_num(linear_name_x[i])
            y_num = self.set_column_num(linear_name_y[i])

            axs[plot_column, plot_row].scatter(stress_0.iloc[:, x_num],
stress_0.iloc[:, y_num], c="red", marker='o', label='Stress Level 0')
            axs[plot_column, plot_row].scatter(stress_1.iloc[:, x_num],
stress_1.iloc[:, y_num], c="blue", marker='*', label='Stress Level 1')
            axs[plot_column, plot_row].scatter(stress_2.iloc[:, x_num],
stress_2.iloc[:, y_num], c="yellow", marker='+', label='Stress Level 2')

            axs[plot_column, plot_row].set(xlabel=linear_name_x[i],
ylabel=linear_name_y[i])
            axs[plot_column, plot_row].set_title("Stress Level 0 ~ 2")
            plot_row += 1
            num = 0

            for stress_level in self.stress_level_classifier_data:
                label_name = "Stress Level" + str(num)
                axs[plot_column, plot_row].scatter(stress_level.iloc[:, x_num],
stress_level.iloc[:, y_num], c=color[num], marker=marker[num], label=label_name)
                axs[plot_column, plot_row].set(xlabel=linear_name_x[i],
ylabel=linear_name_y[i])
                axs[plot_column, plot_row].set_title("Stress Level
{}".format(num))

                num += 1
                plot_row += 1

            if (plot_row == 4):
                plot_column += 1
                plot_row = 0
            else:
                plot_row += 1
```

```
        fig.tight_layout()
        fig.set_figheight(8)
        fig.set_figwidth(15)
        plt.gcf().canvas.set_window_title('Linear analysis with stress level')
        plt.show()


if __name__ == '__main__':

    linear_analysis = LinearAnaiysis()

    linear_analysis.linear_with_correlation_all()
    linear_analysis.linear_show_with_stress_level_all()
```

Stress Level Predict Model Code: SVM

```
class SVMModel:
    def __init__(self):
        lysis_data=pd.read_csv("../Stress-Lysis.csv")
        hum_temp_data = lysis_data[['Humidity', 'Temperature']]
        str_lev_data = lysis_data['Stress Level']

        train_data , self.test_data , train_label , self.test_label =
train_test_split(hum_temp_data, str_lev_data, test_size=0.2)
        self.train_data , self.ver_data , self.train_label , self.ver_label =
train_test_split(train_data, train_label, test_size=0.25)


        self.svc_model = svm.SVC(kernel='linear', C=2).fit(self.train_data,
self.train_label)
        self.rbf_svc_model = svm.SVC(kernel='rbf', gamma=0.7,
C=2).fit(self.train_data, self.train_label)
        self.poly_svc_model = svm.SVC(kernel='poly', degree=3,
C=2).fit(self.train_data, self.train_label)
        self.lin_svc_model = svm.LinearSVC(C=2, dual=False).fit(self.train_data,
self.train_label)

    def svc_prdiction(self, predict_data):
        return self.svc_model.predict(predict_data)

    def rbf_svc_prdiction(self, predict_data):
        return self.rbf_svc_model.predict(predict_data)

    def poly_svc_prdiction(self, predict_data):
        return self.poly_svc_model.predict(predict_data)

    def lin_svc_prdiction(self, predict_data):
        return self.lin_svc_model.predict(predict_data)

if __name__ == '__main__':

    SVMModel = SVMModel()

    pred_ver = SVMModel.svc_prdiction(SVMModel.ver_data)
    acc_train = accuracy_score(SVMModel.ver_label, pred_ver)

    pred_test = SVMModel.svc_prdiction(SVMModel.test_data)
    acc_test = accuracy_score(SVMModel.test_label, pred_test)
```

```
    print("SVM Verification Accuracy: {} \nSVM Test Accuracy:
{}".format(acc_train, acc_test))

    pred_ver = SVMModel.rbf_svc_prdiction(SVMModel.ver_data)
    acc_train = accuracy_score(SVMModel.ver_label, pred_ver)

    pred_test = SVMModel.rbf_svc_prdiction(SVMModel.test_data)
    acc_test = accuracy_score(SVMModel.test_label, pred_test)

    print("SVM RBF Verification Accuracy: {} \nSVM RBF Test Accuracy:
{}".format(acc_train, acc_test))

    pred_ver = SVMModel.poly_svc_prdiction(SVMModel.ver_data)
    acc_train = accuracy_score(SVMModel.ver_label, pred_ver)

    pred_test = SVMModel.poly_svc_prdiction(SVMModel.test_data)
    acc_test = accuracy_score(SVMModel.test_label, pred_test)

    print("SVM Poly Verification Accuracy: {} \nSVM Poly Test Accuracy:
{}".format(acc_train, acc_test))

    pred_ver = SVMModel.lin_svc_prdiction(SVMModel.ver_data)
    acc_train = accuracy_score(SVMModel.ver_label, pred_ver)

    pred_test = SVMModel.lin_svc_prdiction(SVMModel.test_data)
    acc_test = accuracy_score(SVMModel.test_label, pred_test)

    print("SVM Linear Verification Accuracy: {} \nSVM Linear Test Accuracy:
{}".format(acc_train, acc_test))
```

Stress Level Predict Model Code: run model accuracy

```
def svm(classifier_model):
    pred_ver = classifier_model.svc_prediction(classifier_model.ver_data)
    acc_train = accuracy_score(classifier_model.ver_label, pred_ver)

    pred_test = classifier_model.svc_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, pred_test)
    result = "SVM Verification Accuracy: {} \nSVM Test Accuracy:
{}".format(acc_train, acc_test)
    # print(result)
    return result

def svm_rbf(classifier_model):
    pred_ver = classifier_model.rbf_svc_prediction(classifier_model.ver_data)
    acc_train = accuracy_score(classifier_model.ver_label, pred_ver)

    pred_test = classifier_model.rbf_svc_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, pred_test)

    result = "SVM RBF Verification Accuracy: {} \nSVM RBF Test Accuracy:
{}".format(acc_train, acc_test)

    # print(result)
    return result

def svm_poly(classifier_model):
    pred_ver = classifier_model.poly_svc_prediction(classifier_model.ver_data)
    acc_train = accuracy_score(classifier_model.ver_label, pred_ver)

    pred_test = classifier_model.poly_svc_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, pred_test)

    result = "SVM Poly Verification Accuracy: {} \nSVM Poly Test Accuracy:
{}".format(acc_train, acc_test)

    # print(result)
    return result

def svm_linear(classifier_model):
    pred_ver = classifier_model.lin_svc_prediction(classifier_model.ver_data)
    acc_train = accuracy_score(classifier_model.ver_label, pred_ver)

    pred_test = classifier_model.lin_svc_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, pred_test)
```

```
    result = "SVM Linear Verification Accuracy: {} \nSVM Linear Test Accuracy:
{}".format(acc_train, acc_test)

    # print(result)
    return result

def knn(classifier_model):
    predicted = classifier_model.knn_prediction(classifier_model.ver_data) #
0:Overcast, 2:Mild
    acc_train = accuracy_score(classifier_model.ver_label, predicted)

    predicted = classifier_model.knn_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, predicted)

    result = "KNN Verification Accuracy: {} \nKNN Test Accuracy:
{}".format(acc_train, acc_test)

    # print(result)
    return result

def decision_tree(classifier_model):
    predicted =
classifier_model.decision_tree_prediction(classifier_model.ver_data) #
0:Overcast, 2:Mild
    acc_train = accuracy_score(classifier_model.ver_label, predicted)

    predicted =
classifier_model.decision_tree_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, predicted)

    result = "Decision Tree Verification Accuracy: {} \nDecision Tree Test
Accuracy: {}".format(acc_train, acc_test)

    # print(result)
    return result

def adaboost(classifier_model):
    predicted = classifier_model.adaboost_prediction(classifier_model.ver_data) #
0:Overcast, 2:Mild
    acc_train = accuracy_score(classifier_model.ver_label, predicted)

    predicted = classifier_model.adaboost_prediction(classifier_model.test_data)
    acc_test = accuracy_score(classifier_model.test_label, predicted)
```

```
    result = "Adaboost Verification Accuracy: {} \nAdaboost Test Accuracy:
{}".format(acc_train, acc_test)

    # print(result)
    return result

if __name__ == '__main__':
    classifier_model = ClassifierModel()

    svm(classifier_model)
    svm_rbf(classifier_model)
    svm_poly(classifier_model)
    svm_linear(classifier_model)
    knn(classifier_model)
    decision_tree(classifier_model)
    adaboost(classifier_model)
```