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)**

[1. Introduction 1](#_Toc105769419)

[Motivation: 1](#_Toc105769420)

[Objectives: 1](#_Toc105769421)

[2. Literature review and related works 1](#_Toc105769422)

[3. Problem statement 2](#_Toc105769423)

[4. Proposed models (approaches) 2](#_Toc105769424)

[5. Experiments 3](#_Toc105769425)

[K-NN 9](#_Toc105769426)

[Decision\_Tree\_with\_Ada\_Boost\_Classifier 10](#_Toc105769427)

[SVM 12](#_Toc105769428)

[Stress Level Predict Model 14](#_Toc105769429)

[Stress Level Predict Model Code: GUI 15](#_Toc105769430)

[Stress Level Predict Model Code: classifier model 25](#_Toc105769431)

[Stress Level Predict Model Code: classifier model 26](#_Toc105769432)

[Stress Level Predict Model Code: K-NN 28](#_Toc105769433)

[Stress Level Predict Model Code: linear 29](#_Toc105769434)

[Stress Level Predict Model Code: SVM 33](#_Toc105769435)

[Stress Level Predict Model Code: run model accuracy 35](#_Toc105769436)

[6. Conclusion 38](#_Toc105769437)

[7. Others 38](#_Toc105769438)

**Abstract**

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.

1. Introduction

This part is a summary about the whole report and project. Please have the objectives of your project and the motivation in the introduction. In addition, the applications and contributions should be included.

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)”.

1. Literature review and related works

Please write your comments and comparisons to the related materials you have reviewed. If there are some related or similar works, please also state and have a discussion and comparison.

(之前找的2篇論文，或其他的)

1. Problem statement

Give a scenario of the problem you are working and state the problem. If possible, you can try to formulate your problem formally.

1. Proposed models (approaches)

Please present your proposed models (approaches) for the problem and give the reasons why your models or approaches are designed. Give one or two examples to illustrate how your models run.

Use the classifier algorithm instead of regression algorithm because...

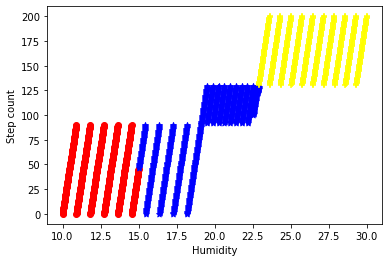
(regression是用連續數據去做，我們的比較像分類)

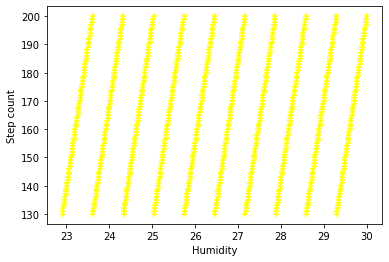
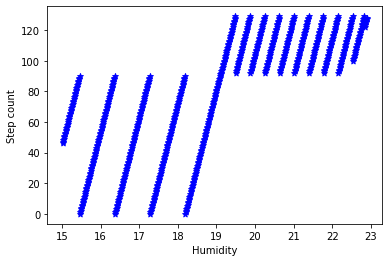
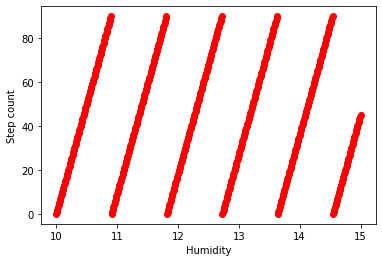
<https://ikala.cloud/supervised-learning-classification-regression/>

1. Experiments

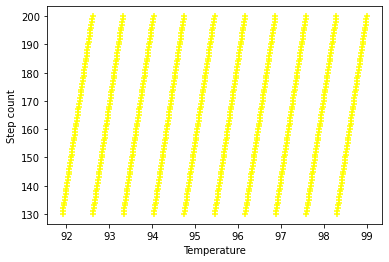
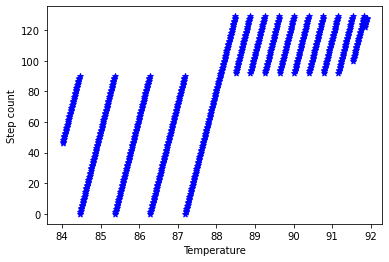
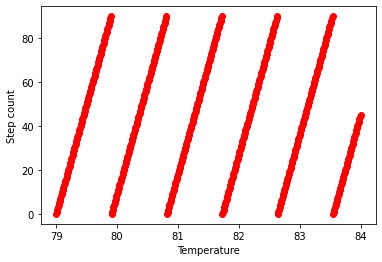
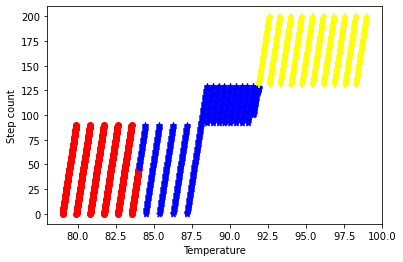
This part is very important. It first includes what the objectives of the experiments are, how you design them, which tools are used, where the data are from, when you have the data, what the measurements are, and what you will compare with. Then, the results are shown with careful analysis and discussion. One should have reasons for all the trends presented in the results.

**Classify by Humidity and Step Count:**

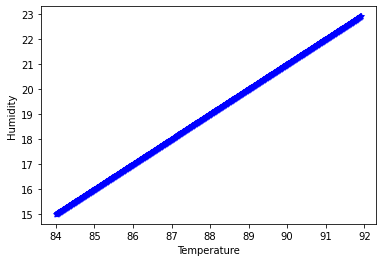
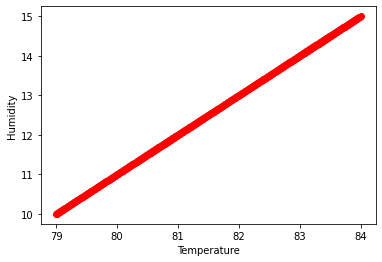
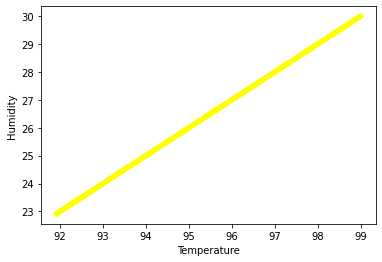
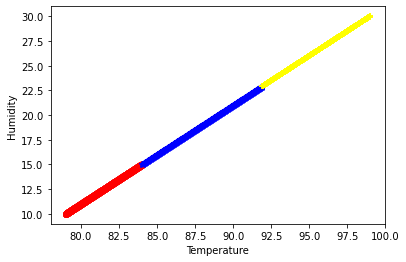




**Classify by Temperature and Step Count:**



**Classify by Temperature and Humidity:**



**Correlation coefficient between humidity and stress level is 0.93**



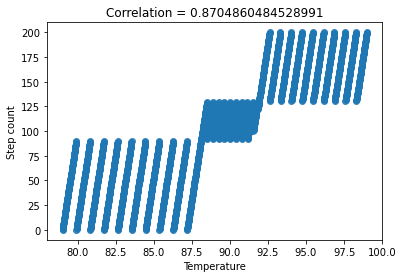
**Correlation coefficient between temperature and stress level is 0.93**



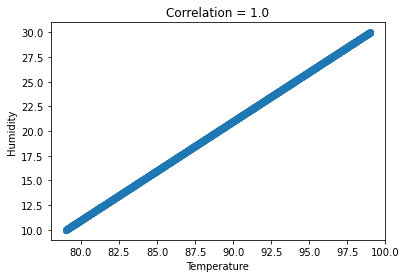
**Correlation coefficient between step count and stress level is 0.83**



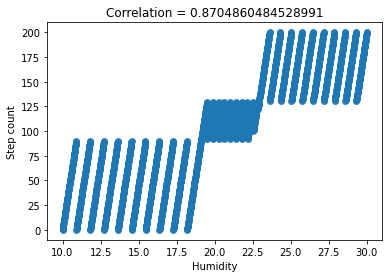
**Correlation coefficient between temperature and step count is 0.87**



**Correlation coefficient between temperature and humidity is 1.0**



**Correlation coefficient between humidity and step count is 0.87**



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

Decision\_Tree\_with\_Ada\_Boost\_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

Through the previous analysis, we found:

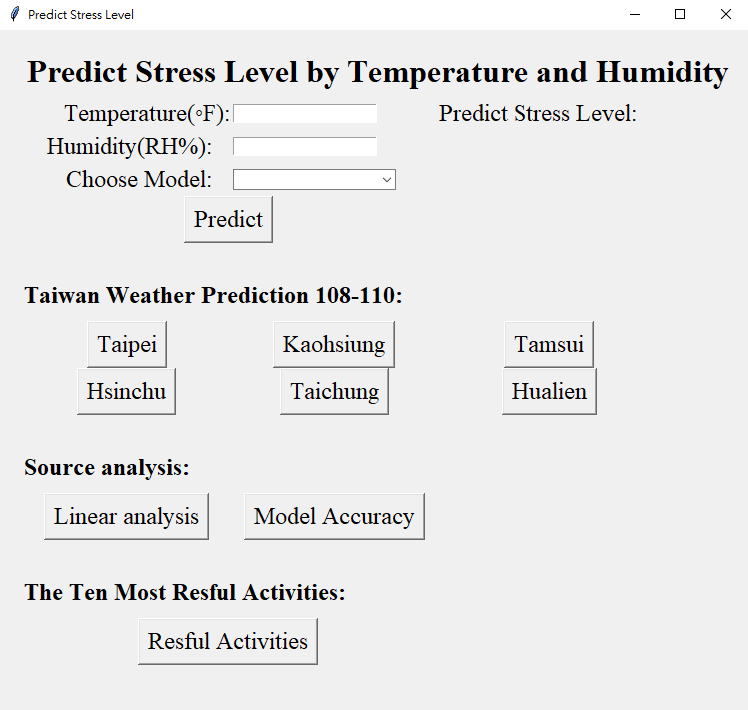
* the higher temperature, the higher pressure
* the higher humidity, the higher pressure
* the higher the pressure, the more steps you take.

Stress Level Predict Model

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.



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 = [60.8,66.38,68.54,72.32,82.76,84.74,86.54,84.02,85.1,78.26,70.16,64.94,64.22,65.66,69.44,69.62,80.42,86.9,87.62,86.36,82.04,76.1,73.94,64.58,65.3,65.84,67.64,75.56,77,83.3,86.54,86.9,81.14,77.54,71.6,66.38]

    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 = [59,64.04,65.3,70.7,80.6,83.12,85.1,83.12,83.84,77.36,68.9,63.14,62.24,63.14,67.28,68.18,78.62,84.38,85.82,84.56,79.88,75.2,72.68,63.32,63.68,63.86,66.38,73.58,75.56,81.32,85.28,84.92,79.34,75.74,70.34,64.4]

    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 = [59.54,64.22,66.56,71.78,81.5,83.66,85.64,83.48,84.74,78.8,69.98,64.22,62.06,63.68,68,68.9,79.88,85.46,87.44,85.28,81.5,76.82,73.94,64.4,63.86,64.4,66.74,74.48,76.46,82.94,86.36,85.1,80.96,77.36,71.24,64.94]

    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 = [60.8,66.2,70.7,75.02,83.84,82.04,84.2,82.04,84.2,80.06,72.32,65.84,64.4,66.2,72.14,71.96,81.5,84.38,85.28,83.3,82.58,78.98,75.02,67.1,66.74,68.9,69.98,76.46,77.72,82.76,84.92,83.12,81.86,78.98,73.4,66.74]

    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 = [61.88,67.1,70.7,72.86,80.42,82.4,84.56,82.76,82.4,77.36,70.88,67.1,67.1,67.28,70.52,70.7,78.98,84.74,85.1,84.38,81.32,77.54,74.3,68.36,68,69.8,69.62,76.1,75.92,81.86,84.74,84.02,81.32,77.54,73.22,68.36]

    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)

**Tools:**

1. Jupyter notebook

2. Python Library

**Data Sources:**

1. Conclusion

Please include what the project has done and whether the objectives are achieved. What do you get in the project and what can be donein the future?

(收獲+未來展望)

1. Others

Please state the workload and role of each member in your team for the project. Besides, show the timeline for the project and check whether all the proposed works have been done.

(分工+timeline)

**Reference**