Final Project Submission

STATS 202: Data Mining and Analysis

Stanford University, Summer Session 2023

Instructor: Dr. Linh Tran

Submitted by:
Adam Kainikara
adamkainikara@gmail.com

Project Overview

This document contains my final project for STATS 202, completed in the Summer of 2023 at Stanford University. The project is divided into two parts:

- **Part I** – Final report
- **Part II** – Python code (approximately 1000 lines)

The project applies various machine learning and data analysis techniques to a clinical dataset of patients diagnosed with schizophrenia. The dataset includes 10 attributes and a total of 110,047 observations (80,046 for training and 30,001 for testing).

Model outputs were submitted to two internal Kaggle competitions:

1. **PANSS Score Prediction:** Predicting the Positive and Negative Syndrome Scale (PANSS) score to assess the severity of schizophrenia.
2. **Treatment Classification:** Classifying patients into categories such as continuing treatment, switching treatment, or discharge.

I placed 2nd in the PANSS prediction task and 7th in the classification task, resulting in a 4th overall ranking out of 40 students who participated in the competition.

Thank you for taking the time to review my final project. To contact or discuss any material of the project, please contact me at: *adamkainikara@gmail.com*

Please continue to the following pages.

STATS 202: DATA MINING AND ANALYSIS FINAL

Instructor: Linh Tran

Final Project

Due Date: August 2, 2023

Stanford University

Adam Kainikara

Kaggle Reference

```
Team Name: Adam Kainikara.
Work done by Adam Kainikara.
```

Part 1. Treatment Effect

Using Python I loaded the data from all available CSV sets. Created a function called find_patients which took the input of the loaded data and returned every unique patient id. After collecting every patient id, I created a blank dictionary. The dictionary would have the patient id as the key. For values, I stored all the associated values of each patient in an array. For patients that had one or more visit days, the information would also be stored. The following is an example of how I stored the data.

dtype=[('study', '<U10'), (country, '<U10'), ('txgroup', '<U10'), ('assesmentid', '<f8'), ('patientid', '<f8'), ('visitday', '<i4'), ('xvalues', '<f8', (30,)), ('panss', '<f8'), ('leadstatus', '<U10')]), 30951.0: array([('"C"', '"China"', '"Control"', 304958., 30951., 0, [4., 3., 2., 1., 3., 4., 2., 4., 5., 4., 5., 4., 5., 3., 1., 2., 1., 1., 4., 1., 4., 4., 3., 3., 4., 6., 3., 1., 5., 2.], 94., '"Assign to'), ('"C"', '"China"', '"Control"', 301327., 30951., 7, [4., 3., 2., 1., 2., 4., 2., 4., 5., 4., 5., 4., 5., 3., 1., 1., 1., 1., 4., 1., 4., 4., 3., 3., 4., 6., 3., 1., 4., 2.], 91., '"Assign to'), ('"C"', '"China"', '"Control"', 303725., 30951., 14, [4., 3., 2., 1., 1., 4., 2., 4., 5., 4., 5., 4., 5., 3., 1., 1., 1., 1., 4., 2., 4., 4., 3., 3., 4., 6., 3., 1., 4., 2.], 91., '"Passed"'), ('"C"', '"China"', '"Control"', 304954., 30951., 42, [4., 3., 4., 1., 1., 4., 2., 4., 5., 5., 5., 4., 5., 3., 1., 2., 1., 1., 4., 3., 4., 4., 3., 3., 4., 6., 3., 2., 5., 2.], 98., '"Passed"'), ('"C"', '"China"', '"Control"', 307645., 30951., 70, [4., 2., 2., 3., 1., 4., 3., 5., 5., 6., 6., 5., 5., 4., 2., 3., 1., 1., 4., 3., 5., 5., 3., 3., 4., 6., 3., 3., 5., 2.], 108., '"Passed"')

After this was done, patients were further filtered into a control group and a treatment group. In addition, patients with fewer than one visit day were removed. Below is a graph of treatment and control groups. The X axis is visit day and Y axis is PANSS score. To reduce clutter on the graph, only Study E is shown.

After, I calculated the difference between each patient's final PANNS score and their initial score. The distribution of the difference in scores can be seen bellow.

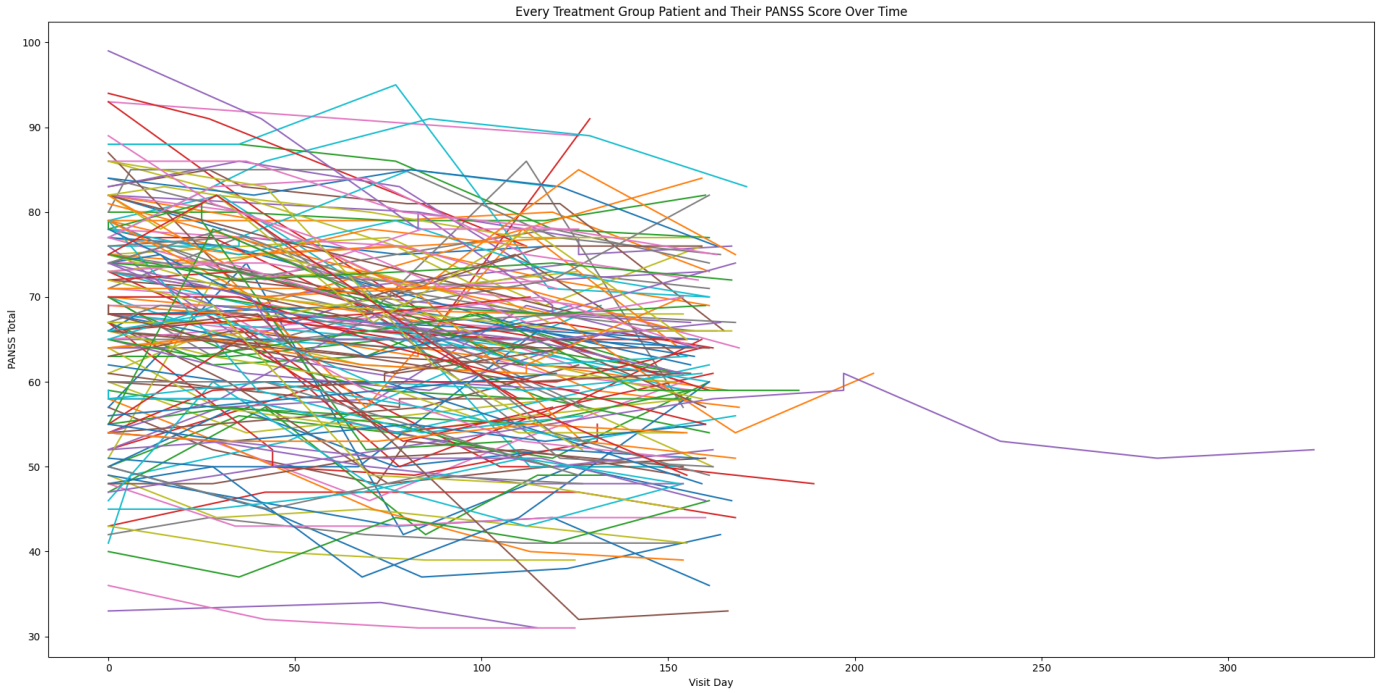


FIGURE 0.1. Treatment Group: Study E

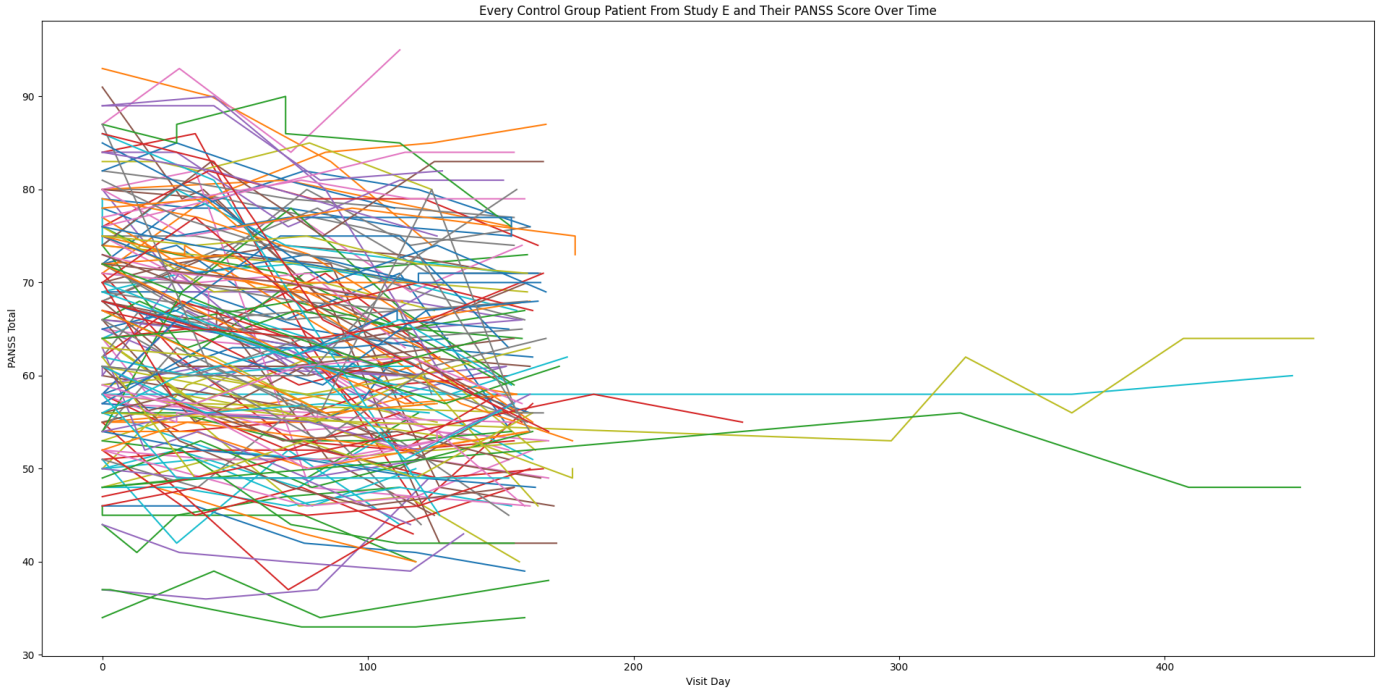


FIGURE 0.2. Control Group: Study E

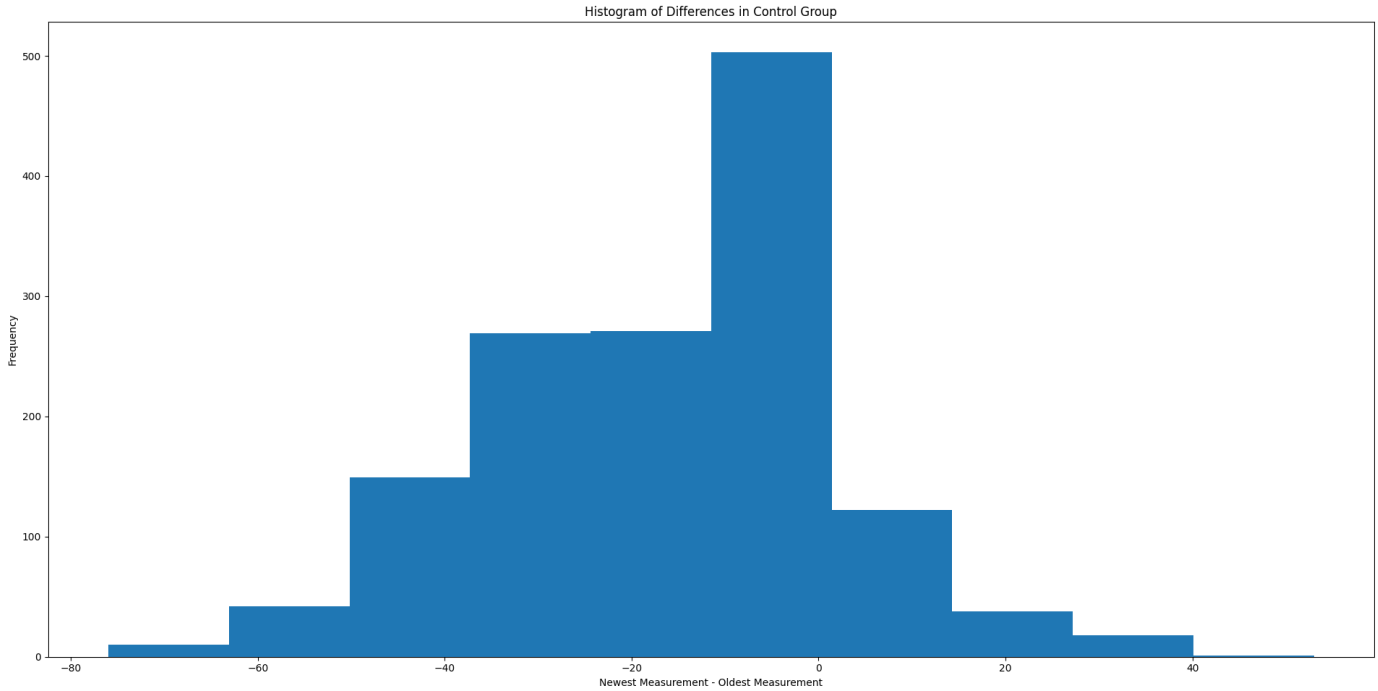


FIGURE 0.3. Differences in Control Group

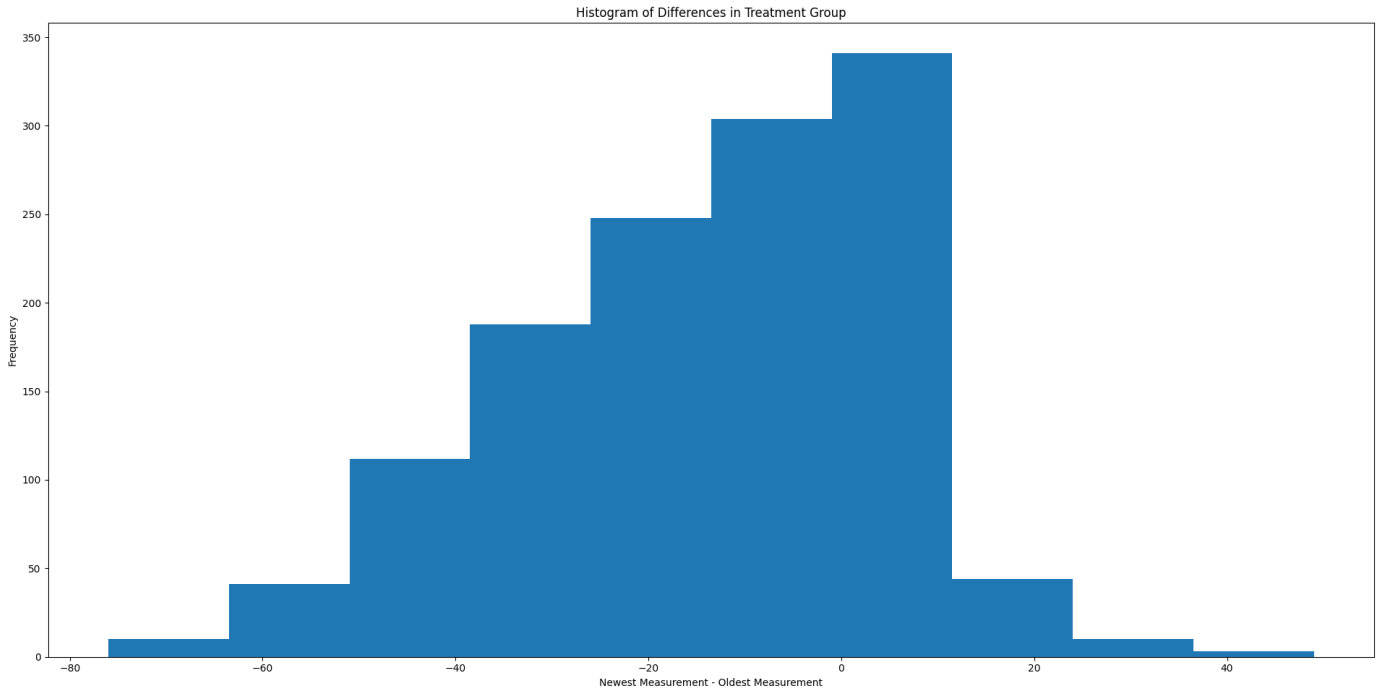


FIGURE 0.4. Differences in Treatment Group

The distribution of difference scores in the control group is somewhat normal where as the distribution of difference of scores in the treatment group is skewed.The mean difference of patients in the control group was -16.13 with a standard deviation of 19.40. That is on average, patients in the control group on average saw their PANSS score decrease by 16 at the end of the study. The mean difference of patients in the treatment group was -15.98 with a standard deviation of 19.24.

To see if the treatment had any affect, a t test will be conducted where H_0 is there is no significant difference in the mean change in score between the treatment and control groups and H_a is there is a significant difference in the mean change in score between the treatment and control groups. Patients were randomly assigned to either treatment or control. With $\alpha = 0.05$ a test statistic of 0.2105 and 2945 degress of freedom. The p value is 0.416. The result is not significant. The treatment affect does not make a significant impact on the decrease in PANSS Scores.

Part 2. Patient Segmentation

For this problem I used two forms of clustering. One being k means clustering, and the other being hierarchical clustering so I could see a dendogram. For K means clustering, I began by clustering on various components that made up the PANSS score. For example I tried P1 with N1, P1 with G1 etc. I choose 3 centroids. I also used cross validation to choose the best k. In the scatter plot below, it shows the locations of the centorids of the clusters when clustering using P1 and N1. The centroids are in orange. The blue dots are the xvalue combinations with P1 and N1.

I then wanted to know what a dendogram of this would look like, so one was constructed. There where two clusters. One thing I found quite interesting is that one of the clusters (green is N1) is larger than the other cluster. I thought that the clusters would be of similar size. Also the clusters don't meet for a while.

Part 3. Forecasting

Kaggle Reference. Team Name: Adam Kainikara.

Work done by Adam Kainikara.

There were many data wrangling steps involved in this part of the project. I used data from set E for this part. At first, I created a structured array which was organized by study, country, txgroup, assesment id, patient id, visit day, xvalues, panss score and lead status. Then I made a function that found every unique patient id. This was to figure out how many patients were in the study. After collecting every patient id, I created a blank dictionary. The dictionary would have the patient id as the key. For values, I stored all the associated values of each patient in an array. These associated values include: PANSS score, country, etc. For patients that had one or more visit days, the information would also be stored in arrays. It is stored in the same way as seen in section 1. Patients who did not have at least two visit days where removed from the study. In a spread sheet, I did some basic calculations. These calculations included figuring out the average % change for each patient from the initial reading in the study to the last reading in the study.

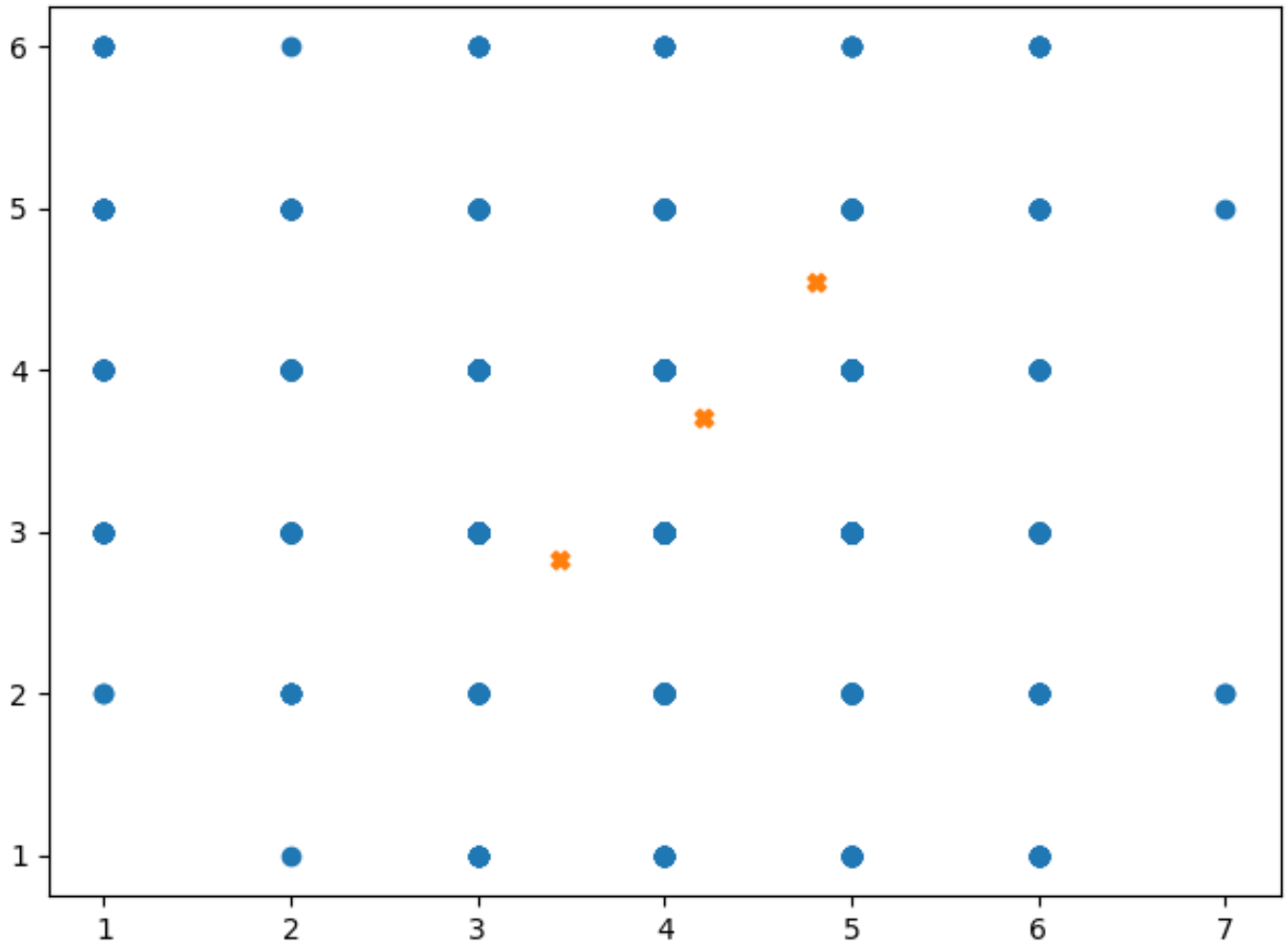


FIGURE 0.5. K Means Clustering

For my first prediction, I uploaded a CSV that contained the most recent visit day's PANSS score. I did this to see if the most recent visit PANSS score would be similar to the real 18th week score. This resulted in a score of 6.61736.

My next submission was the score decreased by the average decrease calculated in the spread sheet. To do this, I calculated each patients change in PANSS score from the first week to the last week. I found the average % drop across the data set. I then changed the initial reading (first day's reading) by the average % change.The average % change across the data set was about 10%. This did not change my score significantly.

For my final few submissions I created a function that did linear interpolation where:
$\alpha = \frac{y_2 - y_1}{x_2 - x_1}$ and
$\hat{y} = (1 - \alpha)y_1 + \alpha y_2$

The linear interpolation resulted in better results. My best score was 6.5099. In my created function, I tried several variations. The first variation I did was predict the average PANSS score if a patient did not get to day 129.5. I used day 129.5 as it is 18.5 weeks. In my spreadsheet I calculated that the average visit day turns into about 17.6 weeks. Although day

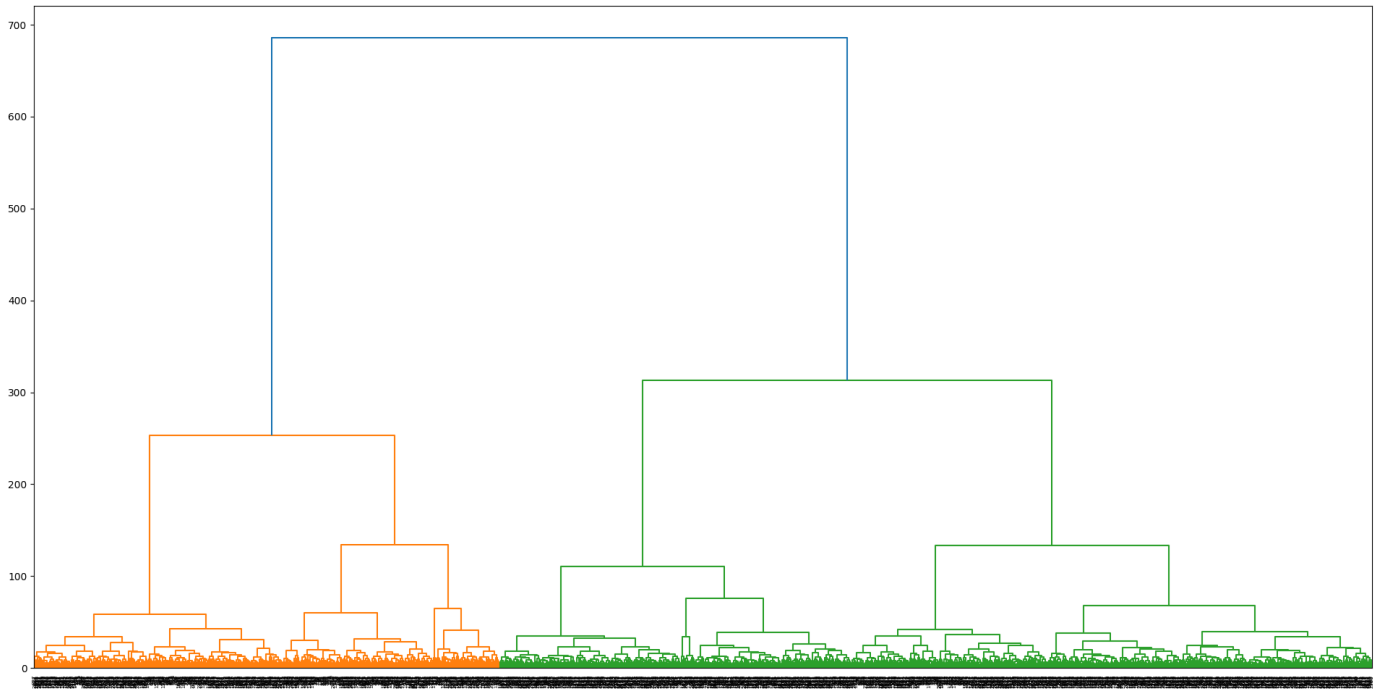


FIGURE 0.6. Dendogram

126 (18 weeks) does not necessarily mean that that was the patient's 18th week, I assumed that most patients would be around this time period. This assumption was made because the average patient stayed for 17.6 weeks.

Another variation I did if a patient did not make it to day 129.5 was to calculate the slope between the last two points, fit a line, and predict what the score for day 129.5 would have been.

The last attempt I did involved extrapolation. When I used extrapolation though, my score got worse (7.74936) so I did not use extrapolation. In the end using linear interpolation with the day as 129.5 days produced the best results for me. The code for this section is the function(s) desired_data in the interpol3.py file.

Part 4. 4 Binary Classification

Kaggle Reference

Team Name: Adam Kainikara.

Work done by Adam Kainikara.

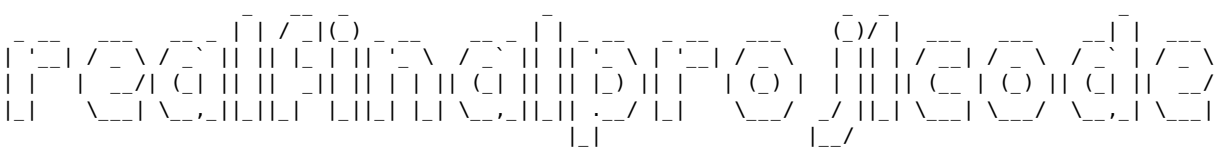
There were many data wrangling steps involved in this part of the project. I used data sets A,B,C,D for training and E as the test set. At first, I created a structured array which was organized by study, country, txgroup, assesment id, patient id, visit day, xvalues, panss score and lead status. Then made a function that found every unique patient id. This was to figure out how many patients were in the study. After collecting every patient id, I created a blank dictionary. The dictionary would have the patient id as the key. For values, I stored all the associated values of each patient in an array. For patients that had one or more visit

days, the information would also be stored. Patients who did not have at least two visit days where removed from the study.

For classification I used many classification methods. These include knn, naive bayes, decision trees, random forests and support vector machines. When using knn, I choose many different K values, however my accuracy was never very good so I decided not to use it. My score on kaggle was 16. The others (naive bayes, decision trees, random forests) produced poor training accuracy.

My best scores occurred while using a support vector machine. The support vector machine gave me a score of 0.858 with a training accuracy of 0.812.

To see if I could improve my score, I tried using different kernels such as linear or sigmoid however these did not improve my score. I did add a regularization parameter of 0.95. This changed my score to 0.854



finalp3.py

```
from numpy import *
import sys
import matplotlib.pyplot as plt
patient_visit_dt = dtype([('study','U10'),('country','U10'),('txgroup','U10'),('patientid', float64),('visitday', int32),('xvalues',float64,(30)),('panss',float64)])

def data_loader(fname):
    #Study      Country PatientID       SiteID  RaterID AssessmentID    TxGroup VisitDay        P1      P2      P3      P4      P5      P6      P7      N1      N2      N3      N4      N5
    # 0     1       2           3       4       5               6       7           8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    # G7        G8      G9      G10     G11     G12     G13     G14     G15     G16     PANSS_Total
    # 28    29  30  31  32  33  34  35  36  37  38

    #data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',')
    cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',', dtype=float64)
    #print(cata_data_a, quant_data_a)
    #print(cata_data_a.shape, quant_data_a.shape)
    #print(type(cata_data_a), type(quant_data_a))


    data_a = empty(quant_data_a.shape[0], dtype=patient_visit_dt)
    #print(data_a.shape)

    #raise SystemExit
    #visit_a = array((visit_a[['study', 'country', 'txgroup']],visit_a['xvalues'],visit_a['yvalues']), dtype = patient_visit_dt)
    #x_v = quant_data_a[:,5:35]

    #y_v = quant_data_a[:,35]
    #print(data_a[['study', 'country', 'txgroup']].shape, cata_data_a.shape)
    data_a['study']  = cata_data_a[:, 0]
    data_a['country']  = cata_data_a[:, 1]
    data_a['txgroup']  = cata_data_a[:, 2]
    data_a['patientid'] = quant_data_a[:,0]
    data_a['visitday'] = quant_data_a[:,4]
    #print("eeeee", quant_data_a[:,4])
    #print("aaaa", quant_data_a[:,0])

    #print(quant_data_a[:,5:35])
    #print(quant_data_a[:,35])

    data_a['xvalues'] = quant_data_a[:,5:35]
    data_a['panss'] = quant_data_a[:,35]
    #print(data_a['xvalues'].shape, data_a['panss'].shape)
    #print(data_a)

    #data_a[['study', 'country', 'txgroup']] = tuple(cata_data_a[:, i] for i in range(3))
    #print('!!', cata_data_a[:2])
    #data_a[['study', ] = cata_data_a[0, :2]
    #cata_data_a[:, :2]

    #visit_a['xvalues'] = x_v
    #visit_a['yvalues'] = y_v
    #visit_a[['country', 'study']] = cata_data_a[:,0], cata_data_a[:,1]
    return data_a, cata_data_a, quant_data_a

def trial():
    num_l = []
    for i in range(100):
        num_l.append(i)
        #print(i)
    return num_l

def find_patients(data_a):
    d = {}
    #rows = range(data_a.shape[0])
    for i in range(data_a.shape[0]):
        key = data_a[i]['patientid']
        if key in d:
            d[key].append(i)
        else:
            d[key] = [i]

    patient_d = {}

    for patientid, index_l in d.items():
        patient_a = data_a[index_l]
        index_v = patient_a['visitday'].argsort()
        patient_d[patientid] = patient_a[index_v]

    print(type(patient_d))
    print("test", patient_d)
    return patient_d

def seperate_control_treatment(patient_d):
    control_l = []
    treatment_l = []

    for patient_a in patient d.values():
        if patient_a[0]['txgroup'] == '"Control"':
            control_l.append(patient_a)
        else:
            treatment_l.append(patient_a)
    #print(control_l, treatment_l)
    return control_l, treatment_l

def plot_patient_data(data_l):
    print(type(data_l))
    print(len(data_l))

    fig, ax = plt.subplots()

    for patient_a in data_l:

        x_l = patient_a['visitday']
        y_l = patient_a['panss']
        ax.plot(x l, y l)
    plt.xlabel('Visit Day')
    plt.ylabel('PANSS Total')
    plt.title('Every Treatment Group Patient and Their PANSS Score Over Time')
    plt.show()
def filter_patients(patient_d, day_limit=126):
    print(type(patient_d))
    print(patient_d)
    #delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] < day_limit]
    delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] - patient_a[0]['visitday'] < day_limit]
    for patientid in delete_patient_l:
        del patient_d[patientid]
def fit_linearmodel():
    pass
def main():
    #data_a = hstack([data_loader(fname) for fname in sys.argv[1:]])

    data_a,cata_data_a, quant data a = data_loader("Study_E.csv")
    patientid_v = data_a['patientid']
    upatientid v = unique(patientid_v)
    print('Patient id:', upatientid_v.shape[0])

    print("just viewing", patientid_v)
    #print(da)

```

```
    # day_v = data_a['visitday']
    # # uday_v = unique(day_v)
    # week_v = uday_v/7
    # print(uday_v)
    # print('------------')
    # print('------------')
    # print(week_v)
    # print('------------')

    # print(f'data a:10 {data_a[10:]}')
    m_v = data_a['txgroup'] ==   '"Treatment'
    day_v = data a[m v]['visitday']
    print('treatment day', sorted(day_v))
    uday_v = unique(day_v)
    # week_v = uday_v//7
    # print(f'uday_v treatments: {uday_v}')
    # print(f'week_v treatments: {week_v}')
    # #week_v = int32(week_v)
    # print('week count:', list(zip(arange(week_v.shape[0]), bincount(week_v))))
    patient d = find patients(data_a)
    print('Before delete:', len(patient_d))
    filter_patients(patient_d, day_limit=105)
    print('After delete:', len(patient_d))

    #print(len(a))
    print('------------')
    #print(d)

    control_l, treatment_l = seperate_control_treatment(patient_d)
    print("control", control_l)
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    print("treatment", treatment_l)

    plot_patient_data(treatment_l)
    plot_patient_data(control_l)
    raise SystemExit
    plt.xlabel('Visit Day')
    plt.ylabel('PANSS Total')
    plt.title('Every Treatment Group Patient and Their PANSS Score Over Time')
if __name__ == '__main__':
    main()
```

interpolp3.py

```
from cgi import test
from numpy import *
import sys
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

patient_visit_dt = dtype([('study','U10'),('country','U10'),('txgroup','U10'),('assesmentid', float64),('patientid', float64),('visitday', int32),('xvalues',float64,(30)),('panss',float64),

def data_loader(fname):
    #Study      Country PatientID       SiteID  RaterID AssessmentID    TxGroup VisitDay        P1      P2      P3      P4      P5      P6      P7      N1      N2      N3      N4      N5
    # 0     1       2           3       4       5               6       7           8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    # G7        G8      G9      G10     G11     G12     G13     G14     G15     G16     PANSS_Total
    # 28    29  30  31  32  33  34  35  36  37  38

    if 'Study_E.csv' in fname:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6), delimiter=',', dtype=str )
    else:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6,39), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',', dtype=float64)



    data_a = empty(quant_data_a.shape[0], dtype=patient_visit_dt)
    #print(data_a.shape)


    data_a['study']  = cata_data_a[:, 0]
    data_a['country']  = cata_data_a[:, 1]
    data_a['txgroup']  = cata_data_a[:, 2]
    data_a['assesmentid'] = quant_data_a[:,3]
    data_a['patientid'] = quant_data_a[:,0]
    data_a['visitday'] = quant_data_a[:,4]


    data_a['xvalues'] = quant_data_a[:,5:35]
    data_a['panss'] = quant_data_a[:,35]
    if 'Study_E.csv' in fname:
        data_a['leadstatus'] =  0
    else:
        data_a['leadstatus'] = cata_data_a[:,3]
    return data_a

def trial():
    num_l = []
    for i in range(100):
        num_l.append(i)
    return num_l

def find_patients(data_a):
    d = {}
    for i in range(data_a.shape[0]):
        key = data_a[i]['patientid']
        if key in d:
            d[key].append(i)
        else:
            d[key] = [i]

    patient_d = {}
    for patientid, index_l in d.items():
        patient_a = data_a[index_l]
        index_v = patient_a['visitday'].argsort()
        patient_d[patientid] = patient_a[index_v]

    print(type(patient_d))
    return patient_d

def seperate_control_treatment(patient_d):
    control_d = {}
    treatment_d = {}

    for patient_a in patient_d.values():

        if patient_a[0]['txgroup'] == '"Control"':
            control_d[patient_a[0]['patientid']] = patient_a
        else:
            #print('test')
            treatment_d[patient_a[0]['patientid']] = patient_a

    return control_d, treatment_d


def plot_patient_data(data_l):
    print(type(data_l))
    print(len(data_l))

    fig, ax = plt.subplots()

    for patient_a in data_l:

        x_l = patient_a['visitday']
        y_l = patient_a['panss']
        ax.plot(x_l, y_l)
    plt.show()

def filter_patients(patient_d, day_limit=126):
    print(type(patient_d))
    print(patient_d)
    #delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] < day_limit]
    delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] - patient_a[0]['visitday'] < day_limit]
    for patientid in delete_patient_l:
        del patient_d[patientid]

def difference_in_fields(patient_d, field_name):
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1][field_name]
        dif2 = patient_a[0][field_name]
        dif = dif1-dif2
        #dif = patient_a[-1]['panss'] - patient_a[1]['panss']
        difference_values_l.append(dif)
    difference_values_a = array(difference_values_l)
    return difference_values_a

def difference_in_scores(patient_d):
    assert 0
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1]['panss']
        dif2 = patient_a[0]['panss']
        dif = dif1-dif2
        difference_values_l.append(dif)
    return difference_values_l

def difference_in_scores_stats(difference_values_l):
    mean_difference = mean(difference_values_l)
```

```
    standev_difference = std(difference_values_l)
    print(f'The mean difference is {mean_difference} and has a standard deviation of {standev_difference}')
    return mean_difference, standev_difference

def knn_pred(xtrain_a, ytrain_v, xtest_a, ytest_v):
    knn_model = KNeighborsClassifier(n_neighbors = 5)
    knn_model.fit(xtrain_a, ytrain_v)
    ytrainpred_v = knn_model.predict(xtrain_a)
    ypred_v = knn_model.predict(xtest_a)
    ypred_proba_a = knn_model.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]

def class_pred(clf, xtrain_a, ytrain_v, xtest_a, ytest_v):
    clf.fit(xtrain_a, ytrain_v)
    ytrainpred_v = clf.predict(xtrain_a)
    ypred_v = clf.predict(xtest_a)
    ypred_proba_a = clf.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]



    # m_v = ypred_v == ytest_v
    # print('match samples:', ypred_v[m_v])
    # print('mismatch samples:', list(zip(ypred_v[logical_not(m_v)], ytest_v[logical_not(m_v)])))

    print(f'The training accuracy is: {accuracy_train} ')
    print(f'The test accuracy is {accuracy_test}')
    return ypred_proba_a

def z_score_conver(data_a):
    train_data_a = data_a['xvalues']
    #print(train_data_a)
    mean_a = train_data_a.mean(axis=0)
    standev_a = train_data_a.std(axis=0)
    zscore = (train_data_a - mean_a)/standev_a
    #print(zscore)
    return zscore

def z_score_convert2(data_a, input_test_data_a):
    train_data_a = data_a['xvalues']
    test_data_a = input_test_data_a['xvalues']
    #print(train_data_a)
    train_mean_v = train_data_a.mean(axis=0)
    train_standev_v = train_data_a.std(axis=0)
    train_zscore_a = (train_data_a - train_mean_v)/train_standev_v
    test_zscore_a = (test_data_a - train_mean_v)/train_standev_v
    #print(zscore)
    return train_zscore_a, test_zscore_a

def time_shifter(patient_d):
    #adjusted_d = {}
    for patient_id, visitday in patient_d.items():
        time_zero = visitday[0]['visitday']
        visitday[0] = 0
        for other_days in visitday[1:]:
            other_days['visitday']-=time_zero



def desired_data0(patient_d):
    predicted_data_l = []
    for patient_id, array_values_a in patient_d.items():
        last_array = array_values_a[-1]
        output_data = last_array['panss']
        predicted_data_l.append((patient_id, output_data))
    submitted_data_a = array(predicted_data_l, output_data)
    return submitted_data_a


def desired_data1(patient_d, target_day = 129.5):
    predicted_data_l = []
    for patient_id, patient_a in patient_d.items():

        week = (patient_a[-1]['visitday']-patient_a[0]['visitday'])/7
        if week > 18:
            print('more days', patient_id, week)
        day_v = patient_a['visitday']
        day_v = day_v - day_v[0]
        panss_v = patient_a['panss']

        if target_day>day_v[-1] and patient_a.shape[0] > 1 and day_v[-1] != day_v[-2]:
            m = (panss_v[-1]-panss_v[-2])/(day_v[-1]-day_v[-2])
            panss = m*(target_day-day_v[-1]) + panss_v[-1]
        else:
            panss = interp(target_day, day_v, panss_v)


        predicted_data_l.append((patient_id, panss))
    return array(predicted_data_l)

def desired_data(patient_d, target_day = 129.5):
    predicted_data_l = []
    for patient_id, patient_a in patient_d.items():

        week = (patient_a[-1]['visitday']-patient_a[0]['visitday'])/7
        if week > 18:
            print('more days', patient_id, week)
        day_v = patient_a['visitday']
        day_v = day_v - day_v[0]
        panss_v = patient_a['panss']

        if target_day-day_v[-1]>50:
            panss = 62.5

        else:
            panss = interp(target_day, day_v, panss_v)


        predicted_data_l.append((patient_id, panss))
    return array(predicted_data_l)


def class_data_merge(test_data_a, maxproba_v):
    #print("see assesment shapes", test_data_a['assesmentid'].shape)
    class_proba_l = list(zip(test_data_a['assesmentid'],maxproba_v))
    class_proba_a = array (class_proba_l)

    return class_proba_a

def save_file(patient_data_a, fname):
    savetxt(fname, patient_data_a, delimiter=',', fmt='%d')

def main():
    data_a = hstack([data_loader(fname) for fname in sys.argv[1:]])
    train_data_a = hstack([data_loader(fname) for fname in sys.argv[1:-1]])
    test_data_a = data_loader(sys.argv[-1])

    m_v = logical_not(train_data_a['leadstatus'] == '"Passed"')
    train_data_a = train_data_a[m_v]
    m_v = logical_not(test_data_a['leadstatus'] == '"Passed"')
    test_data_a = test_data_a[m_v]

```

```
    # print("seeing train after load", train_data_a.shape)
    #print("seeing test after load", test_data_a.shape)

    patientid_v = data_a['patientid']
    upatientid_v = unique(patientid_v)
    #print('Patient id:', upatientid_v.shape[0])

    #print("just viewing", patientid_v)

    #print(da)
    patient_d = find_patients(test_data_a)

    #print('------------')

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("view control people", control_d)
    #print(type(control_d))
    #print("view treatment people", treatment_d)
    #print(type(control_d))
    #print('------------')

    control_patientdiff = difference_in_fields(control_d, 'panss')
    # plt.hist(control_patientdiff, bins = 10)
    # plt.xlabel('Newest Measurement - Oldest Measurement')
    # plt.ylabel('Frequency')
    # plt.title('Histogram of Differences in Control Group')
    # plt.show()
    control_patientdiffstats = difference_in_scores_stats(control_patientdiff)
    #print(type(control_patientdiff))

    treatment_patientdiff = difference_in_fields(treatment_d, 'panss')
    # plt.hist(treatment_patientdiff, bins = 10)
    # plt.xlabel('Newest Measurement - Oldest Measurement')
    # plt.ylabel('Frequency')
    # plt.title('Histogram of Differences in Treatment Group')
    # plt.show()
    treatment_patientdiffstats = difference_in_scores_stats(treatment_patientdiff)

    #print('------------')


    print(f'Control Patients: {control_patientdiffstats}')
    print(len(control d))
    print(f'Treatment Patients: {treatment_patientdiffstats}')
    print(len(treatment_d))

    print('------------')
    #print(patient_d)
    fname = "upload1.csv"
    upload_data_a = desired_data(patient_d)
    print(upload_data_a)
    save_file(upload_data_a, fname)
    print('------------')
    zscore_train_a, zscore_test_a = z_score_convert2(train_data_a, test_data_a)




    #ypredproba_a = knn_pred(zscore_train_a, train_data_a['leadstatus'], zscore_test_a, test_data_a['leadstatus'])
    #ypredproba_a = knn_pred(train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])

    #clf = KNeighborsClassifier(n_neighbors = 5)
    #clf = CategoricalNB(force_alpha=True)
    #clf = GaussianNB()
    #clf = tree.DecisionTreeClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, criterion='entropy')
    clf = RandomForestClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, random_state=0)
    ypredproba_a = class_pred(clf, train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])


    print(ypredproba a)
    print('------------')

    max_proba_v = ypredproba_a.max(axis=1)
    class_data_a = class_data_merge(test_data_a, max_proba_v)

    fname = "classupload1.csv"
    save_file(class_data_a, fname)
    raise SystemExit
    plot_patient_data(control_d)
    plot_patient_data(treatment_d)
    raise SystemExit

    # day_v = data_a['visitday']
    # # uday_v = unique(day_v)
    # week_v = uday_v/7
    # print(uday_v)
    # print('------------')
    # print('------------')
    # print(week_v)
    # print('------------')

    # print(f'data a:10 {data_a[10:]}')
    m_v = data_a['txgroup'] ==   '"Treatment'
    day_v = data a[m v]['visitday']
    print('treatment day', sorted(day_v))
    uday_v = unique(day_v)
    # week_v = uday_v//7
    # print(f'uday_v treatments: {uday_v}')
    # print(f'week_v treatments: {week_v}')
    # #week_v = int32(week_v)
    # print('week count:', list(zip(arange(week_v.shape[0]), bincount(week_v))))
    patient d = find patients(data_a)
    print('Before delete:', len(patient_d))
    filter_patients(patient_d, day_limit=105)
    print('After delete:', len(patient_d))

    #print(len(a))
    print('------------')
    #print(d)
    print(patient_d)

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("control", control_l)
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    #print("treatment", treatment_l)
    plot_patient_data(control_l)
    plot_patient data(treatment_l)
if __name__ == '__main__':
    main()
```

p3class.py

```
from cgi import test
from numpy import *
import sys
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
patient_visit_dt = dtype([('study','U10'),('country','U10'),('txgroup','U10'),('assesmentid', float64),('patientid', float64),('visitday', int32),('xvalues',float64,(31)),('panss',float64),

def data_loader(fname):
    #Study      Country PatientID       SiteID  RaterID AssessmentID    TxGroup VisitDay        P1      P2      P3      P4      P5      P6      P7      N1      N2      N3      N4      N5
    # 0     1       2           3       4       5               6       7           8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    # G7        G8      G9      G10     G11     G12     G13     G14     G15     G16     PANSS_Total
    # 28    29  30  31  32  33  34  35  36  37  38

    if 'Study_E.csv' in fname:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6), delimiter=',', dtype=str )
    else:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6,39), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',', dtype=float64)



    data_a = empty(quant_data_a.shape[0], dtype=patient_visit_dt)
    #print(data_a.shape)


    data_a['study']  = cata_data_a[:, 0]
    data_a['country']  = cata_data_a[:, 1]
    data_a['txgroup']  = cata_data_a[:, 2]
    data_a['assesmentid'] = quant_data_a[:,3]
    data_a['patientid'] = quant_data_a[:,0]
    data_a['visitday'] = quant_data_a[:,4]


    data_a['xvalues'] = quant_data_a[:,5:36]
    data_a['panss'] = quant_data_a[:,35]
    if 'Study_E.csv' in fname:
        data_a['leadstatus'] =  0
    else:
        data_a['leadstatus'] = cata_data_a[:,3]
    return data_a

def trial():
    num_l = []
    for i in range(100):
        num_l.append(i)
    return num_l

def find_patients(data_a):
    d = {}
    for i in range(data_a.shape[0]):
        key = data_a[i]['patientid']
        if key in d:
            d[key].append(i)
        else:
            d[key] = [i]

    patient_d = {}
    for patientid, index_l in d.items():
        patient_a = data_a[index_l]
        index_v = patient_a['visitday'].argsort()
        patient_d[patientid] = patient_a[index_v]

    print(type(patient_d))
    return patient_d

def seperate_control_treatment(patient_d):
    control_d = {}
    treatment_d = {}

    for patient_a in patient_d.values():

        if patient_a[0]['txgroup'] == '"Control"':
            control_d[patient_a[0]['patientid']] = patient_a
        else:
            #print('test')
            treatment_d[patient_a[0]['patientid']] = patient_a

    return control_d, treatment_d


def plot_patient_data(data_l):
    print(type(data_l))
    print(len(data_l))

    fig, ax = plt.subplots()

    for patient_a in data_l:

        x_l = patient_a['visitday']
        y_l = patient_a['panss']
        ax.plot(x_l, y_l)
    plt.show()

def filter_patients(patient_d, day_limit=126):
    print(type(patient_d))
    print(patient_d)
    #delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] < day_limit]
    delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] - patient_a[0]['visitday'] < day_limit]
    for patientid in delete_patient_l:
        del patient_d[patientid]

def difference_in_fields(patient_d, field_name):
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1][field_name]
        dif2 = patient_a[0][field_name]
        dif = dif1-dif2
        #dif = patient_a[-1]['panss'] - patient_a[1]['panss']
        difference_values_l.append(dif)
    difference_values_a = array(difference_values_l)
    return difference_values_a

def difference_in_scores(patient_d):
    assert 0
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1]['panss']
        dif2 = patient_a[0]['panss']
        dif = dif1-dif2
        difference_values_l.append(dif)
    return difference_values_l

def difference_in_scores_stats(difference_values_l):
    mean_difference = mean(difference_values_l)
```

```
    standev_difference = std(difference_values_l)
    print(f'The mean difference is {mean_difference} and has a standard deviation of {standev_difference}')
    return mean_difference, standev_difference

def knn_pred(xtrain_a, ytrain_v, xtest_a, ytest_v):
    knn_model = KNeighborsClassifier(n_neighbors = 5)
    knn_model.fit(xtrain_a, ytrain_v)
    ytrainpred_v = knn_model.predict(xtrain_a)
    ypred_v = knn_model.predict(xtest_a)
    ypred_proba_a = knn_model.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]

def class_pred(clf, xtrain_a, ytrain_v, xtest_a, ytest_v):
    clf.fit(xtrain_a, ytrain_v)
    ytrainpred_v = clf.predict(xtrain_a)
    ypred_v = clf.predict(xtest_a)
    ypred_proba_a = clf.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]



    # m_v = ypred_v == ytest_v
    # print('match samples:', ypred_v[m_v])
    # print('mismatch samples:', list(zip(ypred_v[logical_not(m_v)], ytest_v[logical_not(m_v)])))

    print(f'The training accuracy is: {accuracy_train} ')
    print(f'The test accuracy is {accuracy_test}')
    return ypred_proba_a

def z_score_conver(data_a):
    train_data_a = data_a['xvalues']
    #print(train_data_a)
    mean_a = train_data_a.mean(axis=0)
    standev_a = train_data_a.std(axis=0)
    zscore = (train_data_a - mean_a)/standev_a
    #print(zscore)
    return zscore

def z_score_convert2(data_a, input_test_data_a):
    train_data_a = data_a['xvalues']
    test_data_a = input_test_data_a['xvalues']
    #print(train_data_a)
    train_mean_v = train_data_a.mean(axis=0)
    train_standev_v = train_data_a.std(axis=0)
    train_zscore_a = (train_data_a - train_mean_v)/train_standev_v
    test_zscore_a = (test_data_a - train_mean_v)/train_standev_v
    #print(zscore)
    return train_zscore_a, test_zscore_a

def time_shifter(patient_d):
    #adjusted_d = {}
    for patient_id, visitday in patient_d.items():
        time_zero = visitday[0]['visitday']
        visitday[0] = 0
        for other_days in visitday[1:]:
            other_days['visitday']-=time_zero



def desired_data(patient_d):
    predicted_data_l = []
    for patient_id, array_values_a in patient_d.items():
        last_array = array_values_a[-1]
        output_data = last_array['panss']
        predicted_data_l.append((patient_id, output_data))
    submitted_data_a = array(predicted_data_l, output_data)
    return submitted_data_a

def class_data_merge(test_data_a, maxproba_v):
    #print("see assesment shapes", test_data_a['assesmentid'].shape)
    class_proba_l = list(zip(test_data_a['assesmentid'],maxproba_v))
    class_proba_a = array (class_proba_l)

    return class_proba_a

def save_file(patient_data_a, fname):
    savetxt(fname, patient_data_a, delimiter=',')

def main():
    data_a = hstack([data_loader(fname) for fname in sys.argv[1:]])
    train_data_a = hstack([data_loader(fname) for fname in sys.argv[1:-1]])
    test_data_a = data_loader(sys.argv[-1])

    m_v = logical_not(train_data_a['leadstatus'] == '"Passed"')
    train_data_a = train_data_a[m_v]
    m_v = logical_not(test_data_a['leadstatus'] == '"Passed"')
    test_data_a = test_data_a[m_v]


    # print("seeing train after load", train_data_a.shape)
    #print("seeing test after load", test_data_a.shape)

    patientid_v = data_a['patientid']
    upatientid_v = unique(patientid_v)
    #print('Patient id:', upatientid_v.shape[0])

    #print("just viewing", patientid_v)

    #print(da)
    patient_d = find_patients(data_a)

    #print('------------')

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("view control people", control_d)
    #print(type(control_d))
    #print("view treatment people", treatment_d)
    #print(type(control_d))
    #print('------------')

    control_patientdiff = difference_in_fields(control_d, 'panss')
    control_patientdiffstats = difference_in_scores_stats(control_patientdiff)
    #print(type(control_patientdiff))

    treatment_patientdiff = difference_in_fields(treatment_d, 'panss')
    treatment_patientdiffstats = difference_in_scores_stats(treatment_patientdiff)

    #print('------------')


    print(f'Control Patients: {control_patientdiffstats}')
    print(f'Treatment Patients: {treatment_patientdiffstats}')

    print('------------')
    #print(patient_d)
    fname = "upload1.csv"
    upload_data_a = desired_data(patient_d)
    print(upload_data_a)
    save_file(upload_data_a, fname)

    print('------------')
    zscore_train_a, zscore_test_a = z_score_convert2(train_data_a, test_data_a)
```

```
264
265
266
267
268     #ypredproba_a = knn_pred(zscore_train_a, train_data_a['leadstatus'], zscore_test_a, test_data_a['leadstatus'])
269     #ypredproba_a = knn_pred(train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])
270
271     #clf = KNeighborsClassifier(n_neighbors = 5)
272     #clf = CategoricalNB(force_alpha=True)
273     #clf = GaussianNB()
274     #clf = tree.DecisionTreeClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, criterion='entropy')
275     #clf = RandomForestClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, random_state=0)
276     clf = svm.SVC(probability=True)
277     ypredproba_a = class_pred(clf, train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])
278
279
280     print(ypredproba a)
281     print('------------')
282
283     max_proba_v = ypredproba_a.max(axis=1)
284     class_data_a = class_data_merge(test_data_a, max_proba_v)
285
286     fname = "classupload1.csv"
287     save_file(class_data_a, fname)
288
289     raise SystemExit
290
291     # day_v = data_a['visitday']
292     # # uday_v = unique(day_v)
293     # week_v = uday_v/7
294     # print(uday_v)
295     # print('------------')
296     # print('------------')
297     # print(week_v)
298     # print('------------')
299
300     # print(f'data_a:10 {data_a[10:]}')
301     m_v = data_a['txgroup'] ==   '"Treatment'
302     day_v = data a[m v]['visitday']
303     print('treatment day', sorted(day_v))
304     uday_v = unique(day_v)
305     # week_v = uday_v//7
306     # print(f'uday_v treatments: {uday_v}')
307     # print(f'week_v treatments: {week_v}')
308     # #week_v = int32(week_v)
309     # print('week count:', list(zip(arange(week_v.shape[0]), bincount(week_v))))
310     patient d = find patients(data_a)
311     print('Before delete:', len(patient_d))
312     filter_patients(patient_d, day_limit=105)
313     print('After delete:', len(patient_d))
314
315     #print(len(a))
316     print('------------')
317     #print(d)
318     print(patient_d)
319
320     control_d, treatment_d = seperate_control_treatment(patient_d)
321     #print("control", control_l)
322     print('------------')
323     print('------------')
324     print('------------')
325     print('------------')
326     print('------------')
327     #print("treatment", treatment_l)
328     #plot_patient_data(control_l)
329     #plot_patient_data(treatment_l)
330 if __name__ == '__main__':
331     main()
```

prob2.py

```
from cgi import test
from numpy import *
import sys
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import complete, fcluster
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist
from sklearn.model_selection import KFold

patient_visit_dt = dtype([('study','U10'),('country','U10'),('txgroup','U10'),('assesmentid', float64),('patientid', float64),('visitday', int32),('xvalues',float64,(31)),('panss',float64),

def data_loader(fname):
    #Study      Country PatientID       SiteID  RaterID AssessmentID    TxGroup VisitDay        P1      P2      P3      P4      P5      P6      P7      N1      N2      N3      N4      N5
    # 0     1     2         3       4       5             6       7          8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    # G7        G8      G9      G10     G11     G12     G13     G14     G15     G16     PANSS_Total
    # 28    29  30  31  32  33  34  35  36  37  38

    if 'Study_E.csv' in fname:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6), delimiter=',', dtype=str )
    else:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6,39), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',', dtype=float64)



    data_a = empty(quant_data_a.shape[0], dtype=patient_visit_dt)
    #print(data_a.shape)


    data_a['study']  = cata_data_a[:, 0]
    data_a['country']  = cata_data_a[:, 1]
    data_a['txgroup']  = cata_data_a[:, 2]
    data_a['assesmentid'] = quant_data_a[:,3]
    data_a['patientid'] = quant_data_a[:,0]
    data_a['visitday'] = quant_data_a[:,4]


    data_a['xvalues'] = quant_data_a[:,5:36]
    data_a['panss'] = quant_data_a[:,35]
    if 'Study_E.csv' in fname:
        data_a['leadstatus'] =  0
    else:
        data_a['leadstatus'] = cata_data_a[:,3]
    return data_a

def trial():
    num_l = []
    for i in range(100):
        num_l.append(i)
    return num_l

def find_patients(data_a):
    d = {}
    for i in range(data_a.shape[0]):
        key = data_a[i]['patientid']
        if key in d:
            d[key].append(i)
        else:
            d[key] = [i]

    patient_d = {}
    for patientid, index_l in d.items():
        patient_a = data_a[index_l]
        index_v = patient_a['visitday'].argsort()
        patient_d[patientid] = patient_a[index_v]

    print(type(patient_d))
    return patient_d

def seperate_control_treatment(patient_d):
    control_d = {}
    treatment_d = {}

    for patient_a in patient_d.values():

        if patient_a[0]['txgroup'] == '"Control"':
            control_d[patient_a[0]['patientid']] = patient_a
        else:
            #print('test')
            treatment_d[patient_a[0]['patientid']] = patient_a

    return control_d, treatment_d


def plot_patient_data(data_l):
    print(type(data_l))
    print(len(data_l))

    fig, ax = plt.subplots()

    for patient_a in data_l:

        x_l = patient_a['visitday']
        y_l = patient_a['panss']
        ax.plot(x_l, y_l)
    plt.show()

def filter_patients(patient_d, day_limit=126):
    print(type(patient_d))
    print(patient_d)
    #delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] < day_limit]
    delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] - patient_a[0]['visitday'] < day_limit]
    for patientid in delete_patient_l:
        del patient_d[patientid]

def difference_in_fields(patient_d, field_name):
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1][field_name]
        dif2 = patient_a[0][field_name]
        dif = dif1-dif2
        #dif = patient_a[-1]['panss'] - patient_a[1]['panss']
        difference_values_l.append(dif)
    difference_values_a = array(difference_values_l)
    return difference_values_a

def difference_in_scores(patient_d):
    assert 0
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1]['panss']
        dif2 = patient_a[0]['panss']
```

```
        dif = dif1-dif2
        difference_values_l.append(dif)
    return difference_values_l

def difference_in_scores_stats(difference_values_l):
    mean_difference = mean(difference_values_l)
    standev difference = std(difference_values_l)
    print(f'The mean difference is {mean_difference} and has a standard deviation of {standev_difference}')
    return mean_difference, standev_difference

def knn_pred(xtrain_a, ytrain_v, xtest_a, ytest_v):
    knn_model = KNeighborsClassifier(n_neighbors = 5)
    knn_model.fit(xtrain_a, ytrain_v)
    ytrainpred_v = knn_model.predict(xtrain_a)
    ypred_v = knn_model.predict(xtest_a)
    ypred_proba_a = knn_model.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]

def class_pred(clf, xtrain_a, ytrain_v, xtest_a, ytest_v):
    clf.fit(xtrain_a, ytrain_v)
    ytrainpred_v = clf.predict(xtrain_a)
    ypred_v = clf.predict(xtest_a)
    ypred_proba_a = clf.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]



    # m_v = ypred_v == ytest_v
    # print('match samples:', ypred_v[m_v])
    # print('mismatch samples:', list(zip(ypred_v[logical_not(m_v)], ytest_v[logical_not(m_v)])))

    print(f'The training accuracy is: {accuracy_train} ')
    print(f'The test accuracy is {accuracy_test}')
    return ypred_proba_a

def z_score_conver(data_a):
    train_data_a = data_a['xvalues']
    #print(train_data_a)
    mean_a = train_data_a.mean(axis=0)
    standev_a = train_data_a.std(axis=0)
    zscore = (train_data_a - mean_a)/standev_a
    #print(zscore)
    return zscore

def z_score_convert2(data_a, input test_data_a):
    train_data_a = data_a['xvalues']
    test_data_a = input_test_data_a['xvalues']
    #print(train_data_a)
    train_mean_v = train_data_a.mean(axis=0)
    train_standev_v = train_data_a.std(axis=0)
    train_zscore_a = (train_data_a - train_mean_v)/train_standev_v
    test_zscore_a = (test_data_a - train_mean_v)/train_standev_v
    #print(zscore)
    return train_zscore_a, test_zscore_a

def time_shifter(patient_d):
    #adjusted_d = {}
    for patient_id, visitday in patient d.items():
        time_zero = visitday[0]['visitday']
        visitday[0] = 0
        for other_days in visitday[1:]:
            other_days['visitday']-=time_zero



def desired_data(patient_d):
    predicted_data_l = []
    for patient_id, array_values_a in patient_d.items():
        last_array = array_values a[-1]
        output_data = last_array['panss']
        predicted_data_l.append((patient_id, output_data))
    submitted_data_a = array(predicted_data_l, output_data)
    return submitted_data_a

def class_data_merge(test_data_a, maxproba_v):
    #print("see assesment shapes", test_data_a['assesmentid'].shape)
    class_proba_l = list(zip(test_data_a['assesmentid'],maxproba_v))
    class_proba_a = array (class_proba_l)

    return class_proba_a

def get_first_day(patient_d):
    return array([patient_a[0]['xvalues'] for patient_a in patient_d.values()])


def k_means_clustering(x, n_clusters):
    #best_k = max(range(2, 11), key=lambda k: np.mean(cross_val_score(KMeans(n_clusters=k), cv=10)))
    kf = KFold(n_splits=10, shuffle=True, random_state=42)
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(x)
    cluster_labels = kmeans.labels_
    cluster_centers = kmeans.cluster_centers_
    plt.scatter(x[:, 0], x[:, 7],  marker='o')
    print("x", x)
    #In this case x:,0 and X:,7 are the p1 and n1 values
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='X')
    plt.show()
    return cluster_labels, cluster_centers
def dendo_construct(x_a):
    linkage_matrix = hierarchy.linkage(x_a, method='ward')

    plt.figure(figsize=(10, 6))
    dendrogram = hierarchy.dendrogram(linkage_matrix)
    plt.show()
def save_file(patient_data_a, fname):
    savetxt(fname, patient_data_a, delimiter=',')

def main():
    data_a = hstack([data_loader(fname) for fname in sys.argv[1:]])
    train_data_a = hstack([data_loader(fname) for fname in sys.argv[1:-1]])
    test_data_a = data_loader(sys.argv[-1])

    m_v = logical_not(train_data_a['leadstatus'] == '"Passed"')
    train_data_a = train_data_a[m_v]
    m_v = logical_not(test_data_a['leadstatus'] == '"Passed"')
    test_data_a = test_data_a[m_v]


    # print("seeing train after load", train_data_a.shape)
    #print("seeing test after load", test_data_a.shape)

    patientid_v = data_a['patientid']
    upatientid_v = unique(patientid_v)
    #print('Patient id:', upatientid_v.shape[0])

    #print("just viewing", patientid_v)

    #print(da)
    patient_d = find_patients(data_a)

    #print('------------')

```

```
    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("view control people", control_d)
    #print(type(control_d))
    #print("view treatment people", treatment_d)
    #print(type(control_d))
    #print('------------')

    control_patientdiff = difference_in_fields(control_d, 'panss')
    control_patientdiffstats = difference_in_scores_stats(control_patientdiff)
    #print(type(control_patientdiff))

    treatment_patientdiff = difference_in_fields(treatment_d, 'panss')
    treatment_patientdiffstats = difference_in_scores_stats(treatment_patientdiff)

    #print('------------')


    print(f'Control Patients: {control_patientdiffstats}')
    print(f'Treatment Patients: {treatment_patientdiffstats}')

    print('------------')
    #print(patient_d)
    fname = "upload1.csv"
    upload_data_a = desired_data(patient_d)
    print(upload_data_a)
    save_file(upload_data_a, fname)

    print('------------')
    zscore_train_a, zscore_test_a = z_score_convert2(train_data_a, test_data_a)
    first_x_a = get_first_day(patient_d)
    cluster labels, cluster_centers = k_means_clustering(first_x_a,3)
    print("cluster labels:", cluster_labels)
    print("cluster centers:", cluster_centers)

    #ypredproba_a = knn_pred(zscore_train_a, train_data_a['leadstatus'], zscore_test_a, test_data_a['leadstatus'])
    #ypredproba_a = knn_pred(train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])

    #clf = KNeighborsClassifier(n_neighbors = 5)
    #clf = CategoricalNB(force_alpha=True)
    #clf = GaussianNB()
    #clf = tree.DecisionTreeClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, criterion='entropy')
    #clf = RandomForestClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, random_state=0)
    clf = svm.SVC(probability=True)
    ypredproba_a = class_pred(clf, train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])


    print(ypredproba a)
    print('------------')

    max_proba_v = ypredproba_a.max(axis=1)
    class_data_a = class_data_merge(test_data_a, max_proba_v)

    fname = "classupload1.csv"
    save_file(class_data_a, fname)

    dendo_construct(get_first_day(patient_d))

    raise SystemExit

    # day_v = data_a['visitday']
    # # uday_v = unique(day_v)
    # week_v = uday_v/7
    # print(uday_v)
    # print('------------')
    # print('------------')
    # print(week_v)
    # print('------------')

    # print(f'data_a:10 {data_a[10:]}')
    m_v = data_a['txgroup'] ==   '"Treatment'
    day_v = data a[m v]['visitday']
    print('treatment day', sorted(day_v))
    uday_v = unique(day_v)
    # week_v = uday_v//7
    # print(f'uday_v treatments: {uday_v}')
    # print(f'week_v treatments: {week_v}')
    # #week_v = int32(week_v)
    # print('week count:', list(zip(arange(week_v.shape[0]), bincount(week_v))))
    patient d = find patients(data_a)
    print('Before delete:', len(patient_d))
    filter_patients(patient_d, day_limit=105)
    print('After delete:', len(patient_d))

    #print(len(a))
    print('------------')
    #print(d)
    print(patient_d)

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("control", control_l)
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    #print("treatment", treatment_l)
    #plot_patient_data(control_l)
    #plot_patient data(treatment_l)
if __name__ == '__main__':
    main()
```

realp3.py

```
from cgi import test
from numpy import *
import sys
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

patient_visit_dt = dtype([('study','U10'),('country','U10'),('txgroup','U10'),('assesmentid', float64),('patientid', float64),('visitday', int32),('xvalues',float64,(30)),('panss',float64),

def data_loader(fname):
    #Study      Country PatientID       SiteID  RaterID AssessmentID    TxGroup VisitDay        P1      P2      P3      P4      P5      P6      P7      N1      N2      N3      N4      N5
    # 0     1       2           3       4       5               6       7           8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
    # G7        G8      G9      G10     G11     G12     G13     G14     G15     G16     PANSS_Total
    # 28    29  30  31  32  33  34  35  36  37  38

    if 'Study_E.csv' in fname:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6), delimiter=',', dtype=str )
    else:
        cata_data_a = loadtxt(fname,skiprows=1, usecols=(0,1,6,39), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38), delimiter=',', dtype=float64)



    data_a = empty(quant_data_a.shape[0], dtype=patient_visit_dt)
    #print(data_a.shape)


    data_a['study']  = cata_data_a[:, 0]
    data_a['country']  = cata_data_a[:, 1]
    data_a['txgroup']  = cata_data_a[:, 2]
    data_a['assesmentid'] = quant_data_a[:,3]
    data_a['patientid'] = quant_data_a[:,0]
    data_a['visitday'] = quant_data_a[:,4]


    data_a['xvalues'] = quant_data_a[:,5:35]
    data_a['panss'] = quant_data_a[:,35]
    if 'Study_E.csv' in fname:
        data_a['leadstatus'] =  0
    else:
        data_a['leadstatus'] = cata_data_a[:,3]
    return data_a

def trial():
    num_l = []
    for i in range(100):
        num_l.append(i)
    return num_l

def find_patients(data_a):
    d = {}
    for i in range(data_a.shape[0]):
        key = data_a[i]['patientid']
        if key in d:
            d[key].append(i)
        else:
            d[key] = [i]

    patient_d = {}
    for patientid, index_l in d.items():
        patient_a = data_a[index_l]
        index_v = patient_a['visitday'].argsort()
        patient_d[patientid] = patient_a[index_v]

    print(type(patient_d))
    return patient_d

def seperate_control_treatment(patient_d):
    control_d = {}
    treatment_d = {}

    for patient_a in patient_d.values():

        if patient_a[0]['txgroup'] == '"Control"':
            control_d[patient_a[0]['patientid']] = patient_a
        else:
            #print('test')
            treatment_d[patient_a[0]['patientid']] = patient_a

    return control_d, treatment_d


def plot_patient_data(data_l):
    print(type(data_l))
    print(len(data_l))

    fig, ax = plt.subplots()

    for patient_a in data_l:

        x_l = patient_a['visitday']
        y_l = patient_a['panss']
        ax.plot(x_l, y_l)
    plt.show()

def filter_patients(patient_d, day_limit=126):
    print(type(patient_d))
    print(patient_d)
    #delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] < day_limit]
    delete_patient_l = [patientid for patientid, patient_a in patient_d.items() if patient_a[-1]['visitday'] - patient_a[0]['visitday'] < day_limit]
    for patientid in delete_patient_l:
        del patient_d[patientid]

def difference_in_fields(patient_d, field_name):
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1][field_name]
        dif2 = patient_a[0][field_name]
        dif = dif1-dif2
        #dif = patient_a[-1]['panss'] - patient_a[1]['panss']
        difference_values_l.append(dif)
    difference_values_a = array(difference_values_l)
    return difference_values_a

def difference_in_scores(patient_d):
    assert 0
    difference_values_l = []
    for patient_a in patient_d.values():
        dif1 = patient_a[-1]['panss']
        dif2 = patient_a[0]['panss']
        dif = dif1-dif2
        difference_values_l.append(dif)
    return difference_values_l

def difference_in_scores_stats(difference_values_l):
    mean_difference = mean(difference_values_l)
```

```
    standev difference = std(difference_values_l)
    print(f'The mean difference is {mean_difference} and has a standard deviation of {standev_difference}')
    return mean_difference, standev_difference

def knn_pred(xtrain_a, ytrain_v, xtest_a, ytest_v):
    knn_model = KNeighborsClassifier(n_neighbors = 5)
    knn_model.fit(xtrain_a, ytrain_v)
    ytrainpred_v = knn_model.predict(xtrain_a)
    ypred_v = knn_model.predict(xtest_a)
    ypred_proba_a = knn_model.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]

def class_pred(clf, xtrain_a, ytrain_v, xtest_a, ytest_v):
    clf.fit(xtrain_a, ytrain_v)
    ytrainpred_v = clf.predict(xtrain_a)
    ypred_v = clf.predict(xtest_a)
    ypred_proba_a = clf.predict_proba(xtest_a)
    accuracy_train = ((ytrain_v==ytrainpred_v).sum())/ytrainpred_v.shape[0]
    accuracy_test = ((ytest_v==ypred_v).sum())/ytest_v.shape[0]



    # m_v = ypred_v == ytest_v
    # print('match samples:', ypred_v[m_v])
    # print('mismatch samples:', list(zip(ypred_v[logical_not(m_v)], ytest_v[logical_not(m_v)])))

    print(f'The training accuracy is: {accuracy_train} ')
    print(f'The test accuracy is {accuracy_test}')
    return ypred_proba_a

def z_score_conver(data_a):
    train_data_a = data_a['xvalues']
    #print(train_data_a)
    mean_a = train_data_a.mean(axis=0)
    standev_a = train_data_a.std(axis=0)
    zscore = (train_data_a - mean_a)/standev_a
    #print(zscore)
    return zscore

def z_score_convert2(data_a, input_test_data_a):
    train_data_a = data_a['xvalues']
    test_data_a = input_test_data_a['xvalues']
    #print(train_data_a)
    train_mean_v = train_data_a.mean(axis=0)
    train_standev_v = train_data_a.std(axis=0)
    train_zscore_a = (train_data_a - train_mean_v)/train_standev_v
    test_zscore_a = (test_data_a - train_mean_v)/train_standev_v
    #print(zscore)
    return train_zscore_a, test_zscore_a

def time_shifter(patient_d):
    #adjusted_d = {}
    for patient_id, visitday in patient_d.items():
        time_zero = visitday[0]['visitday']
        visitday[0] = 0
        for other_days in visitday[1:]:
            other_days['visitday']-=time_zero



def desired_data(patient_d):
    predicted_data_l = []
    for patient_id, array_values_a in patient_d.items():
        last_array = array_values_a[-1]
        output_data = last_array['panss']
        predicted_data_l.append((patient_id, output_data))
    submitted_data_a = array(predicted_data_l, output_data)
    return submitted_data_a

def class_data_merge(test_data_a, maxproba_v):
    #print("see assesment shapes", test_data_a['assesmentid'].shape)
    class_proba_l = list(zip(test_data_a['assesmentid'],maxproba_v))
    class_proba_a = array (class_proba_l)

    return class_proba_a

def save_file(patient_data_a, fname):
    savetxt(fname, patient_data_a, delimiter=',')

def main():
    data_a = hstack([data_loader(fname) for fname in sys.argv[1:]])
    train_data_a = hstack([data_loader(fname) for fname in sys.argv[1:-1]])
    test_data_a = data_loader(sys.argv[-1])

    m_v = logical_not(train_data_a['leadstatus'] == '"Passed"')
    train_data_a = train_data_a[m_v]
    m_v = logical_not(test_data_a['leadstatus'] == '"Passed"')
    test_data_a = test_data_a[m_v]


    # print("seeing train after load", train_data_a.shape)
    #print("seeing test after load", test_data_a.shape)

    patientid_v = data_a['patientid']
    upatientid_v = unique(patientid_v)
    #print('Patient id:', upatientid_v.shape[0])

    #print("just viewing", patientid_v)

    #print(da)
    patient_d = find_patients(data_a)

    #print('------------')

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("view control people", control_d)
    #print(type(control_d))
    #print("view treatment people", treatment_d)
    #print(type(control_d))
    #print('------------')

    control_patientdiff = difference_in_fields(control_d, 'panss')
    plt.hist(control_patientdiff, bins = 10)
    plt.xlabel('Newest Measurement - Oldest Measurement')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences in Control Group')
    plt.show()
    control_patientdiffstats = difference_in_scores_stats(control_patientdiff)
    #print(type(control_patientdiff))

    treatment_patientdiff = difference_in_fields(treatment_d, 'panss')
    plt.hist(treatment_patientdiff, bins = 10)
    plt.xlabel('Newest Measurement - Oldest Measurement')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences in Treatment Group')
    plt.show()
    treatment_patientdiffstats = difference_in_scores_stats(treatment_patientdiff)

    #print('------------')


    print(f'Control Patients: {control_patientdiffstats}')
    print(len(control_d))
```

```
    print(f'Treatment Patients: {treatment_patientdiffstats}')
    print(len(treatment_d))

    print('------------')
    #print(patient_d)
    fname = "upload1.csv"
    upload_data_a = desired_data(patient_d)
    print(upload_data_a)
    save_file(upload data_a, fname)
    print('------------')
    zscore_train_a, zscore_test_a = z_score_convert2(train_data_a, test_data_a)




    #ypredproba_a = knn_pred(zscore_train_a, train_data_a['leadstatus'], zscore_test_a, test_data_a['leadstatus'])
    #ypredproba_a = knn_pred(train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])

    #clf = KNeighborsClassifier(n_neighbors = 5)
    #clf = CategoricalNB(force_alpha=True)
    #clf = GaussianNB()
    #clf = tree.DecisionTreeClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, criterion='entropy')
    clf = RandomForestClassifier(max_depth=8, min_samples_leaf=10, ccp_alpha=0.005, random_state=0)
    ypredproba_a = class_pred(clf, train_data_a['xvalues'], train_data_a['leadstatus'], test_data_a['xvalues'], test_data_a['leadstatus'])


    print(ypredproba_a)
    print('------------')

    max_proba_v = ypredproba_a.max(axis=1)
    class_data_a = class_data_merge(test_data_a, max_proba_v)

    fname = "classupload1.csv"
    save_file(class_data_a, fname)
    raise SystemExit
    plot_patient_data(control_d)
    plot_patient_data(treatment_d)
    raise SystemExit

    # day_v = data_a['visitday']
    # # uday_v = unique(day_v)
    # week_v = uday_v/7
    # print(uday_v)
    # print('------------')
    # print('------------')
    # print(week_v)
    # print('------------')

    # print(f'data_a:10 {data_a[10:]}')
    m_v = data_a['txgroup'] ==   '"Treatment'
    day_v = data_a[m_v]['visitday']
    print('treatment day', sorted(day_v))
    uday_v = unique(day_v)
    # week_v = uday_v//7
    # print(f'uday_v treatments: {uday_v}')
    # print(f'week_v treatments: {week_v}')
    # #week_v = int32(week_v)
    # print('week count:', list(zip(arange(week_v.shape[0]), bincount(week_v))))
    patient_d = find_patients(data_a)
    print('Before delete:', len(patient_d))
    filter_patients(patient_d, day_limit=105)
    print('After delete:', len(patient_d))

    #print(len(a))
    print('------------')
    #print(d)
    print(patient_d)

    control_d, treatment_d = seperate_control_treatment(patient_d)
    #print("control", control_l)
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    print('------------')
    #print("treatment", treatment_l)
    plot_patient_data(control_l)
    plot_patient_data(treatment_l)
if __name__ == '__main__':
    main()
```