Data Mining: Individual Project II Report

Q1. Prediction modeling with multiple linear regression.

a. Fit a multiple linear regression model to the depression score (DS) as a function of tag_id, step, and battery_low. Write the equation for predicting the depression score from the predictors in the model. (Note: You need to convert the categorical tag_id into dummy variables to use in regression models.

```
import pandas as pd
import numpy as np

df = pd.read_csv('device_uplink.csv')
df = df.dropna(axis=0)
df = df.drop(df[df['owner_id']==-1].index)
df = df.drop(df[df['step']==-9999].index)
df['client_time'] = pd.to_datetime(df.client_time)
df = df.drop(df[df['client_time'].dt.year == 1970].index)

unknown = [478,502,506,514,524,543,545,551,645,669,670,676]
for i in unknown:
    df = df.drop(df[df['owner_id'] == i].index)

df = df.drop(df[df['tag_id'] == 0].index)
df = df.drop(df[df['tag_id'] == 19].index)
df = df.drop(df[df['tag_id'] == 20].index)
df = df.drop(['Unnamed: 0','uplink_id'],axis=1)
user = pd.read_csv('user_information.csv')
user = user.drop(8)


a = []
b = []
for i in df['owner_id']:
    a.extend(user[user['user_id'] == i].depression_score.values.tolist())
    continue

for i in df['owner_id']:
    b.extend(user[user['user_id'] == i].depression_class.values.tolist())
    continue

df.insert(7,"depression",a,True)
df.insert(8,"depression_c",b,True)


df.head()
```

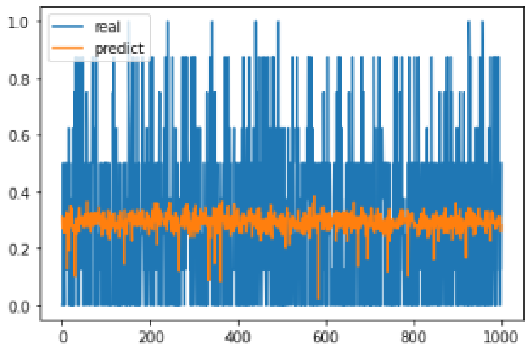
	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low	depression	depression_c
55	230	2019-12-02 10:58:10	8.0	421	99	1	0	0.125	Normal
56	230	2019-12-02 11:08:48	3.0	441	98	0	0	0.125	Normal
59	230	2019-12-02 11:11:43	10.0	472	98	0	0	0.125	Normal
60	230	2019-12-02 11:13:41	10.0	480	98	0	0	0.125	Normal
61	230	2019-12-02 11:17:09	4.0	489	97	0	0	0.125	Normal

The user's depression score and depression class columns were added to the pre-processed 'uplink_device.csv' in project 1. The value of the added column was taken from 'user_information.csv' by referring to the value of 'owner_id'. The 'user_information.csv' had two values for the same user, so I deleted one.

```
new = df[['tag_id','step','battery_low','depression']].copy()
new = new.astype({'tag_id':'int'})


temp = pd.get_dummies(new['tag_id'],prefix='tag')
new = new.drop(['tag_id'],axis=1)
new = pd.concat([new,temp],axis=1)

new.corr()
predictors = ['step','battery_low','tag_1','tag_2','tag_3','tag_4','tag_5','tag_6','tag_7','tag_8','tag_9','tag_10','tag_11','tag_12']
outcome = 'depression'
from sklearn.model_selection import train_test_split
x = new[predictors]
y = new[outcome]

train_x,valid_x,train_y,valid_y = train_test_split(x,y,test_size= 0.4)

from sklearn.linear_model import LinearRegression
df_lm = LinearRegression()
df_lm.fit(train_x,train_y)



df_lm_pred = df_lm.predict(valid_x)

result = pd.DataFrame({'predicted':df_lm_pred,'actual':valid_y,'residual':valid_y-df_lm_pred})

regressionSummary(valid_y,df_lm_pred)

import matplotlib.pyplot as plt


plt.plot(valid_y.values[:1000], label="real")
plt.plot(df_lm_pred[:1000], label="predict")
plt.legend()
```

After that, with the code above, I created a new data frame called 'new', which collected only the necessary information for multiple linear regression. After changing the column to a dummy variable and making a model, the test result with validation set was visualized with regressionSummary and graph. To calculate the depression score of a particular user, I decided to use the average value of the predicted value from the tagging information of each user.

```
Regression statistics

               Mean Error (ME) : 0.0012
Root Mean Squared Error (RMSE) : 0.2822
     Mean Absolute Error (MAE) : 0.2480

<matplotlib.legend.Legend at 0x24e13ddbdc0>
```



When looking at the results, there is an error of 0.24 on average. As can be seen from the graph, there is a large error between real value and prediction. I thought this was a problem of data bias. To solve this problem, I tried the methods such as oversampling techniques and reducing the number of data itself, but there was no significant difference. I think it means that the shape of the data I used is not suitable to use multiple linear regression.

b. Using the estimated regression model, what depression score is predicted for user 495 and 496? What is the prediction error?

```
p_495 = pd.read_csv('data_495.csv')
p_495 = p_495.drop(p_495[p_495['tag_0']==1].index)
p_495 = p_495.drop(['Unnamed: 0','client_time','is_charge','tag_battery_low','tag_0'],axis=1)
p_495_pred = df_lm.predict(p_495)
answer= 0

for i in p_495_pred:
    answer+=i
answer = answer/len(p_495_pred)
print("Predict:",answer)
print("Error:",0.25-answer)
```

```
Predict: 0.31216752813099063
Error: -0.062167528130990635
```

```
p_496 = pd.read_csv('data_496.csv')
p_496 = p_496.drop(p_496[p_496['tag_0']==1].index)

p_496 = p_496.drop(['Unnamed: 0','client_time','is_charge','tag_battery_low','tag_0'],axis=1)
p_496_pred = df_lm.predict(p_496)
answer= 0
for i in p_496_pred:
    answer+=i
answer = answer/len(p_496_pred)
print("Predict:",answer)
print("Error:",0.75-answer)
```

```
Predict: 0.30975755300990165
Error: 0.44024244699009835
```

I used the above-mentioned method to predict the user's depression score. After predicting the depression score of each row using the value of the user's tagging information and take the average of the calculated values as the user's final depression score. There was a relatively small error for user 495, but user 496 had a large error.

c. Use stepwise regression with the three options (backward, forward, both) to reduce the remaining predictors as follows: Run stepwise on the training set. Choose the top model from each stepwise run. Then use each of these models separately to predict the validation set. Compare RMSE, MAPE, and mean error, as well as lift charts. Finally, describe the best model.

```
from dmba import stepwise_selection
from dmba import forward_selection
from dmba import backward_elimination
from dmba import AIC_score


train_x,valid_x,train_y,valid_y = train_test_split(x,y,test_size= 0.4)


def train_model(variables):
    if len(variables) == 0:
        return None
    model = LinearRegression()
    model.fit(train_x[list(variables)],train_y)
    return model

def score_model(model,variables):
    if len(variables) == 0:
        return AIC_score(train_y,[train_y.mean()] * len(train_y),model,df=1)
    return AIC_score(train_y,model.predict(train_x[variables]),model)


forward_model, forward_variables = forward_selection(train_x.columns,train_model,score_model,verbose=True)

backward_model, backward_variables = backward_elimination(train_x.columns,train_model,score_model,verbose=True)

stepwise_model, stepwise_variables = stepwise_selection(train_x.columns,train_model,score_model,verbose=True)


from dmba import regressionSummary

print("FORWARD")
regressionSummary(valid_y,forward_model.predict(valid_x[forward_variables]))
print(forward_variables)
print("BACKWARD")
regressionSummary(valid_y,backward_model.predict(valid_x[backward_variables]))
print(backward_variables)
print("STEPWISE")
regressionSummary(valid_y,stepwise_model.predict(valid_x[stepwise_variables]))
print(stepwise_variables)
```

The above code is written by referring to the lecture slide. Using the same training set, the model was written in three ways: forward, backward, and stepwise. and I evaluated them using the validation set.

```
FORWARD

Regression statistics

               Mean Error (ME) : 0.0030
Root Mean Squared Error (RMSE) : 0.2830
     Mean Absolute Error (MAE) : 0.2482
['step', 'tag_7', 'tag_8', 'battery_low', 'tag_10', 'tag_1', 'tag_11', 'tag_3', 'tag_5', 'tag_12', 'tag_6', 'tag_4', 'tag_2']

BACKWARD

Regression statistics

               Mean Error (ME) : 0.0030
Root Mean Squared Error (RMSE) : 0.2830
     Mean Absolute Error (MAE) : 0.2482
['step', 'battery_low', 'tag_2', 'tag_3', 'tag_5', 'tag_6', 'tag_7', 'tag_8', 'tag_9', 'tag_11', 'tag_12']

STEPWISE

Regression statistics

               Mean Error (ME) : 0.0030
Root Mean Squared Error (RMSE) : 0.2830
     Mean Absolute Error (MAE) : 0.2482
['step', 'tag_7', 'tag_8', 'battery_low', 'tag_10', 'tag_1', 'tag_11', 'tag_3', 'tag_5', 'tag_12', 'tag_6', 'tag_4', 'tag_2']
```

Since MAPE can diverge when the result value is less than 1, so I used MAE. All three methods show the same statistical result value. However, backward elimination is the best model since it had the least number of variables. (Simple is better) I did not add the lift chart, since I already could find the best model with the above results.

Q2. Classification modeling with naïve Bayes classifier

a. Run a naïve Bayes classifier on the training set with the relevant predictors (and the depression class (DC) as the response). Note that all predictors should be categorical. Show the confusion matrix

```
user = pd.read_csv('user_information.csv')
diabetes = [0]*53
diabetes[0],diabetes[6],diabetes[27], diabetes[29],diabetes[36],diabetes[39],diabetes[50]= 1,1,1,1,1,1,1
diabetes = pd.Series(diabetes)
user = pd.concat([user,diabetes],axis=1)
user = user.rename(columns={0:'diabetes'})

high_pressure = [0]*53
high_pressure[0],high_pressure[4],high_pressure[14],high_pressure[16],high_pressure[19],high_pressure[27] = 1,1,1,1,1,1
high_pressure[29], high_pressure[36],high_pressure[37],high_pressure[38], high_pressure[39],high_pressure[41]= 1,1,1,1,1,1
high_pressure[42],high_pressure[48],high_pressure[50] = 1,1,1
high_pressure = pd.Series(high_pressure)
user = pd.concat([user,high_pressure],axis=1)
user = user.rename(columns={0:'high_pressure'})

blood_vessel = [0]*53
blood_vessel[6],blood_vessel[14],blood_vessel[16],blood_vessel[28],blood_vessel[30],blood_vessel[34] = 1,1,1,1,1,1
blood_vessel[35],blood_vessel[38] = 1,1
blood_vessel = pd.Series(blood_vessel)
user = pd.concat([user,blood_vessel],axis=1)
user = user.rename(columns={0:'blood_vessel'})

heart = [0]*53
heart[2],heart[18] ,heart[29],heart[38],heart[39],heart[42],heart[46]= 1,1,1,1,1,1,1
heart = pd.Series(heart)
user = pd.concat([user,heart],axis=1)
user = user.rename(columns={0:'heart'})

joint = [0]*53
joint[4],joint[11],joint[19],joint[27],joint[42] = 1,1,1,1,1
joint = pd.Series(joint)
user = pd.concat([user,joint],axis=1)
user = user.rename(columns={0:'joint'})

etc = [0]*53
etc[0],etc[6],etc[9],etc[10],etc[15],etc[17],etc[31],etc[37],etc[38],etc[43],etc[48] =1,1,1,1,1,1,1,1,1,1,1
etc = pd.Series(etc)
user = user.drop('etc',axis=1)
user = pd.concat([user,etc],axis=1)
user = user.rename(columns={0:'etc'})

user = user.drop(8)
user.head()
```

First, I used the 'user_information.csv' to classify the results. To make 'etc' column into category variables, I divided them into several categories: diabetes, high_pressure, blood_vessel, heart, joint, etc.

	user_id	birth year	age	sex	depression_score	depression_class	diabetes	high_pressure	blood_vessel	heart	joint	etc
0	519	1934	88	F	0.500	Moderate	1	1	0	0	0	1
1	520	1934	88	F	0.375	Mild	0	0	0	0	0	0
2	580	1935	87	F	0.125	Normal	0	0	0	1	0	0
3	495	1937	85	F	0.250	Normal	0	0	0	0	0	0
4	486	1937	85	F	0.500	Moderate	0	1	0	0	1	0

This is the shape of the pre-processed 'user_information.csv'.

```
steps_avg= [0]*52
count=0
for i in user['user_id']:
    temp=df[df['owner_id'] == i].copy()
    temp2 = temp.groupby(temp.client_time.dt.day).max()

    steps = 0
    for j in temp2['step']:
        steps += j
    steps = steps / len(temp2['step'])
    steps_avg[count] = steps
    count+=1

count= 0
steps_class = [""]*52
for i in range(len(steps_avg)):
    if steps_avg[i] <3000:
        steps_class[i] = "None"
    elif steps_avg[i] <6000:
        steps_class[i] = "small"
    elif steps_avg[i] <10000:
        steps_class[i] = "medium"
    else:
        steps_class[i] = "large"
steps_final = list(steps_class[:8])

steps_final.append("a")
steps_final.extend(steps_class[8:])

steps_final = pd.Series(steps_final)
user = pd.concat([user,steps_final],axis=1)
user = user.rename(columns={0:'steps'})
user = user.drop(8)

user.head()
```

	user_id	birth year	age	sex	depression_score	depression_class	diabetes	high_pressure	blood_vessel	heart	joint	etc	steps
0	519.0	1934.0	88.0	F	0.500	Moderate	1.0	1.0	0.0	0.0	0.0	1.0	small
1	520.0	1934.0	88.0	F	0.375	Mild	0.0	0.0	0.0	0.0	0.0	0.0	medium
2	580.0	1935.0	87.0	F	0.125	Normal	0.0	0.0	0.0	1.0	0.0	0.0	small
3	495.0	1937.0	85.0	F	0.250	Normal	0.0	0.0	0.0	0.0	0.0	0.0	small
4	486.0	1937.0	85.0	F	0.500	Moderate	0.0	1.0	0.0	0.0	1.0	0.0	large

Thereafter, the average number of steps per day of each user was measured and divided into four categories (none if average steps <3000, small if 3000<average steps<6000, medium if 6000<average steps<10000, and large if average steps>10000). Thereafter, the steps column has been successfully added to the ‘users_information.csv’.

```
from sklearn.naive_bayes import MultinomialNB
from dmba import classificationSummary, gainsChart

chart = user.copy()
chart = chart.reset_index()
arr = []
for i in chart['age']:
    if i<80:
        arr.append("low")
    else:
        arr.append("high")

arr = pd.Series(arr)
chart = pd.concat([chart,arr],axis=1)
chart = chart.drop(['age','index','birth year','depression_score'],axis=1)
chart =chart.rename(columns={0:'age'})
chart.head()
```

	user_id	sex	depression_class	diabetes	high_pressure	blood_vessel	heart	joint	etc	steps	age
0	519.0	F	Moderate	1.0	1.0	0.0	0.0	0.0	1.0	small	high
1	520.0	F	Mild	0.0	0.0	0.0	0.0	0.0	0.0	medium	high
2	580.0	F	Normal	0.0	0.0	0.0	1.0	0.0	0.0	small	high
3	495.0	F	Normal	0.0	0.0	0.0	0.0	0.0	0.0	small	high
4	486.0	F	Moderate	0.0	1.0	0.0	0.0	1.0	0.0	large	high

The age distribution of the users was between 70 and 90, so I divided ages into high and low based on the age of 80. At the same time, I deleted the columns 'birth year' and 'depression score' that I decided not to use.

```
tag_1,tag_2,tag_3,tag_4,tag_5,tag_6,tag_7,tag_8,tag_9,tag_10,tag_11,tag_12=[],[],[],[],[],[],[],[],[],[],[],[]

for i in chart['user_id']:
    tag1,tag2,tag3,tag4,tag5,tag6,tag7,tag8,tag9,tag10,tag11,tag12=0,0,0,0,0,0,0,0,0,0,0,0
    temp=df[df['owner_id'] == i].copy()

    for j in temp['tag_id']:
        if j ==  1:
            tag1+=1
        elif j ==  2:
            tag2+=1
        elif j ==  3:
            tag3+=1
        elif j ==  4:
            tag4+=1
        elif j ==  5:
            tag5+=1
        elif j ==  6:
            tag6+=1
        elif j ==  7:
            tag7+=1
        elif j ==  8:
            tag8+=1
        elif j ==  9:
            tag9+=1
        elif j ==  10:
            tag10+=1
        elif j ==  11:
            tag11+=1
        elif j ==  12:
            tag12+=1

    sum_ = tag1+tag2+tag3+tag4+tag5+tag6+tag7+tag8+tag9+tag10+tag11+tag12
    tag1 /= sum_
    tag2 /= sum_
    tag3 /= sum_
    tag4 /= sum_
    tag5 /= sum_
    tag6 /= sum_
    tag7 /= sum_
    tag8 /= sum_
    tag9 /= sum_
    tag10 /= sum_
    tag11 /= sum_
    tag12 /= sum_
```

```python
    tag_1.append(tag1)
    tag_2.append(tag2)
    tag_3.append(tag3)
    tag_4.append(tag4)
    tag_5.append(tag5)
    tag_6.append(tag6)
    tag_7.append(tag7)
    tag_8.append(tag8)
    tag_9.append(tag9)
    tag_10.append(tag10)
    tag_11.append(tag11)
    tag_12.append(tag12)

tag_1f = []
mean = np.mean(tag_1)
for i in tag_1:
    if i>= mean:
        tag_1f.append("high")
    else:
        tag_1f.append("low")

tag_2f = []
mean = np.mean(tag_2)
for i in tag_2:
    if i>= mean:
        tag_2f.append("high")
    else:
        tag_2f.append("low")

tag_3f = []
mean = np.mean(tag_3)
for i in tag_3:
    if i>= mean:
        tag_3f.append("high")
    else:
        tag_3f.append("low")


tag_4f = []
mean = np.mean(tag_4)
for i in tag_4:
    if i>= mean:
        tag_4f.append("high")
    else:
        tag_4f.append("low")
```

```python
tag_5f = []
mean = np.mean(tag_5)
for i in tag_5:
    if i>= mean:
        tag_5f.append("high")
    else:
        tag_5f.append("low")


tag_6f = []
mean = np.mean(tag_6)
for i in tag_6:
    if i>= mean:
        tag_6f.append("high")
    else:
        tag_6f.append("low")


tag_7f = []
mean = np.mean(tag_7)
for i in tag_7:
    if i>= mean:
        tag_7f.append("high")
    else:
        tag_7f.append("low")


tag_8f = []
mean = np.mean(tag_8)
for i in tag_8:
    if i>= mean:
        tag_8f.append("high")
    else:
        tag_8f.append("low")


tag_9f = []
mean = np.mean(tag_9)
for i in tag_9:
    if i>= mean:
        tag_9f.append("high")
    else:
        tag_9f.append("low")


tag_10f = []
mean = np.mean(tag_10)
for i in tag_10:
    if i>= mean:
        tag_10f.append("high")
    else:
        tag_10f.append("low")
```

```python
tag_12f = []
mean = np.mean(tag_12)
for i in tag_12:
    if i>= mean:
        tag_12f.append("high")
    else:
        tag_12f.append("low")

tag_1f = pd.Series(tag_1f)
chart = pd.concat([chart,tag_1f],axis=1)
chart =chart.rename(columns={0:'tag_1'})

tag_2f = pd.Series(tag_2f)
chart = pd.concat([chart,tag_2f],axis=1)
chart =chart.rename(columns={0:'tag_2'})

tag_3f = pd.Series(tag_3f)
chart = pd.concat([chart,tag_3f],axis=1)
chart =chart.rename(columns={0:'tag_3'})

tag_4f = pd.Series(tag_4f)
chart = pd.concat([chart,tag_4f],axis=1)
chart =chart.rename(columns={0:'tag_4'})

tag_5f = pd.Series(tag_5f)
chart = pd.concat([chart,tag_5f],axis=1)
chart =chart.rename(columns={0:'tag_5'})

tag_6f = pd.Series(tag_6f)
chart = pd.concat([chart,tag_6f],axis=1)
chart =chart.rename(columns={0:'tag_6'})

tag_7f = pd.Series(tag_7f)
chart = pd.concat([chart,tag_7f],axis=1)
chart =chart.rename(columns={0:'tag_7'})

tag_8f = pd.Series(tag_8f)
chart = pd.concat([chart,tag_8f],axis=1)
chart =chart.rename(columns={0:'tag_8'})

tag_9f = pd.Series(tag_9f)
chart = pd.concat([chart,tag_9f],axis=1)
chart =chart.rename(columns={0:'tag_9'})

tag_10f = pd.Series(tag_10f)
chart = pd.concat([chart,tag_10f],axis=1)
chart =chart.rename(columns={0:'tag_10'})

tag_11f = pd.Series(tag_11f)
chart = pd.concat([chart,tag_11f],axis=1)
chart =chart.rename(columns={0:'tag_11'})

tag_12f = pd.Series(tag_12f)
chart = pd.concat([chart,tag_12f],axis=1)
chart =chart.rename(columns={0:'tag_12'})
```

The user's tagging information was also added using the code above. First, save the tag type ratio of each user. If the ratio of each tag is higher than the average of all users, it is saved as high, and if it is lower, it is saved as low.

```
chart.head()
```

	user_id	sex	depression_class	diabetes	high_pressure	blood_vessel	heart	joint	etc	steps	...	tag_3	tag_4	tag_5	tag_6	tag_7	tag_8	tag_9	tag_1
0	519.0	F	Moderate	1.0	1.0	0.0	0.0	0.0	1.0	small	...	low	low	low	high	low	low	high	lo
1	520.0	F	Mild	0.0	0.0	0.0	0.0	0.0	0.0	medium	...	low	high	low	high	low	low	high	hig
2	580.0	F	Normal	0.0	0.0	0.0	1.0	0.0	0.0	small	...	low	low	low	low	low	low	high	hig
3	495.0	F	Normal	0.0	0.0	0.0	0.0	0.0	0.0	small	...	low	low	low	high	low	low	high	lo
4	486.0	F	Moderate	0.0	1.0	0.0	0.0	1.0	0.0	large	...	low	low	low	high	low	low	high	lo

This is the final user information dataframe after pre-processing. All variables are categorical.

```
predictors = ['sex','diabetes','high_pressure','blood_vessel','joint','etc','heart','steps','age','tag_1','tag_2','tag_3','tag_4','tag_5','
outcome = 'depression_class'
x = pd.get_dummies(chart[predictors])
y = chart[outcome].astype('category')
classes = list(y.cat.categories)

x_train,x_valid,y_train,y_valid = train_test_split(x,y,test_size=0.4,random_state=88)

chart_nb = MultinomialNB(alpha=.01)
chart_nb.fit(x_train,y_train)
y_train_pred = chart_nb.predict(x_train)


classificationSummary(y_train,y_train_pred,class_names=classes)
```

```
Confusion Matrix (Accuracy 0.7097)

            Prediction
         Actual          Mild        Moderate Moderately severe       Normal       Severe
           Mild             5               0                 0            1            0
       Moderate             0               3                 0            2            0
Moderately severe           1               0                 3            3            0
         Normal             0               2                 0           10            0
         Severe             0               0                 0            0            1
```

I made a model with pre-processed data and run on a training set. It had 70% of accuracy which can be observed in confusion matrix.

b. Compute the overall error, specificity, sensitivity, false-positive error, and false-negative error for the validation set.

```
y_valid_pred = chart_nb.predict(x_valid)
classificationSummary(y_valid,y_valid_pred,class_names=classes)

print("Overall Error:",1-0.4762)
print("Sensitivity:",8/12)
print("Specificity:",2/9)
print("false-positive error:",4/12)
print("false-negative error:",7/9)
```

```
Confusion Matrix (Accuracy 0.4762)

                  Prediction
           Actual            Mild         Moderate Moderately severe            Normal            Severe
             Mild               0                0                 0                 1                 0
         Moderate               2                1                 1                 0                 0
Moderately severe               0                0                 1                 2                 0
           Normal               1                1                 2                 8                 0
           Severe               0                0                 0                 1                 0
Overall Error: 0.5238
Sensitivity: 0.6666666666666666
Specificity: 0.2222222222222222
false-positive error: 0.3333333333333333
false-negative error: 0.7777777777777778
```

I thought whether there is depression or not was the most important thing, so I chose 'normal' s the most important class. Overall error is 1-accuracy = 1 – 0.4762 = 0.5238. Sensitivity is P((prediction Normal) | (Actual Normal)). Actual Normal = 12 and prediction Normal in actual Normal= 8, so 8/12 = 0.67. Specificity is P (prediction is True | !(Acutal Normal)). !(Actual Normal) = 1+4+3+1 = 9, and true prediction above them = 2, so 2/9 = .22. False-positive error is P(Prediction Wrong| Prediction Normal) = 4/12 = .33. False-negative error is P(Prediction wrong| !(Prediction Normal)) = 7/9 = .78.

c. Examine the conditional probabilities of the output. What is the probability of a user having 'moderately severe' depression when their average total steps per day is below 10000? $P(DC = "Moderately\ severe" | (average\ total\ steps\ per\ day) \leq 10000)$?

```
y_valid_pred= pd.Series(y_valid_pred)
x_valid = x_valid.reset_index()
x_valid = pd.concat([x_valid,y_valid_pred],axis=1)
x_valid = x_valid.rename(columns={0:'y_valid_pred'})
temp = x_valid[x_valid['steps_large']!=1]
print(temp.groupby('y_valid_pred').size())

print("Probability:", 4/len(temp))
```

```
y_valid_pred
Mild                 2
Moderately severe    4
Normal               9
dtype: int64
Probability: 0.26666666666666666
```

I concatenated the x_valid and y_valid_pred to see the prediction easier. I excluded people who walked an average of 10,000 steps or more per day from the set and confirmed that four of them had 'moderately severe' characteristics. The total set is 15 people, so the probability is 4/15 = 0.27.

Q3. Classification modeling with k-nearest neighbors classifier for daily activities. First, you need to annotate the daily activities from the 13 actions for the three people assigned as follows with the provided tagging tool.

a. Following the instruction in “How to Create Annotations (Q3).pdf”, annotate the activities for the three people designated to you (User1~3), and describe the life patterns for each person.

I grouped each user's annotation by time and analyzed which label had the largest value.

1. User 644

```
pd.options.display.max_rows=140
p_644 = pd.read_csv('label_644.csv')
p_644['start'] = pd.to_datetime(p_644.start)
p_644['end'] = pd.to_datetime(p_644.end)
temp = p_644.groupby([p_644.start.dt.hour,p_644.labelName]).size()
temp
```

start	labelName	
2	Sleep	1
3	Go to bathroom	1
5	Go to bathroom	3
	Wake up	2
6	Eat	2
	Go to bathroom	6
	Wake up	1
	Wash dishes	1
7	Eat	6
	Go to bathroom	2
	Wake up	8
8	Cook	1
	Drink	1
	Eat	7
	Go to bathroom	8
	Wake up	7
	Wash dishes	1
9	Cook	9
	Eat	10
	Go to bathroom	6
	Take medicine	1
	Take shower	1
	Wake up	4
	Wash dishes	1
10	Cook	6
	Eat	12
	Go to bathroom	3
	Take medicine	2
	Wake up	2
11	Cook	6
	Drink	1
	Eat	13
	Go to bathroom	7
	Take medicine	1
	Wash dishes	1
12	Cook	4
	Eat	12
	Go to bathroom	2
	Take shower	1
	Wash dishes	2
13	Drink	1
	Eat	12
	Go to bathroom	4
	Wash dishes	2
14	Cook	2
	Eat	8
	Go to bathroom	7
	Take shower	2
15	Cook	5
	Eat	5
	Go to bathroom	5
	Take shower	1
	Wash dishes	2
16	Cook	6
	Drink	2
	Eat	8
	Go to bathroom	7
	Take medicine	1
	Wash dishes	1
17	Cook	6
	Drink	2
	Eat	11
	Go to bathroom	3
	Take medicine	1
18	Cook	1
	Drink	1
	Eat	12
	Go to bathroom	3
	Sleep	2
	Take shower	1
19	Cook	2
	Eat	5
	Go to bathroom	5
	Sleep	1
	Take shower	1
	Wash dishes	1
20	Drink	3
	Eat	4
	Go to bathroom	5
	Sleep	1
	Wash dishes	1
21	Drink	1
	Eat	5
	Go to bathroom	4
	Sleep	3
	Take medicine	1
	Wash dishes	2
22	Drink	1
	Eat	1
	Go to bathroom	3
	Sleep	5
	Take medicine	1
23	Eat	1
	Go to bathroom	2
	Sleep	4
	Wash dishes	1

User 644 usually gets up at 7 to 8, eats breakfast at 9 to 10, lunch at 12 to 13, dinner at 18 to 19, and sleeps at 22 to 23. Medicine is usually taken between 9-10 o'clock, 16-17 o'clock, and 21-22 o'clock. He's living his life in a pretty regular pattern.

2. User 651

I used the same method as user 644.

0	Cook	1
	Eat	1
	Go to bathroom	7
1	Drink	1
	Eat	1
	Go to bathroom	13
2	Eat	2
	Go to bathroom	19
3	Go to bathroom	10
	Take shower	1
4	Go to bathroom	21
	Take shower	1
	Wake up	2
5	Go to bathroom	22
	Wake up	8
6	Drink	1
	Eat	1
	Go to bathroom	36
	Take shower	8
	Wake up	5
	Wash dishes	1
7	Drink	11
	Eat	1
	Go to bathroom	29
	Go walk	1
	Take medicine	2
	Take shower	11
	Wake up	4
	Wash dishes	8
8	Drink	3
	Eat	4
	Go to bathroom	28
	Go walk	1
	Take medicine	2
	Take shower	9
	Wash dishes	5
9	Drink	2
	Eat	1
	Go to bathroom	10
10	Cook	1
	Drink	3
	Eat	4
	Go to bathroom	14
	Wake up	1
	Wash dishes	3
11	Cook	1
	Drink	12
	Eat	10
	Go to bathroom	9
	Take shower	2
	Wash dishes	8
12	Cook	1
	Drink	9
	Eat	15
	Go to bathroom	12
	Take shower	3
	Wash dishes	10

13	Cook	1
	Drink	5
	Eat	9
	Go to bathroom	11
	Take medicine	2
	Wash dishes	2
14	Cook	4
	Drink	8
	Eat	8
	Go to bathroom	15
	Take medicine	2
	Take shower	3
	Wake up	1
15	Cook	3
	Drink	5
	Eat	20
	Go to bathroom	16
	Take medicine	1
	Wash dishes	1
16	Cook	5
	Drink	7
	Eat	12
	Go to bathroom	18
	Sleep	3
	Take shower	3
	Wash dishes	1
17	Drink	8
	Eat	16
	Go to bathroom	13
	Sleep	2
	Take medicine	2
	Take shower	2
	Wash dishes	3
18	Cook	4
	Drink	6
	Eat	10
	Go to bathroom	7
	Sleep	3
	Take medicine	2
	Take shower	1
	Wash dishes	1
19	Drink	3
	Eat	10
	Go to bathroom	8
	Sleep	9
	Take medicine	4
	Take shower	3
	Wash dishes	1
20	Drink	3
	Eat	6
	Go to bathroom	6
	Sleep	10
	Take medicine	1
	Take shower	1
	Wash dishes	2

21	Drink	1
	Eat	3
	Go to bathroom	4
	Sleep	4
22	Drink	1
	Eat	6
	Go to bathroom	8
	Sleep	7
	Take shower	1
	Wash dishes	2
23	Drink	1
	Eat	4
	Go to bathroom	11
	Sleep	7

User 651 was living a very irregular life. He didn't have a particular pattern; he drank a lot and went to the bathroom often. He often eats between 15 and 19 o'clock. He didn't have a fixed time to sleep, and he showed a pattern of going to the bathroom often while sleeping.

3. User 505

0	Go to bathroom	6
	Sleep	1
1	Drink	1
	Go to bathroom	1
	Sleep	1
	Wash dishes	1
2	Go to bathroom	1
4	Go to bathroom	1
	Wake up	1
5	Go to bathroom	1
	Wake up	3
6	Go to bathroom	3
	Wake up	7
7	Cook	2
	Drink	5
	Go to bathroom	3
	Sleep	1
	Wake up	9
8	Cook	6
	Drink	3
	Eat	4
	Go to bathroom	12
	Wake up	50
	Wash dishes	3
9	Cook	13
	Drink	16
	Eat	14
	Go to bathroom	24
	Take medicine	2
	Take shower	2
	Wake up	68
	Wash dishes	5
10	Cook	15
	Drink	28
	Eat	17
	Go to bathroom	50
	Take medicine	3
	Take shower	3
	Wake up	17
	Wash dishes	5
11	Cook	25
	Drink	19
	Eat	42
	Go to bathroom	29
	Go walk	1
	Take medicine	6
	Take shower	4
	Wake up	6
	Wash dishes	5
12	Cook	23
	Drink	14
	Eat	38
	Go to bathroom	26
	Take medicine	3
	Take shower	2
	Wake up	1
	Wash dishes	3
13	Cook	20
	Drink	16
	Eat	40
	Go to bathroom	38
	Take shower	6
	Wash dishes	3
14	Cook	33
	Drink	12
	Eat	46
	Go to bathroom	31
	Go walk	1
	Take medicine	2
	Take shower	2
	Wake up	1
	Wash dishes	4
15	Cook	27
	Drink	15
	Eat	41
	Go to bathroom	28
	Take medicine	1
	Take shower	6
	Wash dishes	6
16	Cook	20
	Drink	10
	Eat	46
	Go to bathroom	33
	Take medicine	1
	Take shower	4
	Wash dishes	4
17	Cook	16
	Drink	25
	Eat	39
	Go to bathroom	31
	Go walk	1
	Sleep	3
	Take medicine	1
	Take shower	8
	Wash dishes	3
18	Cook	13
	Drink	34
	Eat	33
	Go to bathroom	27
	Sleep	4
	Take shower	1
	Wash dishes	10
19	Cook	3
	Drink	21
	Eat	22
	Go to bathroom	33
	Sleep	15
	Take medicine	2
	Take shower	1
	Wash dishes	5
20	Drink	13
	Eat	13
	Go to bathroom	23
	Sleep	24
	Take medicine	2
	Take shower	4
	Wash dishes	8
21	Cook	1
	Drink	8
	Eat	7
	Go to bathroom	22
	Sleep	34
	Take shower	2
	Wash dishes	9
22	Drink	4
	Eat	4
	Go to bathroom	11
	Sleep	31
	Take shower	6
	Wash dishes	1
23	Go to bathroom	7
	Go walk	1
	Sleep	25
	Take shower	1

User 505 has a relatively regular life compared to the previous case. User 505 usually get up at 8 or 9 and go to bed at 21 or 23. The amount of time he eats is not exactly fixed and is distributed throughout the whole time. Compared to other users, the time in the kitchen was longer, so I assumed that time was cooking time. One of the characteristics is that the amount of tag information was higher than that of the other two users.

b. Fit a k-nearest neighbor classifier using your annotated data for User1 and User2. Then classify the activities of User3 using the best k. Explain the daily activities of User3, and discuss the effect of k

```
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier

data = pd.concat([p_651,p_644])
data = data.reset_index()
data = data.drop("index",axis=1)
train, valid = train_test_split(data, test_size=0.4,random_state=0)

predictors = ['start','end']
scaler = preprocessing.StandardScaler()
scaler.fit(train[predictors])

dataNorm = pd.concat([pd.DataFrame(scaler.transform(data[predictors])),data['labelName']],axis=1)
trainNorm = dataNorm.iloc[train.index]
validNorm = dataNorm.iloc[valid.index]
p_505Norm = pd.DataFrame(scaler.transform(p_505[predictors]))

knn = NearestNeighbors(n_neighbors=3)
knn.fit(trainNorm.iloc[:,0:2])

distances, indices = knn.kneighbors(p_505Norm)
train_x = dataNorm[[0,1]]
train_y = dataNorm['labelName']
valid_x = p_505Norm[[0,1]]
valid_y = p_505['labelName']

results = []
for k in range(1,15):
    knn  = KNeighborsClassifier(n_neighbors=k).fit(train_x,train_y)
    results.append({'k':k,'accuracy':accuracy_score(valid_y,knn.predict(valid_x))})

results = pd.DataFrame(results)
print(results)


knn = KNeighborsClassifier(n_neighbors=9).fit(train_x,train_y)
p_505_pred = knn.predict(p_505Norm)
```

```
     k  accuracy
0    1  0.193311
1    2  0.195578
2    3  0.191610
3    4  0.229592
4    5  0.235261
5    6  0.235828
6    7  0.237528
7    8  0.234694
8    9  0.244331
9   10  0.225057
10  11  0.223356
11  12  0.220522
12  13  0.223923
13  14  0.227324
```

I used user 651 and 644 as the training data. To find the best k, I tried each k from 1 to 14. The accuracy was largest on k=9, so I used that. Since the training dataset is relatively small, I could not have better accuracy rate than this. If k is smaller, it has less accuracy because neighbors could contain many outliers. If k gets bigger, the outlier problem could be solved, but the range for neighbors can include other categories other than the answer.

```
pd.options.display.max_rows=140
p_644 = pd.read_csv('label_644.csv')
p_644['start'] = pd.to_datetime(p_644.start)
p_644['end'] = pd.to_datetime(p_644.end)
temp = p_644.groupby([p_644.start.dt.hour,p_644.labelName]).size()
temp
```

With this code, I made the prediction table according to the hours again.

0	Eat	3
	Go to bathroom	4
1	Cook	1
	Drink	1
	Eat	1
	Go to bathroom	1
2	Go to bathroom	1
4	Eat	1
	Go to bathroom	1
5	Cook	2
	Eat	1
	Go to bathroom	1
6	Drink	1
	Eat	5
	Go to bathroom	4
7	Drink	1
	Eat	8
	Go to bathroom	11
8	Cook	2
	Drink	2
	Eat	30
	Go to bathroom	44
9	Cook	1
	Drink	7
	Eat	54
	Go to bathroom	82
10	Cook	1
	Drink	12
	Eat	52
	Go to bathroom	73
11	Cook	4
	Drink	7
	Eat	59
	Go to bathroom	67
12	Cook	3
	Drink	7
	Eat	48
	Go to bathroom	52
13	Cook	2
	Drink	4
	Eat	54
	Go to bathroom	63
14	Cook	1
	Drink	6
	Eat	54
	Go to bathroom	70
	Wash dishes	1
15	Cook	3
	Drink	11
	Eat	56
	Go to bathroom	54
16	Cook	4
	Drink	4
	Eat	52
	Go to bathroom	58
17	Drink	7
	Eat	58
	Go to bathroom	62
18	Cook	5
	Drink	9
	Eat	54
	Go to bathroom	54
19	Cook	3
	Drink	4
	Eat	50
	Go to bathroom	45
20	Cook	2
	Drink	5
	Eat	31
	Go to bathroom	49
21	Cook	2
	Drink	5
	Eat	30
	Go to bathroom	46
22	Drink	6
	Eat	22
	Go to bathroom	29
23	Drink	2
	Eat	16
	Go to bathroom	16

Most of the values contain only Eat and go to bathroom. I think this is because most of the tagging information occurred in toilet and microwave. If the company wants to get more accurate information, I think it's better to find a better way to collect information. Based on this predicted information, the tag information from 23:00 to 8:00 is reduced, indicating that this is the approximate sleep time. And it is observable that he usually eats between 9 and 19 o'clock.