APPENDIX H: Programming Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
df= pd.read_csv('sample.csv')
df.head()
```

	timestamp	back_x	back_y	back_z	thigh_x	thigh_y	thigh_z	label
0	2019-01-12 00:00:00.000	-0.760242	0.299570	0.468570	-5.092732	-0.298644	0.709439	6
1	2019-01-12 00:00:00.010	-0.530138	0.281880	0.319987	0.900547	0.286944	0.340309	6
2	2019-01-12 00:00:00.020	-1.170922	0.186353	-0.167010	-0.035442	-0.078423	-0.515212	6
3	2019-01-12 00:00:00.030	-0.648772	0.016579	-0.054284	-1.554248	-0.950978	-0.221140	6
4	2019-01-12 00:00:00.040	-0.355071	-0.051831	-0.113419	-0.547471	0.140903	-0.653782	6

Lables and their corresponding anotation

1 walking
2 running
3 shuffling standing with leg movement 4 stairs (ascending)
5 stairs (descending)
6 standing
7 sitting
8 lying
13 cycling (sit)
14 cycling (stand)
130 cycling (sit, inactive) cycling (sit) without leg movement 140 cycling (stand, inactive) cycling (stand) without leg movement

Labels considered in this study

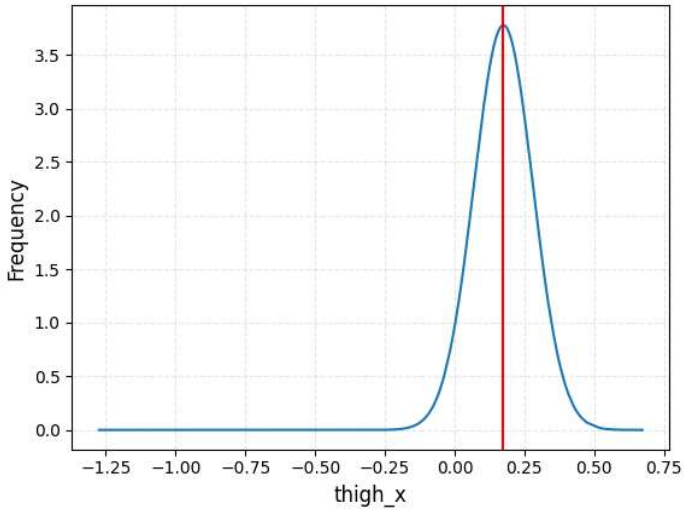
1 walking 6 standing 7 sitting 8 lying

```
req_labels= [1,6,7,8]
label_dic={
```

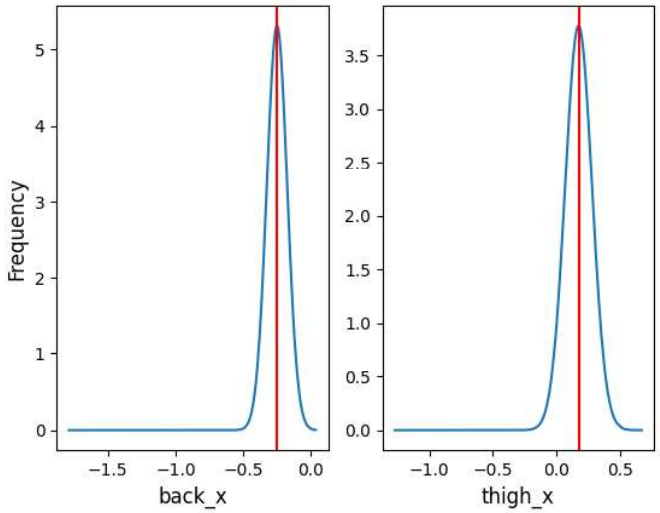
```
    1: "walking",
    6: "standing",
    7: "sitting",
    8: "lying"
}
sel_df= df.loc[df['label'].isin(req_labels)]
sel_df.label.value_counts()
```

```
7    253029
6     62682
1     24889
8     13036
Name: label, dtype: int64
```

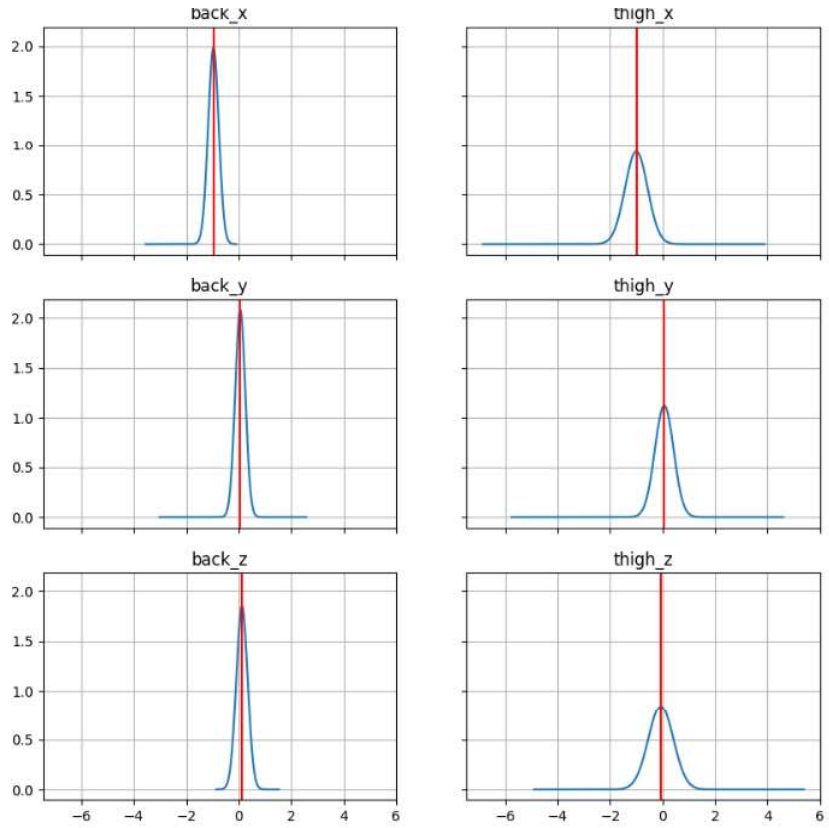
```
def find_pdf(df, column_name, label_value):

    filtered_df= df.loc[df['label']== int(label_value)][[column_name, 'label']].reset_index()

    mean= np.mean(filtered_df[column_name])
    std= np.std(filtered_df[column_name])

    return stats.norm.pdf(filtered_df[column_name].sort_values(), mean, std),mean,std
def plot_pdf(df, column_name, label_value):
    pdf, mean, std= find_pdf(df, column_name, label_value)
    plt.plot(df.loc[df['label']==                    int(label_value)][[column_name,
'label']].reset_index()[column_name].sort_values(), pdf)
    plt.axvline(x= mean, color= 'r')
    plt.xlabel(column_name, size=12)
    plt.ylabel("Frequency", size=12)
    plt.grid(True, alpha=0.3, linestyle="--")
    plt.show()
plot_pdf(df, 'thigh_x', 8)
```




```
def comparison_pdf_plot(df,column_name1, column_name2, label_value):
    x1=                df.loc[df['label']==                int(label_value)][[column_name1,
'label']].reset_index()[column_name1].sort_values()
    x2=                df.loc[df['label']==                int(label_value)][[column_name2,
'label']].reset_index()[column_name2].sort_values()
    y1, m1, s1= find_pdf(df, column_name1, label_value)
    y2, m2, s2= find_pdf(df, column_name2, label_value)
    fig, axes = plt.subplots(nrows=1, ncols=2)
    axes[0].plot(x1, y1)
    axes[0].axvline(x= m1, color= 'r')
    axes[0].set_xlabel(column_name1, size=12)
    axes[0].set_ylabel("Frequency", size=12)
    axes[1].plot(x2, y2)
    axes[1].axvline(x= m2, color= 'r')
    axes[1].set_xlabel(column_name2, size=12)
```

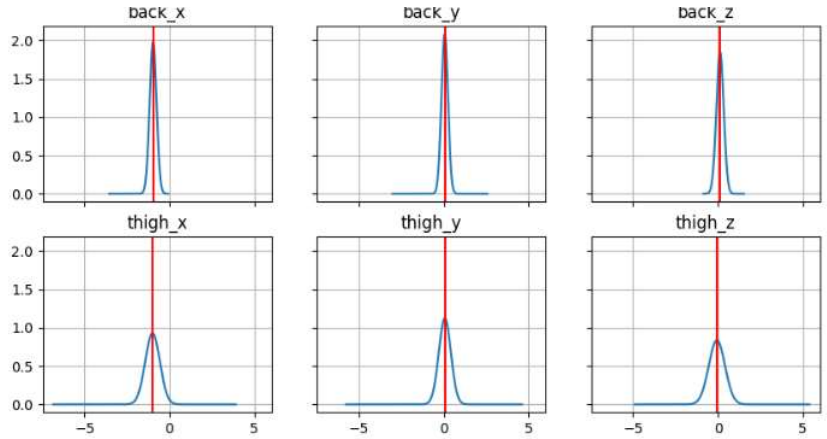



```
def all_comparsion_plot_horizontal(df, label_value):
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10, 10), sharex=True, sharey=True)

    i=0
    j=0
    for col in df.columns[1:-1]:
        x= df.loc[df['label']== int(label_value)][[col, 'label']].reset_index()[col].sort_values()
        y, mean, std= find_pdf(df, col, label_value)
        axes[i,j].plot(x,y)
        axes[i,j].axvline(x= mean, color= 'r')
        axes[i,j].set_title(col, size=12)
        axes[i,j].grid()

        i+=1
        if i==3:
            i=0
            j+=1
all_comparsion_plot_horizontal(df,1)
```



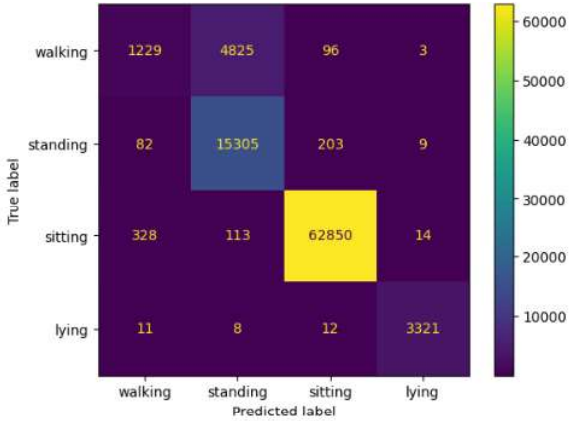

```
def all_comparsion_plot_vertical(df, label_value):
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(10, 5), sharex=True, sharey=True)

    i=0
    j=0
    for col in df.columns[1:-1]:
        x= df.loc[df['label']== int(label_value)][[col, 'label']].reset_index()[col].sort_values()
        y, mean, std= find_pdf(df, col, label_value)
        axes[i,j].plot(x,y)
        axes[i,j].axvline(x= mean, color= 'r')
        axes[i,j].set_title(col, size=12)
```

```
        axes[i,j].grid()

        j+=1
        if j==3:
            j=0
            i+=1
all_comparsion_plot_vertical(df,1)
```

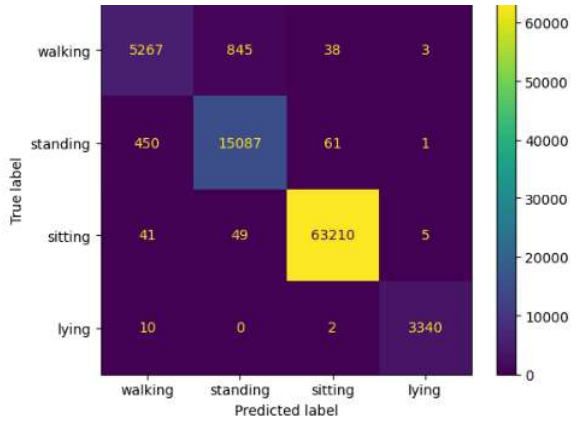



```
from sklearn.model_selection import train_test_split

X= sel_df[df.columns[1:-1]]
y= sel_df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)
from sklearn import svm
clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(X_train, y_train)
```

```
y_pred_svm = clf_svm.predict(X_test)
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


def get_confusion_matrix(y_true, y_pred):
    cm= confusion_matrix(y_test, y_pred)
    cm_display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
label_dic.values())
    cm_display.plot()
    plt.show()
    return cm
cf_svm= get_confusion_matrix(y_test, y_pred_svm)
```




```
from sklearn.neighbors import KNeighborsClassifier

clf_knn = KNeighborsClassifier(n_neighbors=3)
clf_knn.fit(X_train, y_train)
y_pred_knn= clf_knn.predict(X_test)
cm_knn= get_confusion_matrix(y_test, y_pred_knn)
```



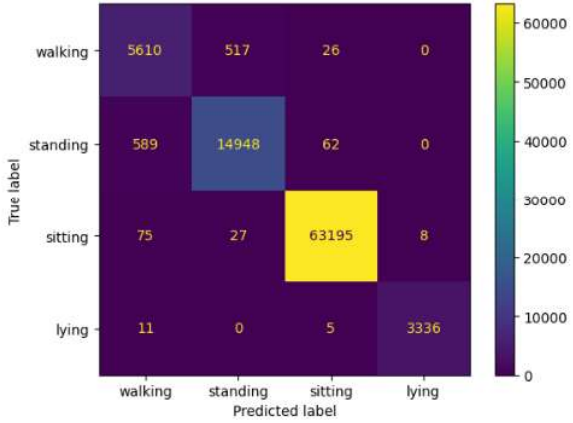
from sklearn.ensemble import RandomForestClassifier

clf_rf=RandomForestClassifier(n_estimators=100)

clf_rf.fit(X_train,y_train)

y_pred_rf=clf_rf.predict(X_test)

cm_rf= get_confusion_matrix(y_test, y_pred_rf)



Code to create Patient daily body vitals details in random and store in info_db

import random

```
def get_temp():
    return round(random.uniform(97, 99),2)

def get_bloodsugar():
    return round(random.uniform(7, 12),2)

def get_fluidin():
    return int(random.uniform(300, 1500))

def get_fluidout():
    return int(random.uniform(300, 1000))

def get_foodtake():
    f= ['100', '75', '50', '25']
    return random.choice(f)

if __name__== "__main__":
    import pandas as pd
    import datetime
    import sqlite3



    conn = sqlite3.connect('patientsinfo_database')
    c = conn.cursor()

    c.execute('CREATE TABLE IF NOT EXISTS products (id, date, temperature, sugar, fluidin,
fuildout, food)')
    conn.commit()

    days= 30
    patients= 10
```

```
current_date= datetime.datetime(2022,11,1,20,00,00)

db= pd.DataFrame(columns=['id', 'date', 'temperature', 'sugar', 'fluidin', 'fuildout', 'food'])

index=0
day=0
while True:
    for i in range(patients):
        db.loc[index]= [
            str(i),
            current_date,
            get_temp(),
            get_bloodsugar(),
            get_fluidin(),
            get_fluidout(),
            get_foodtake()
        ]
        index+=1
    current_date+= datetime.timedelta(hours=24)
    day+=1
    if day== days:
        break
db.to_csv('info_db.csv')
db.to_sql('patientsinfoDB', conn, if_exists='replace', index = False)
```

Code to create daily activities of 10 residents from 08:00 am to 08:00 pm for 1 month and store in a database

```
import random
import datetime
```

```
import sqlite3

''' Initial date time: 1st November 2022 8hr:0min:0s am'''
start_date_time= datetime.datetime(2022,11,1,8,00,00)

no_of_days= 30
no_of_patients=10




conn = sqlite3.connect('patients_database')
c = conn.cursor()

c.execute('CREATE TABLE IF NOT EXISTS products (id, timestamp, activity)')
conn.commit()

'''List of activities a pantient performs daily'''
activity_list=[
    "lying",
    "sitting",
    "standing",
    "walking"
]

'''Type of patients and probabilities of the patient doing an activity'''
patient_type_list={
    "bedridden": [90, 10, 0, 0],
    "in active": [80, 10, 7, 3],
    "active": [60, 30,  5, 5],
    "fully active": [50, 30, 10, 10],
```

```
}

sitting_time= [
    datetime.datetime(2022,11,1,9,00,00),
    datetime.datetime(2022,11,1,12,00,00),
    datetime.datetime(2022,11,1,4,00,00)
]

def get_heart_rate():
        return int(random.uniform(80,100))

def get_spo2():
    return int(random.uniform(90,100))



def update_lying_to_db(patient_id, last_activity, current_activity, last_updated_time, heart_rate,
spo2, df):
    index= len(df)

    if last_updated_time in sitting_time:
        current_activity= "sitting"
        last_activity= "sitting"

    if current_activity== last_activity:
        for t in range(1,16):
            last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1

    else:
```

```
        for t in range(1,16):
            if t<5:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
            else:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)

            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1


    return {
        "current_activity": current_activity,
        "last_updated_time": last_updated_time
    }

def update_sitting_to_db(patient_id, last_activity, current_activity, last_updated_time, heart_rate,
spo2, df):
    index= len(df)

    if last_updated_time in sitting_time:
        current_activity= "sitting"
        last_activity= "sitting"

    for t in range(1,16):
        last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
        df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
        index+=1

    return {
        "current_activity": current_activity,
        "last_updated_time": last_updated_time
```

```
    }

def update_standing_to_db(patient_id, last_activity, current_activity, last_updated_time,
heart_rate, spo2, df):
    index= len(df)

    if last_updated_time in sitting_time:
        current_activity= "sitting"
        last_activity= "sitting"

    if current_activity== last_activity:
        current_activity= "sitting"
        for t in range(1,16):
            last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1
    else:
        for t in range(1,16):
            if t<5:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
            else:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)

            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1

    return {
        "current_activity": current_activity,
        "last_updated_time": last_updated_time
    }
```

```
def update_walking_to_db(patient_id, last_activity, current_activity, last_updated_time, heart_rate, spo2, df):
    index= len(df)

    if last_updated_time in sitting_time:
        current_activity= "sitting"
        last_activity= "sitting"

    if current_activity== last_activity:
        current_activity= "sitting"
        for t in range(1,16):
            last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)
            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1

    else:
        for t in range(1,16):
            if t<5:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)

            else:
                last_updated_time= last_updated_time+ datetime.timedelta(minutes=1)

            df.loc[index]= [patient_id, last_updated_time, current_activity, heart_rate, spo2]
            index+=1

    return {
        "current_activity": current_activity,
        "last_updated_time": last_updated_time
    }
```

```
class Patient:
    global patient_type_list
    global start_date_time

    def __init__(self, id):

        self.id= id

        self.current_activity= "lying"
        self.last_activity= "lying"

        self.last_updated_time= start_date_time

    '''
    Assaigns a patient type randomly from the patient type dictionary
    '''

    def get_patient_type(self):

        self.patient_type= random.choice(list(patient_type_list.keys()))

    def update_activity(self):

        self.last_activity= self.current_activity
        self.current_activity= random.choices(
            activity_list,
            weights=patient_type_list[self.patient_type],
            k=1
        )[0]

    def update_activity_to_db(self,db):
```

```
self.heart_beat= get_heart_rate()
self.spo2= get_spo2()
if self.current_activity== "lying":
    results= update_lying_to_db(
        self.id,
        self.last_activity,
        self.current_activity,
        self.last_updated_time,
        self.heart_beat,
        self.spo2,
        db
    )
    self.last_activity= results['current_activity']
    self.last_updated_time= results['last_updated_time']

elif self.current_activity== "sitting":
    results= update_sitting_to_db(
        self.id,
        self.last_activity,
        self.current_activity,
        self.last_updated_time,
        self.heart_beat,
        self.spo2,
        db
    )
    self.last_activity= results['current_activity']
    self.last_updated_time= results['last_updated_time']

elif self.current_activity== "standing":
    results= update_standing_to_db(
        self.id,
```

```
            self.last_activity,
            self.current_activity,
            self.last_updated_time,
            self.heart_beat,
            self.spo2,
            db
        )
        self.last_activity= results['current_activity']
        self.last_updated_time= results['last_updated_time']

    elif self.current_activity== "walking":
        results= update_walking_to_db(
            self.id,
            self.last_activity,
            self.current_activity,
            self.last_updated_time,
            self.heart_beat,
            self.spo2,
            db
        )
        self.last_activity= results['current_activity']
        self.last_updated_time= results['last_updated_time']

def update_heartbeat(self):
    self.heart_beat= int(random.uniform(80,100))
def reset_time(self):
    if self.last_updated_time.time().hour>=20:
        self.last_updated_time+= datetime.timedelta(hours=12)
    else:
        pass
```

```
if __name__== "__main__":
    import pandas as pd

    db= pd.DataFrame(columns=['id', 'timestamp', 'activity', 'heartrate', 'spo2'])
    patients=[]
    for i in range(no_of_patients):
        patients.append(Patient(i))

    for ts in range(no_of_days*12*4):
        for patient in patients:

            patient.get_patient_type()
            patient.update_activity()
            patient.update_activity_to_db(db)
            patient.reset_time()

    db.to_csv('db.csv')
    db.to_sql('patientsDB', conn, if_exists='replace', index = False)
```