# Team Members

Ghada Ahmed Hamdy

Esraa Moataz Farouk

Hassnaa Alaa El-dean

Neama Essam Mohammed

Rana Alaa Ahmed

Elham Hesham Refaat

Aml Mohammed Abdullah

# Data preprocessing

**1-Convert Time from h:s:m to minutes**

**2-Replace 00:00:00 to 24:00:00**

```
data = pd.read_csv('call10.csv')
data_df_o=data.time_stamp
d=int(data_df_o[0].split()[0].split('-')[2])
h=int(data_df_o[0].split()[1].split(':')[0])
if(h==0):
    h=24
m=int(data_df_o[0].split()[1].split(':')[1])
t=h*60+m
```

## 1- How to identify the outliers?!

**- We removed the outliers using *Standard Deviation* and *Normal Distribution* where :-**

**Standard deviation is a metric of variance i.e. how much the individual data points are spread out from the mean.**

**- We needed to remove these outlier values because they were making the scales on our graph unrealistic.**

**The challenge was that the number of these outlier values was never fixed. Sometimes we would get all valid values and sometimes these erroneous readings would cover as much as 10% of the data points.**

**- Our approach was to remove the outlier points by eliminating any points that were above *(Mean + 1.5*SD)* and any points below *(Mean – 1.5*SD)***

```
for x in range (len(arr)):
    if (arr[x] > mean - 1.5 * sd):
        f.append(arr[x])
        ind1.append(ind[x])

for x in range (len(f)):
    if (f[x] < mean + 1.5 * sd):
        ff.append(f[x])
        ind2.append(ind1[x])
```

# 2- How to identify home and work location?!


**First Specify the features that clusters we need depends on ( *time* , *X_ Coordinate* , *Y_Coordinate* of tower_id )**

## Clustering

**First we initialize 3 lists f1 [ ] ,f2[ ] , f3[ ] , then append in**

**f1->Seconds**

**f2->Y_coordinate**

**f3->X_coordinate**

**And then use function *Zip()* to combine them into one list ,**
**this list represent the 3 features entered in clustering algorithm .**

```
        t=data.tower_id[i]
        f1.append(int(data.seconds[i]))
        f2.append(data2.Y_coordinate[t])
        f3.append(data2.X_coordinate[t])
    else :
        x = np.array(list(zip(f1,f2,f3)))
```

**For clustering, we used *k-mean* to cluster the data into 2 categories ( home , work )**

**Implement K-Means Clustering using *scikit-learn***

```
        # Number of clusters
        kmeans = KMeans(n_clusters=2)
        # Fitting the input data
        kmeans = kmeans.fit(x)
        # Getting the cluster labels
        labels = kmeans.predict(x)
```

**Sample of the result of k-means for the first 11 users.**

```
[1 0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1]
1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0]
1 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1
0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1
0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1
[1 0 0 1 0
[1 1 1 0 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 0]
[1 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0]
[0 1 0 1 1 0 0 0 1 1 0 0
[1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1]
[0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0]
[1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1]
0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0]
[1 0 0 0 0 0 0 1 0 0 0 0 0 0
[1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0]
```

**Each user have a single list (labels) consists of 0,1 that are produced for k-means.**

**Until this step we don't know what 0 represent and 1 represent**

***Home or Work ??!***

# Find home/work location

**For each user's label we take the first value then find the corresponding time .**

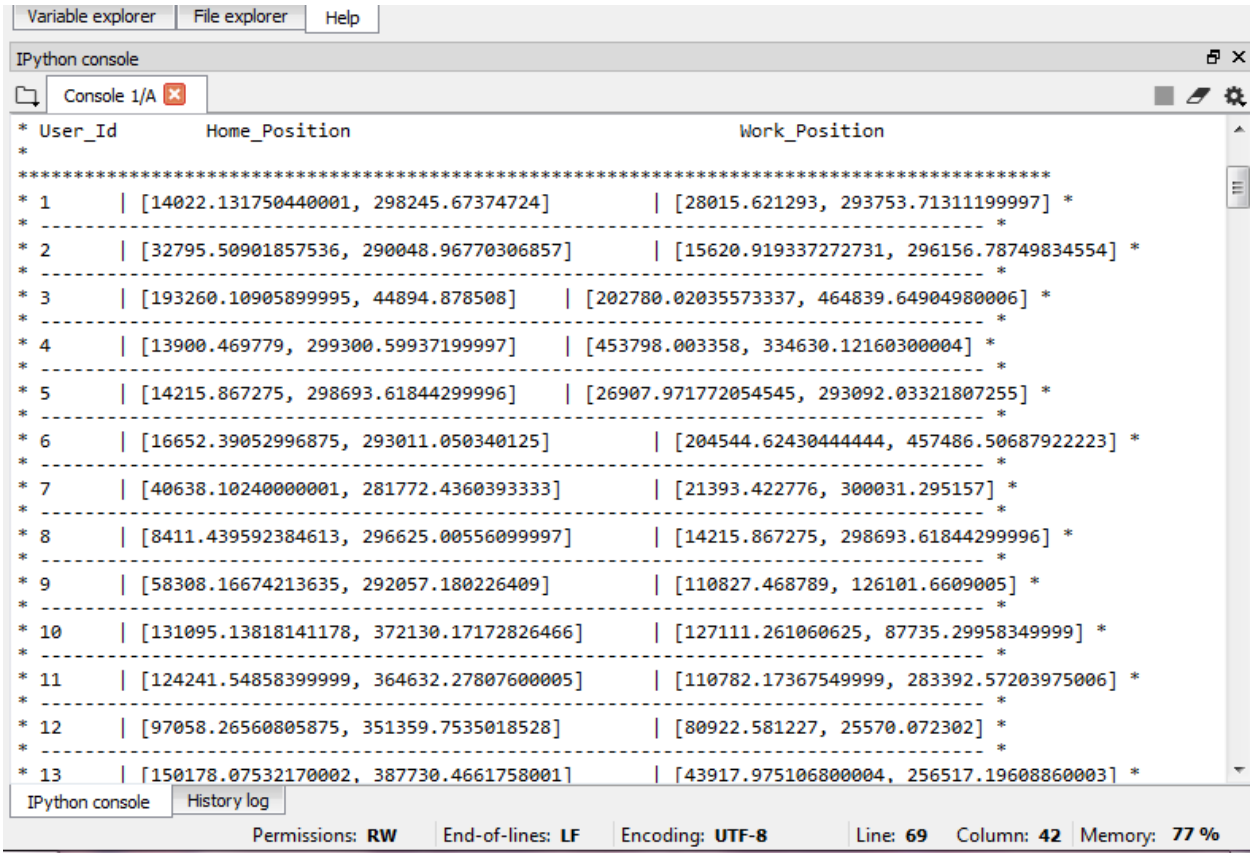
**We considered that user from *06:00 PM To 06:00 AM* is at *home* ,*otherwise* he could be at *work* .**

**Home position = the average of towers position in home cluster**

**Work position = the average of towers position in work cluster**

```
L=labels[0]
h=f1[0]/(60*60)
if(h>=18 or h<=6):
    for i in range (len(labels)):
        if (labels[i]==L):
            #home
            sumx1+=f3[i]
            sumy1+=f2[i]
            count1+=1
        else :
            #work
            sumx2+=f3[i]
            sumy2+=f2[i]
            count2+=1
else:
    for i in range (len(labels)):
        if (labels[i]==L):
            #work
            sumx2+=f3[i]
            sumy2+=f2[i]
            count2+=1
        else :
            #home
            sumx1+=f3[i]
            sumy1+=f2[i]
            count1+=1
homepx=sumx1/count1
homepy=sumy1/count1
workpx=sumx2/count2
workpy=sumy2/count2
```

# Final output is a table consists of *user_id* , *home position* , *work position* for each user :



## 3.1- What's the distribution of the frequently visited locations over the whole dataset?!

**We first need to find the towers in each place , and for that we use *DBSCAN* with *eps=1600*, *min sample =1* to cluster the towers with respect to places .**

```
for i in range(len(tower_data)) :
    x.append([tower_data.x[i] , tower_data.y[i]])
clustering = DBSCAN(eps=1600, min_samples=1).fit(x)
result = clustering.labels_
```

**Now , we have the places of the entire data set with the towers corresponding to each place.**

```
for i in range(1,len(result)) :
    if str(result[i]) not in places :
        places[str(result[i])] = [tower_data.ID[i]]
    else:
        places[str(result[i])].append(tower_data.ID[i])
```

**Then we count how many times each place is visited by users of the data set.**

```
for i in range(len(user_data.tower_id)) :
    for j in places :
        if str(user_data.tower_id[i]) in j :
            if j in visited :
                if  str(user_data.id[i]) not in temp :
                    temp[str(user_data.id[i])] = [user_data.day [i]]
                    visited[j] = visited[j] + 1
                elif user_data.day[i] not in temp[str(user_data.id[i])] :
                    temp[str(user_data.id[i])].append(user_data.day[i])
                    visited[j] = visited[j] + 1
            else :
                visited[j] = 1
```

**We plot the distribution between the places and the visiting frequency :-**

```
pyplot.plot(visited.keys(), visited.values())
plt.xlabel("places")

plt.ylabel("visiting frequency")
plt.show()
```

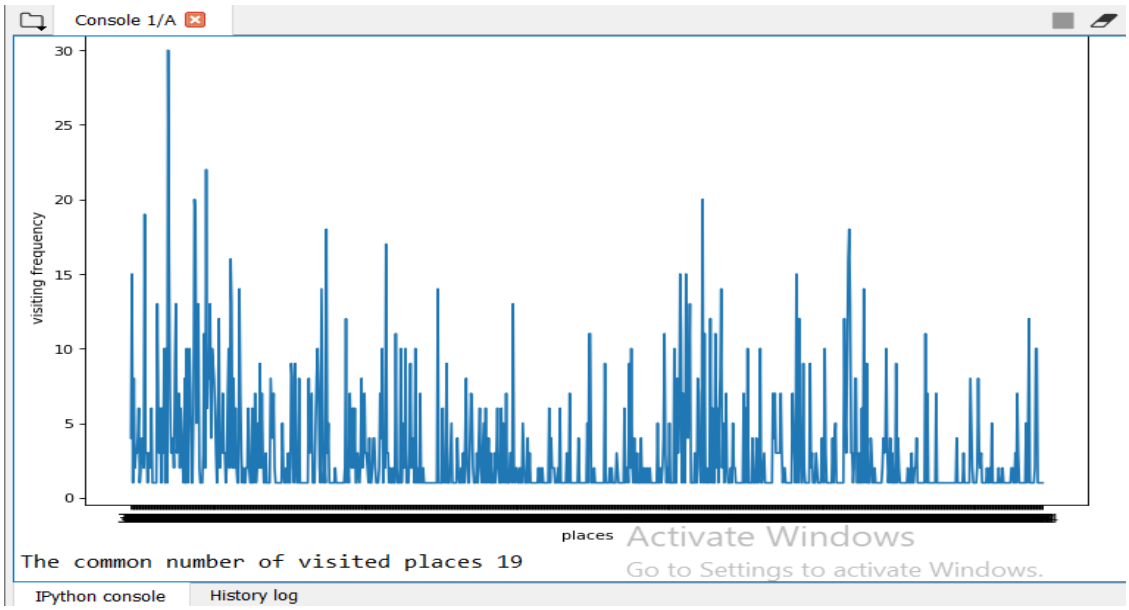
# 3.2-What's the common number of visited places?!

**We want to calculate the average of places visited by each user , then we need to count the sum of visited places by users and divide them by users number .**

```
for i in range(len(user_data.id)) :
    for j in places :
        if str(user_data.tower_id[i]) in j :
            if str(user_data.id[i]) in temp2 :
                if j not in temp2[str(user_data.id[i])] :
                    temp2[str(user_data.id[i])].append(j)
                    common = common + 1
            else :
                temp2[str(user_data.id[i])] = [j]
                common = common + 1


average = common / user_data.id[len(user_data)]
print ("The common number of visited places" , int(average))
```

**So, the final output is :**



# 4- How to categorize peoples with same habits?!

**To categorize people we need to know the users who visit the same places**

```
for i in range(1,len(result)) :
    if str(result[i]) not in places :
        places[str(result[i])] = [tower_data.ID[i]]
        groups[str(result[i])] = []
    else:
        places[str(result[i])].append(tower_data.ID[i])

for i in range(len(user_data.tower_id)) :
    for j in places :
        if str(user_data.tower_id[i]) in j :
            if j in visited :
                if  str(user_data.id[i]) not in temp :
                    temp[str(user_data.id[i])] = [user_data.day [i]]
                    visited[j] = visited[j] + 1
                    groups[j].append(user_data.id[i])
                elif user_data.day[i] not in temp[str(user_data.id[i])] :
                    temp[str(user_data.id[i])].append(user_data.day[i])
                    visited[j] = visited[j] + 1
            else :
                visited[j] = 1
                groups[j].append(user_data.id[i])
```

## Then we plot the output and cluster them with different colors

```
j=0

plt.figure(figsize=(11,7), dpi=80)
plt.xlabel("places")
plt.ylabel("users")
for k, v in groups.items():
    for i in range (len(v)):
        plt.scatter(int(k),v[i],c=color[j])
    j+=1
    if(j==len(color)):
        j=0
```

## So , the final output will be :-