```
import time
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
```

Part a

```
dist_mat=pd.read_csv("/content/TexasCityDistanceMatrix.xls")
keys=pd.read_csv("/content/keys.csv").to_numpy()
#keys = pd.ExcelFile(r"/content/Keys.xls")

print(dist_mat.head())

#remove the index of the cities
dist_mat2=dist_mat.drop(["Place"], axis=1)
ndist_mat2=dist_mat2.to_numpy()
n_dist_z=np.nan_to_num(ndist_mat2)
sdist_mat=csr_matrix(dist_mat2.fillna(0))

print("Maximum distance", sdist_mat.max())


print("Number of elements greater than 99.99971353 are:",len(n_dist_z[n_dist_z>99.99971353]))
```

```
   Place   V1         V2   V3   V4         V5         V6  V7  V8  V9  ...  \
0      1  0.0        NaN  NaN  NaN        NaN  75.440971 NaN NaN NaN  ...
1      2  NaN   0.000000  NaN  NaN  90.673319        NaN NaN NaN NaN  ...
2      3  NaN        NaN  0.0  NaN        NaN        NaN NaN NaN NaN  ...
3      4  NaN        NaN  NaN  0.0        NaN        NaN NaN NaN NaN  ...
4      5  NaN  90.673319  NaN  NaN   0.000000        NaN NaN NaN NaN  ...

       V1743  V1744  V1745  V1746      V1747      V1748      V1749      V1750  \
0  85.953325    NaN    NaN    NaN        NaN        NaN        NaN        NaN
1        NaN    NaN    NaN    NaN        NaN        NaN        NaN        NaN
2        NaN    NaN    NaN    NaN        NaN        NaN        NaN        NaN
3        NaN    NaN    NaN    NaN  37.771591  70.901728  37.784807  15.815951
4        NaN    NaN    NaN    NaN        NaN        NaN        NaN        NaN

   V1751  V1752
0    NaN    NaN
1    NaN    NaN
2    NaN    NaN
3    NaN    NaN
4    NaN    NaN

[5 rows x 1753 columns]
Maximum distance 99.99971353
Number of elements greater than 99.99971353 are: 0
```

Answer

Maximum distance in the matrix : 99.99971353

If we are interested in a matrix with far locations, we can infer that if we get to know the unknown values and store only the values greater to the maximum distance we have found, we will have a sparse matrix because it will be uncommon to have locations further than 99.99971353

Part b

```
indexk=0
for i in range(1,keys.shape[0]):
  if "Katy" in str(keys[i]):
    indexk=i

max_dist_katy=np.argmax(np.nan_to_num(ndist_mat2[indexk]))
max_num_distance_katy=ndist_mat2[indexk][max_dist_katy]
```

```
print("Number of elements greater", max_num_distance_katy, "are", len(ndist_mat2[indexk][ndist_mat2[indexk]>max_num_distance_katy
print("Maximum Katy distance is to ", keys[max_dist_katy])


Nans_huge=np.nan_to_num(ndist_mat2,nan=10000)
Nans_huge[indexk][indexk]=10000


index_min_Nans_huge=np.argmin(Nans_huge[indexk])

print(index_min_Nans_huge)

print("Closest place  to Katy city is :" , keys[index_min_Nans_huge])
print("Closest distance  to Katy city is :" , Nans_huge[indexk][index_min_Nans_huge])


index_fmt=0

for i in range(1,keys.shape[0]):
  if "Flower Mound town" in str(keys[i]):
    index_fmt=i

#All nan values made them huge , so that they are not converted to zero
# because we want all those values lower 25 miles
no_nan_fmd=np.nan_to_num(ndist_mat2[index_fmt],nan=1000)
print(no_nan_fmd)
print("Number of locations within 25 miles of Flower Mound town are ",len(no_nan_fmd[no_nan_fmd<=25])-1)
```

```
Number of elements greater 99.60808349 are 0
Maximum Katy distance is to  ["1249,Port O'Connor CDP"]
276
Closest place  to Katy city is : ['277,Cinco Ranch CDP']
Closest distance  to Katy city is : 5.337001509
[  79.67487195 1000.         1000.         ... 1000.         1000.
 1000.        ]
Number of locations within 25 miles of Flower Mound town are  75
```

Answer

Closest location to Katy city is to Cinco Ranch CDP

Number of locations within 25 miles of Flower Mound town are 75

Part c

```
#Not sparse n_dist_z

#sparse sdist_mat

times_sum_all=[]
for i in range (1000):
  start = time.time()
  np.sum(n_dist_z)
  end = time.time()
  times_sum_all.append(end - start)

average_sum_all=sum(times_sum_all)/len(times_sum_all)
print("Average time for summing all elements of the matrix ",average_sum_all )
```

```
Average time for summing all elements of the matrix  0.007972248077392578
```

Answer

In order to get a more accurate result, I take the average over 1000 repetitions of summing all values of the matrix.

The average time for summing all elements of the matrix 0.007972248077392578

Part d

```
dist_mat_d=pd.read_csv("/content/TexasCityDistanceMatrix.xls")

#remove the index of the cities
dist_mat_d=dist_mat_d.drop(["Place"], axis=1)

ndist_mat_d=dist_mat_d.to_numpy()
n_dist_wNan=np.nan_to_num(ndist_mat_d)

times_sum_all_d=[]
for i in range(n_dist_wNan.shape[0]):
  for j in range(n_dist_wNan.shape[1]):
    if n_dist_wNan[i][j]>50:
      n_dist_wNan[i][j]=0

for i in range (1000):
  start3 = time.time()
  np.sum(n_dist_wNan)
  end3 = time.time()
  times_sum_all_d.append(end3 - start3)
average_sum_all_d=sum(times_sum_all_d)/len(times_sum_all_d)


print("Average time for summing all elements of the matrix with values greater than 50 and  nans set to 0 is ", average_sum_all_d
```

```
Average time for summing all elements of the matrix with values greater than 50 and  nans set to 0 is  0.0017164371013641357
```

```
print((1-average_sum_all_d/average_sum_all)*100)
```

```
78.46984834514184
```

Answer

In order to get a more accurate result, I take the average over 1000 repetitions of summing all values of the matrix.

Average time for summing all elements of the matrix with values greater than 50 and nans set to 0 is 0.0017164371013641357

The time when setting all values greater than 50 and nans to zero was much smaller than taking the sum of all values. Specifically it was 78.46984834514184% smaller than just summing all values. Making many elements as zero simplifies the computation, so resulting a faster operation

Part e

```
dist_mat_e=pd.read_csv("/content/TexasCityDistanceMatrix.xls")
ndist_mat_e=dist_mat_e.to_numpy()


for i in range(ndist_mat_e.shape[0]):
  for j in range(ndist_mat_e.shape[1]):
    if ndist_mat_e[i][j]==0:
      ndist_mat_e[i][j]=-2

n_diste_wNan=np.nan_to_num(ndist_mat_e)



times_sum_all_e=[]


for i in range (1000):
  start4 = time.time()
  np.sum(spartse_e)
  end4 = time.time()
  times_sum_all_e.append(end4 - start4)
average_sum_all_e=sum(times_sum_all_e)/len(times_sum_all_e)
```

```
print("Average time for summing all elements of the matrix ",average_sum_all_e )
```

```
Average time for summing all elements of the matrix  0.0007139577865600586
```

```
p_smaller_d=(1- average_sum_all_e/average_sum_all_d)*100
p_smaller_c=(1- average_sum_all_e/average_sum_all)*100

print("Percentage smaller than d", p_smaller_d)

print("Percentage smaller than c", p_smaller_c)
```

```
Percentage smaller than d 58.40466359107236
Percentage smaller than c 91.04446098980945
```

Answer

In order to get a more accurate result, I take the average over 1000 repetitions of summing all values of the matrix.

Average time for summing all elements of the matrix not considering all Nan values and keeping all original zeros, was 0.0007139577865600586

There is a huge difference compared to the last two results. Specifically comparing to the sum done in part c , the time for summing all values now was 91.04446098980945% faster and compared to part d was 58.40466359107236% faster .

The sum is now much faster because we are ignoring a great amount of the matrix elements. Given that the matrix is sparse with many Nans elements, by ignoring them the computing time is much faster.

Part f

Answer

We can exploit symmetry by just storing elements a_{ij} such that $j > i$. By doing this we are counting just once the distances because we are storing the upper elements of the matrix . If we are interested in summing all elements of the matrix (i.e. double counting the distances as we have just done before), we just have to sum all stored a_{ij} such that $j > i$ and multiply by two.

In the following piece of code is shown that indeed we are getting the same sum if we just consider all values a_{ij} such that $j > i$. By this way we are saving just saving $N(N-1)/2$ elements instead of N^2

```
dist_mat_f=pd.read_csv("/content/TexasCityDistanceMatrix.xls")
ndist_mat_f=dist_mat_f.to_numpy()
original=np.nan_to_num(ndist_mat_f)

mod=np.nan_to_num(ndist_mat_f)

sum_mod=0
sum_original=0
for i in range(mod.shape[0]):
  for j in range(mod.shape[1]):
    sum_original=sum_original+1
    if j>i:
      sum_mod=sum_mod+1

print("Sum using all the elements of the matrix ( double counting the distances ) : ", sum_original)

print("Sum just storing the upper triangular matrix, the elements j>i : ", sum_mod*2)
```

```
Sum using all the elements of the matrix ( double counting the distances ) :  3071256
Sum just storing the upper triangular matrix, the elements j>i :  3071256

"""
```

0s completed at 3:47 PM