```
from scipy.sparse import csr_matrix
import sparse_dot_topn.sparse_dot_topn as ct
from sklearn.feature_extraction.text import TfidfVectorizer


import pandas as pd


import numpy as np


!pip install sparse_dot_topn
```

```
Collecting sparse_dot_topn
  Downloading sparse_dot_topn-0.3.1.tar.gz (17 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: setuptools>=42 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cython>=0.29.15 in /usr/local/lib/python3.7/dis
Requirement already satisfied: scipy>=1.2.3 in /usr/local/lib/python3.7/dist-p
Building wheels for collected packages: sparse-dot-topn
  Building wheel for sparse-dot-topn (PEP 517) ... done
  Created wheel for sparse-dot-topn: filename=sparse_dot_topn-0.3.1-cp37-cp37m
  Stored in directory: /root/.cache/pip/wheels/3b/3e/02/4ee8cb28ed8b608d530bc4
Successfully built sparse-dot-topn
Installing collected packages: sparse-dot-topn
Successfully installed sparse-dot-topn-0.3.1
```

```
df = pd.read_csv('/content/drive/MyDrive/Seattle Hotels Duplicates.csv', encoding="
```

```
df.head(3)
```

```
df.name.value_counts()
```

```
Hilton Garden Inn Seattle Downtown                    2
Ace Hotel Seattle                                     2
citizenM Seattle South Lake Union hotel               2
Hyatt Regency Lake Washington At SeattleS Southport   2
Quality Inn & Suites Seattle Center                   2
                                                     ..
DoubleTree by Hilton Hotel Seattle Airport            1
Radisson Hotel Seattle Airport                        1
Crowne Plaza Seattle Airport                          1
Hilton Seattle Airport & Conference Center            1
```

```
MarQueen Hotel                                                  1
Name: name, Length: 155, dtype: int64
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
df.loc[df['name'] == 'Roy Street Commons']
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
df['name_address'] = df['name'] + ' ' + df['address']
name_address = df['name_address']
vectorizer = TfidfVectorizer(analyzer = 'char', ngram_range=(1, 4), sublinear_tf=Tr
tf_idf_matrix = vectorizer.fit_transform(name_address)
```

```
tf_idf_matrix
```

```
<168x4957 sparse matrix of type '<class 'numpy.float64'>'
        with 32557 stored elements in Compressed Sparse Row format>
```

```
def awesome_cossim_top(A, B, ntop, lower_bound=0):

    A = A.tocsr()
    B = B.tocsr()
    M, _ = A.shape
    _, N = B.shape

    idx_dtype = np.int32

    nnz_max = M*ntop

    indptr = np.zeros(M+1, dtype=idx_dtype)
    indices = np.zeros(nnz_max, dtype=idx_dtype)
    data = np.zeros(nnz_max, dtype=A.dtype)

    ct.sparse_dot_topn(
        M, N, np.asarray(A.indptr, dtype=idx_dtype),
        np.asarray(A.indices, dtype=idx_dtype),
        A.data,
        np.asarray(B.indptr, dtype=idx_dtype),
        np.asarray(B.indices, dtype=idx_dtype),
```

```
            B.data,
            ntop,
            lower_bound,
            indptr, indices, data)

        return csr_matrix((data,indices,indptr),shape=(M,N))

    matches = awesome_cossim_top(tf_idf_matrix, tf_idf_matrix.transpose(), 5)

    matches
```

```
     <168x168 sparse matrix of type '<class 'numpy.float64'>'
             with 840 stored elements in Compressed Sparse Row format>
```

```
    def get_matches_df(sparse_matrix, name_vector, top=840):
        non_zeros = sparse_matrix.nonzero()

        sparserows = non_zeros[0]
        sparsecols = non_zeros[1]

        if top:
            nr_matches = top
        else:
            nr_matches = sparsecols.size

        left_side = np.empty([nr_matches], dtype=object)
        right_side = np.empty([nr_matches], dtype=object)
        similairity = np.zeros(nr_matches)

        for index in range(0, nr_matches):
            left_side[index] = name_vector[sparserows[index]]
            right_side[index] = name_vector[sparsecols[index]]
            similairity[index] = sparse_matrix.data[index]

        return pd.DataFrame({'left_side': left_side,
                             'right_side': right_side,
                              'similarity': similairity})

    matches_df = get_matches_df(matches, name_address)


    matches_df[matches_df['similarity'] < 0.99999].sort_values(by=['similarity'], ascen
```

```
matches_df[matches_df['similarity'] < 0.50].right_side.nunique()
```

```
150
```

✓ 0s completed at 1:23 PM