Assignmenr 6 Text Analytics: String Distances

by Prudvi Kamtam

Step 0: Load packages and data

In [ ]:
```
import pandas as pd
```

In [ ]:
```
babynames_df = pd.read_csv("babyNamesUSYOB-full.csv")
babynames_df.head()
```

Out[ ]:

	YearOfBirth	Name	Sex	Number
0	1880	Mary	F	7065
1	1880	Anna	F	2604
2	1880	Emma	F	2003
3	1880	Elizabeth	F	1939
4	1880	Minnie	F	1746

Step 0.1: Define distance definitions

In [ ]:
```
from collections import Counter
import numpy as np
import math
import re

def jaccard(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection) / union

def get_cosine(text1, text2):

    vec1 = text_to_vector(text1)
    vec2 = text_to_vector(text2)

    intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])

    sum1 = sum([vec1[x] ** 2 for x in list(vec1.keys())])
    sum2 = sum([vec2[x] ** 2 for x in list(vec2.keys())])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)

    if not denominator:
```

```
        return 0.0
    else:
        return float(numerator) / denominator

def text_to_vector(text):
    CHARACTER = re.compile(r".")
    chars = CHARACTER.findall(text)
    return Counter(chars)

def manhattan_distance(s1, s2):
    return sum(abs(ord(a) - ord(b)) for a, b in zip(s1, s2))
```

In [ ]:

```
import pandas as pd
from scipy.spatial.distance import hamming, cosine, euclidean, cityblock
from Levenshtein import distance as levenshtein_distance
from pyjarowinkler import distance as jaro_winkler_distance
from jaro import jaro_metric, jaro_winkler_metric
import numpy as np
from numpy.linalg import norm
from jellyfish import jaro_distance, nysiis, metaphone, soundex, match_ratir
import jellyfish

# Define a function that computes string distances
def compute_distance(s1, s2, distance_measure):
    if distance_measure == 'hamming':
        return sum(c1 != c2 for c1, c2 in zip(s1, s2))
    elif distance_measure == 'levenshtein':
        return levenshtein_distance(s1, s2)
    elif distance_measure == 'jaro': # https://pypi.org/project/jaro-winkler
        return jaro_metric(s1, s2)
    elif distance_measure == 'jaro_winkler': # https://pypi.org/project/jarc
        return jaro_winkler_metric(s1, s2)
    elif distance_measure == 'cosine': # https://www.geeksforgeeks.org/how-t
        # return cosine(s1, s2)
        return get_cosine(s1,s2)
    elif distance_measure == 'manhattan':
        # return cityblock(s1, s2)
        return manhattan_distance(s1, s2)
    elif distance_measure == 'jaccard':
        return jaccard(s1, s2)
    elif distance_measure == 'sorensen_dice':
        return 1 - (2 * len(set(s1) & set(s2))) / (len(s1) + len(s2))
    elif distance_measure == 'dice':
        return 1 - jellyfish.damerau_levenshtein_distance(s1, s2) / max(len(

# Define the input string
s = pd.Series(['apple', 'banana', 'cherry', 'orange', 'mango', 'pear'])
strings = ['appel' , 'bnana', 'cheeery', 'orngae', 'mengo', 'ear']
distances = ['hamming', 'levenshtein', 'jaro', 'jaro_winkler', 'cosine', 'ma
# distances = ['jaro', 'jaro_winkler']

for distance in distances:
    print(f'\nDistance : {distance}')
    for input_string in strings:
        print(input_string, end=': ')
        distances = pd.Series([compute_distance(input_string, x, distance) f
```

```
    # print(distances)
    # print(s.loc[distances.sort_values().head().index])
    # Sort the Series by the distances and print the resulting Series
    if distance in ['hamming', 'levenshtein', 'manhattan', 'sorensen_dic
        print(list(s.loc[distances.sort_values().head().index])[0])
    else:
        print(list(s.loc[distances.sort_values(ascending=False).head().i
    # print(distances.sort_values())
```

```
Distance : hamming
appel: apple
bnana: orange
cheeery: cherry
orngae: orange
mengo: mango
ear: banana

Distance : levenshtein
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear

Distance : jaro
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear

Distance : jaro_winkler
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear

Distance : cosine
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear

Distance : manhattan
appel: apple
bnana: orange
cheeery: apple
orngae: orange
mengo: mango
ear: banana

Distance : jaccard
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear
```

```
Distance : sorensen_dice
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear

Distance : dice
appel: apple
bnana: banana
cheeery: cherry
orngae: orange
mengo: mango
ear: pear
```

Step 1. Find the ten names in the babynames data set that are the most similar to your first name

In [ ]:

```python
import pandas as pd
import numpy as np
from scipy.spatial.distance import hamming, cosine, euclidean, cityblock
from Levenshtein import distance as levenshtein_distance
from pyjarowinkler import distance as jaro_winkler_distance
from jaro import jaro_metric, jaro_winkler_metric
from numpy.linalg import norm
from jellyfish import jaro_distance, nysiis, metaphone, soundex, match_ratir
import jellyfish

s = babynames_df["Name"].drop_duplicates()

# Define the input string
name = 'Elon'

def get_similar_names(name='Steve', s=babynames_df["Name"].drop_duplicates()
    results = {}
    distances = ['hamming', 'levenshtein', 'jaro', 'jaro_winkler', 'cosine',
    for i, distance in enumerate(distances, 1):
        print(f'\nDistance {i}: {distance}; Given Name: {name}')
        # Compute the distance between each element of the Series and the ir
        distances = s.apply(lambda x: compute_distance(name, x, distance))

        # result = list(s.loc[distances.sort_values(ascending=False).index])
        if distance in ['hamming', 'levenshtein', 'manhattan', 'sorensen_dic
            result = list(s.loc[distances.sort_values().index])
        else:
            result = list(s.loc[distances.sort_values(ascending=False).index

        # Remove the give name from the list
        results[distance] = [x for x in result if x != name][:10]
        # Sort the Series by the distances and print the resulting Series
        print(results[distance])
    return results

similar_names = get_similar_names(name='Steve')
```

```
Distance 1: hamming; Given Name: Steve
['Steveland', 'Stevee', 'Stevenmichael', 'Stevenson', 'Stevens', 'Stevena',
'Stevephen', 'Stevette', 'Stevey', 'St']

Distance 2: levenshtein; Given Name: Steve
['Stevie', 'Stevn', 'Stevee', 'Stevy', 'Steva', 'Seve', 'Steven', 'Steeve',
'Steave', 'Stevey']

Distance 3: jaro; Given Name: Steve
['Steven', 'Stevey', 'Steave', 'Stevee', 'Stevie', 'Seve', 'Stevens', 'Steave
n', 'Stevena', 'Stevenn']

Distance 4: jaro_winkler; Given Name: Steve
['Stevee', 'Stevie', 'Steven', 'Stevey', 'Steave', 'Stevena', 'Stevens', 'Ste
venn', 'Seve', 'Steaven']

Distance 5: cosine; Given Name: Steve
['Stevee', 'Steeve', 'Shevette', 'Steeven', 'Steevie', 'Steave', 'Stevey', 'S
teven', 'Stevie', 'Severt']

Distance 6: manhattan; Given Name: Steve
['Steveson', 'Steven', 'Stevenmichael', 'St', 'Stevee', 'Steveland', 'Stevett
e', 'Stevenray', 'Stevey', 'Stevenson']

Distance 7: jaccard; Given Name: Steve
['Stven', 'Stevi', 'Steva', 'Stevy', 'Stevn', 'Set', 'Stevee', 'Sivert', 'Sti
ven', 'Stevon']

Distance 8: sorensen_dice; Given Name: Steve
['Stevy', 'Stevi', 'Stven', 'Stevn', 'Steva', 'Set', 'Stover', 'Stevee', 'Sev
ert', 'Stevan']

Distance 9: dice; Given Name: Steve
['Stevee', 'Stevey', 'Steeve', 'Steven', 'Stevie', 'Steave', 'Steva', 'Stev
n', 'Stevy', 'Seve']
```

In [ ]:

```
babynames_df[babynames_df['Name'] == 'Mary']['Number']
```

Out[ ]:

```
0          7065
1273         27
2000       6919
3238         29
3935       8148
           ...
1756827       6
1759451    2639
1792680    2624
1824999       5
1825860    2602
Name: Number, Length: 266, dtype: int64
```

In [ ]:

```
def get_time_series(name='Steve', sex='MF'):
    df = babynames_df.copy()
    if sex =='MF':
        df = df.groupby(['YearOfBirth', 'Name'], as_index=False)['Number'].s
        return df.loc[(df['Name'] == name), ['Name', 'YearOfBirth', 'Number'
```

```
    else:
        return df.loc[(df['Name'] == name) & (df['Sex'] == sex), ['Name', 'Y
```

In [ ]:
```
get_time_series(name='Pete', sex='MF')#[0]
```

Out[ ]:

	Name	YearOfBirth	Number
1502	Pete	1880	50
3344	Pete	1881	57
5305	Pete	1882	60
7306	Pete	1883	43
9382	Pete	1884	81
...	...	...	...
1564842	Pete	2011	66
1596103	Pete	2012	64
1627054	Pete	2013	63
1657781	Pete	2014	68
1688316	Pete	2015	62

136 rows × 3 columns

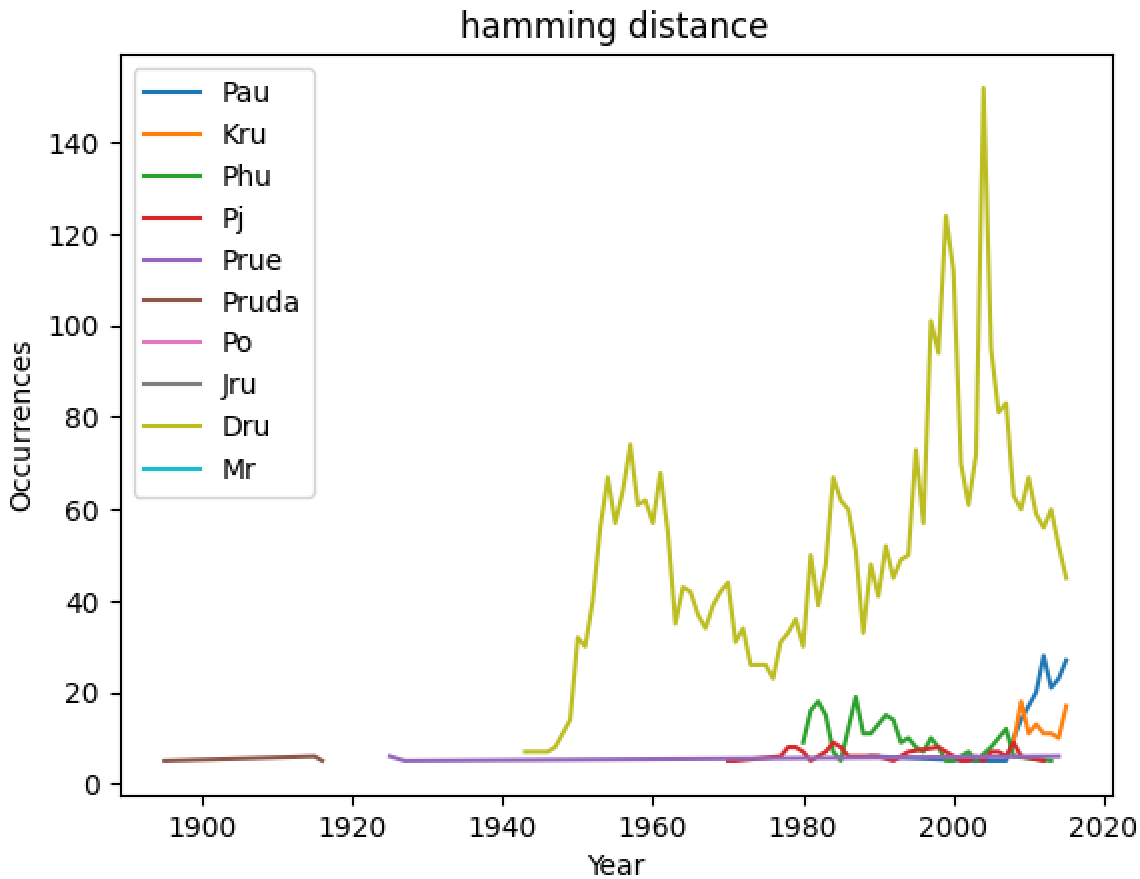
Step 2: Plot the names as times series by year. Put the string distance used in the title of the plot

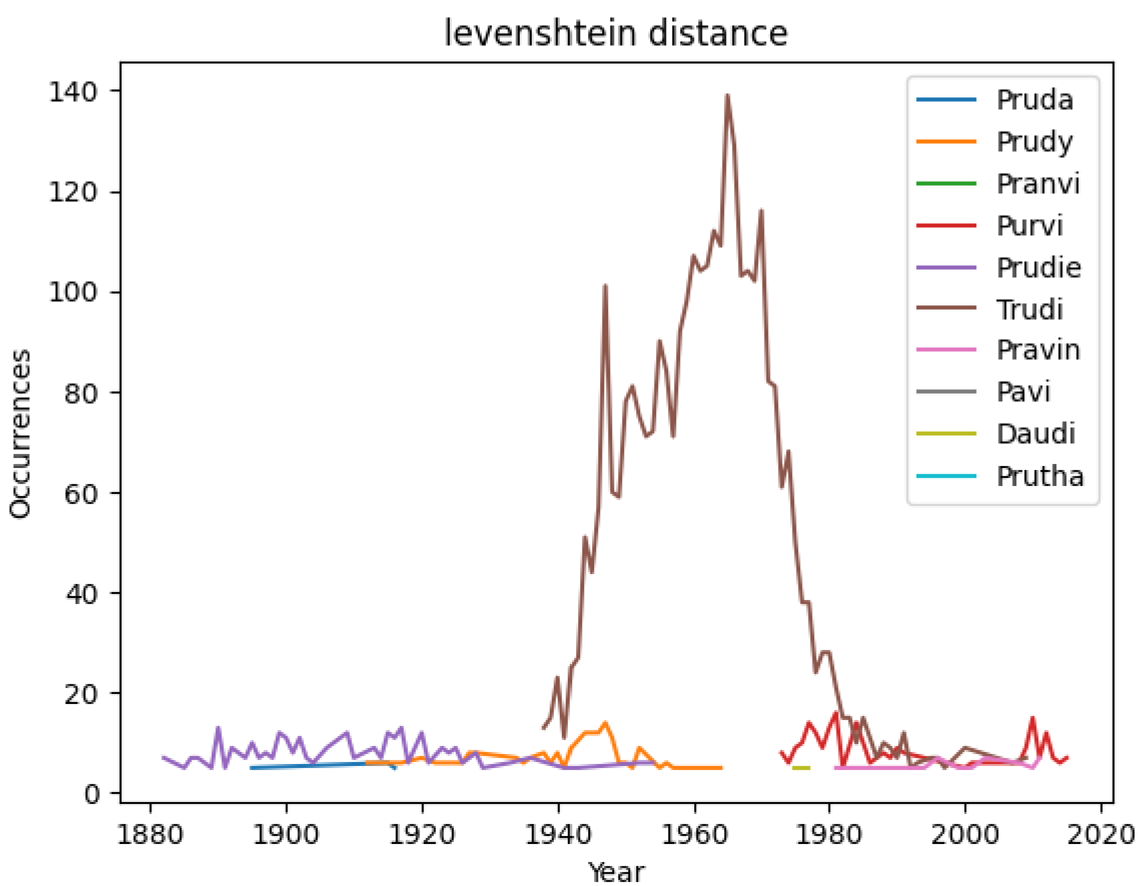
In [ ]:
```
import matplotlib.pyplot as plt

def plot_occurrences_by_year(dfs):
    # Create a time series plot of the number of occurrences by year for eac
    for _, df in dfs:
        try:
            plt.plot(df['YearOfBirth'], df['Number'], label=list(df['Name'])
        except:
            continue

    # Set the title and axis labels
    plt.title(f'{dfs[0][0]} distance')
    plt.xlabel('Year')
    plt.ylabel('Occurrences')
    plt.legend()

    # Show the plot
    plt.show()
```

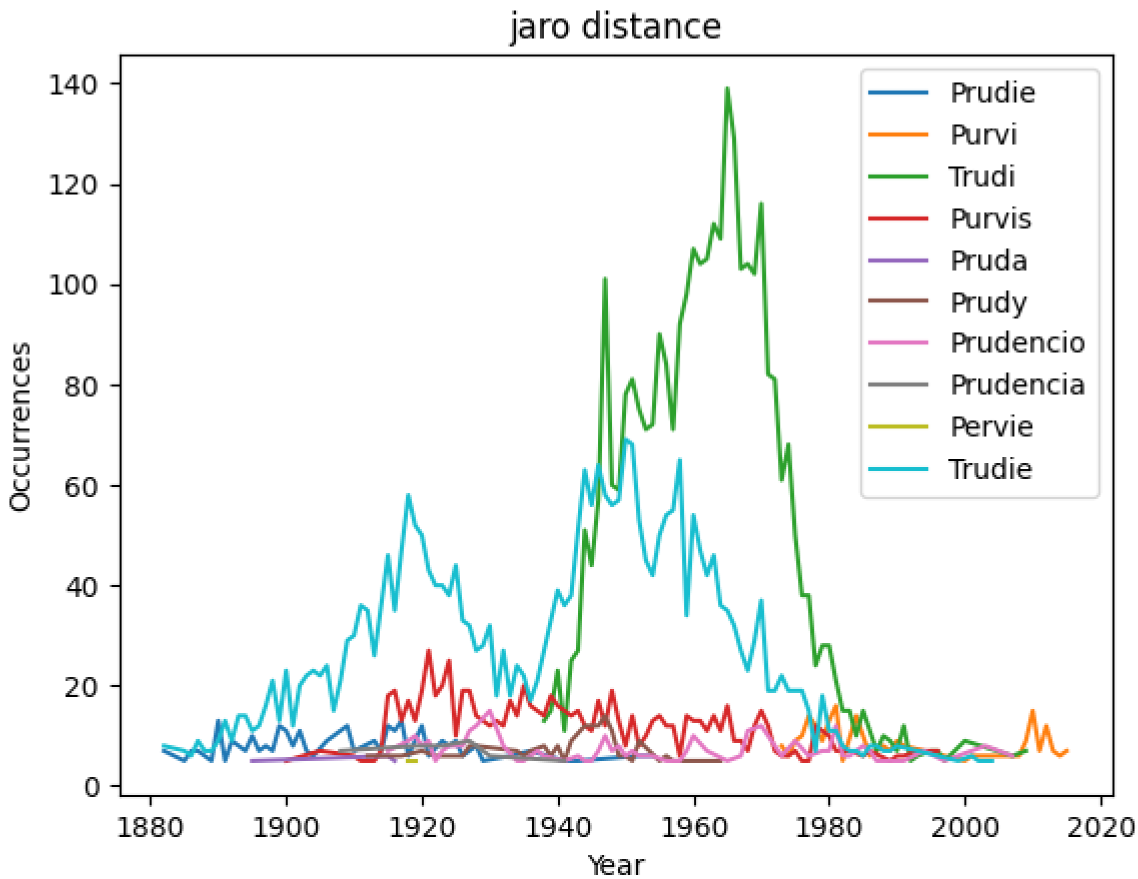
In [ ]:
```
distances = ['hamming', 'levenshtein', 'jaro', 'jaro_winkler', 'cosine', 'ma

similar_names = get_similar_names(name='Prudvi')
```

```python
for distance in distances:
    all_dfs = []
    for name in similar_names[distance]:
        all_dfs.append([distance, get_time_series(name=name)])

    plot_occurrences_by_year(all_dfs)
```

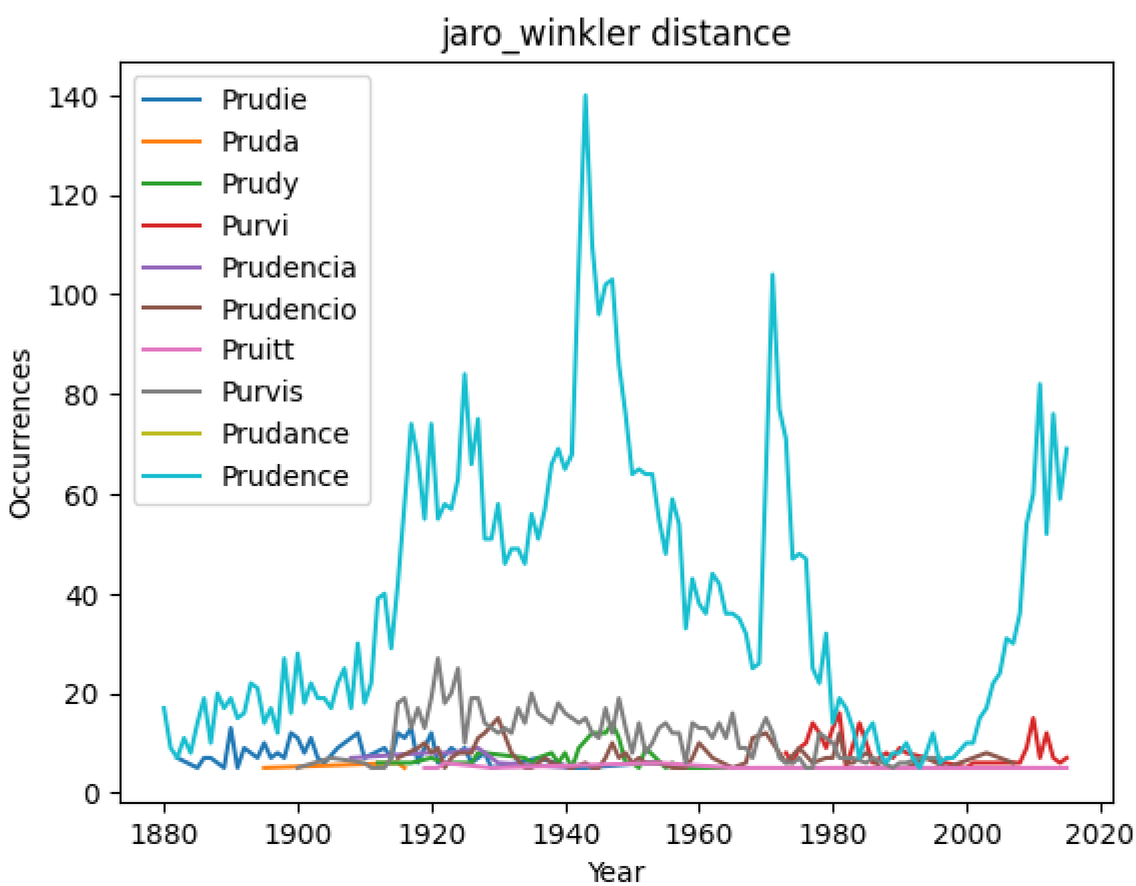
```
Distance 1: hamming; Given Name: Prudvi
['Pau', 'Kru', 'Phu', 'Pj', 'Prue', 'Pruda', 'Po', 'Jru', 'Dru', 'Mr']

Distance 2: levenshtein; Given Name: Prudvi
['Pruda', 'Prudy', 'Pranvi', 'Purvi', 'Prudie', 'Trudi', 'Pravin', 'Pavi', 'D
audi', 'Prutha']

Distance 3: jaro; Given Name: Prudvi
['Prudie', 'Purvi', 'Trudi', 'Purvis', 'Pruda', 'Prudy', 'Prudencio', 'Pruden
cia', 'Pervie', 'Trudie']

Distance 4: jaro_winkler; Given Name: Prudvi
['Prudie', 'Pruda', 'Prudy', 'Purvi', 'Prudencia', 'Prudencio', 'Pruitt', 'Pu
rvis', 'Prudance', 'Prudence']

Distance 5: cosine; Given Name: Prudvi
['Purvi', 'Prudie', 'Purvis', 'Gurvinder', 'Princedavid', 'Prudy', 'Audri',
'Orvid', 'Trudi', 'Suvir']

Distance 6: manhattan; Given Name: Prudvi
['Or', 'Prue', 'Prudy', 'Mr', 'Po', 'Tru', 'Tr', 'Ott', 'Orr', 'Kru']

Distance 7: jaccard; Given Name: Prudvi
['Purvi', 'Prudie', 'Purvis', 'Purva', 'Purdy', 'Vidur', 'Trudi', 'Pride', 'P
urav', 'Suvir']

Distance 8: sorensen_dice; Given Name: Prudvi
['Purvi', 'Purvis', 'Prudie', 'Audri', 'Prudy', 'Trudi', 'Purav', 'Pride', 'D
uvid', 'Orvid']

Distance 9: dice; Given Name: Prudvi
['Prudie', 'Purvi', 'Trudi', 'Prudy', 'Pruda', 'Pranvi', 'Pranavi', 'Paridh
i', 'Sridevi', 'Prithvi']
```


hamming distance
Pau
Kru
Phu
Pj
Prue
Pruda
Po
Jru
Dru
Mr
Occurrences
0
20
40
60
80
100
120
140
1900
1920
1940
1960
1980
2000
2020
Year

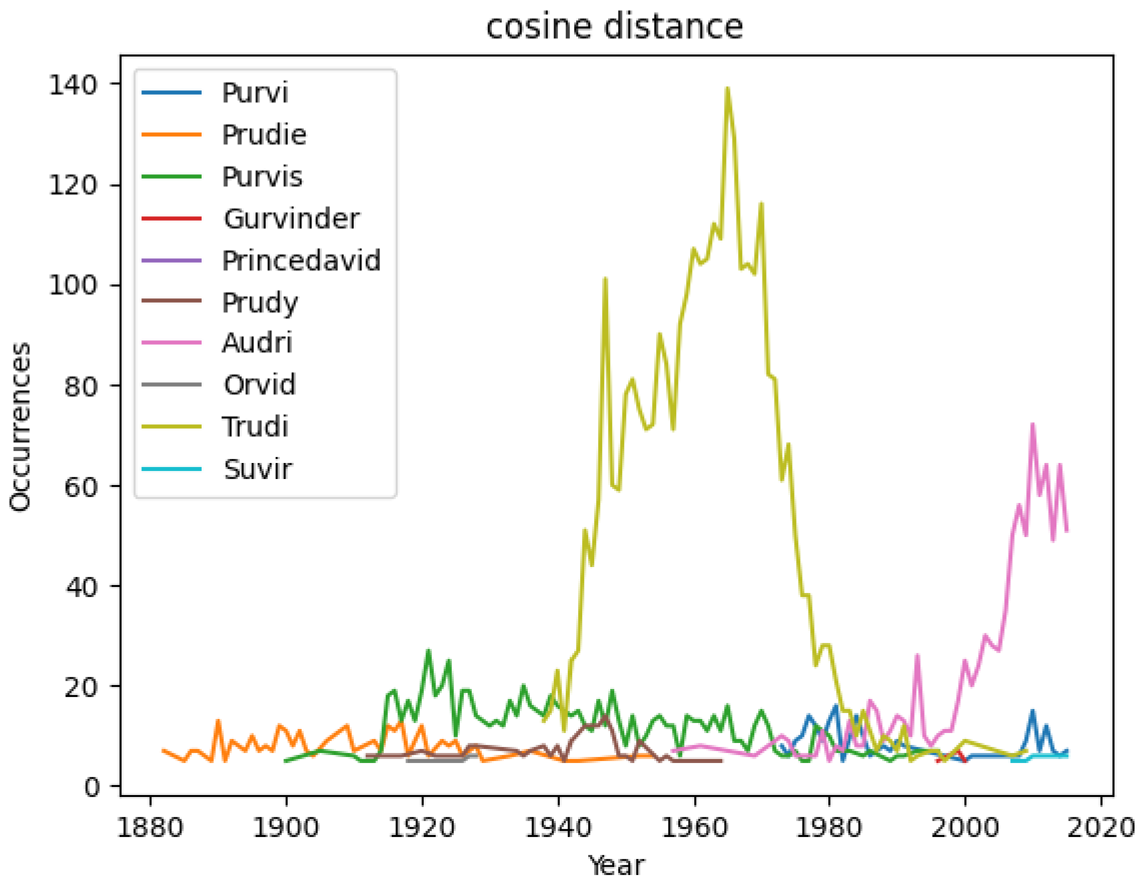

levenshtein distance
Pruda
Prudy
Pranvi
Purvi
Prudie
Trudi
Pravin
Pavi
Daudi
Prutha
Occurrences
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020
Year

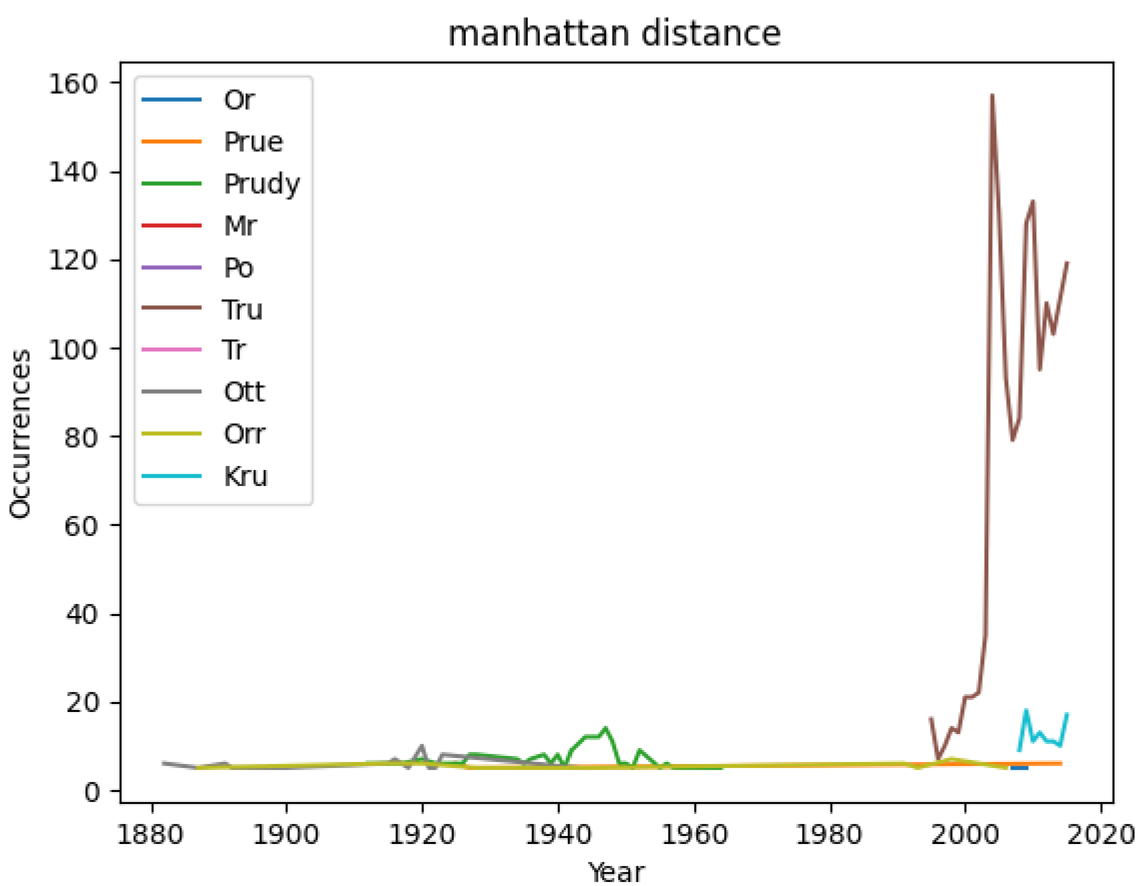


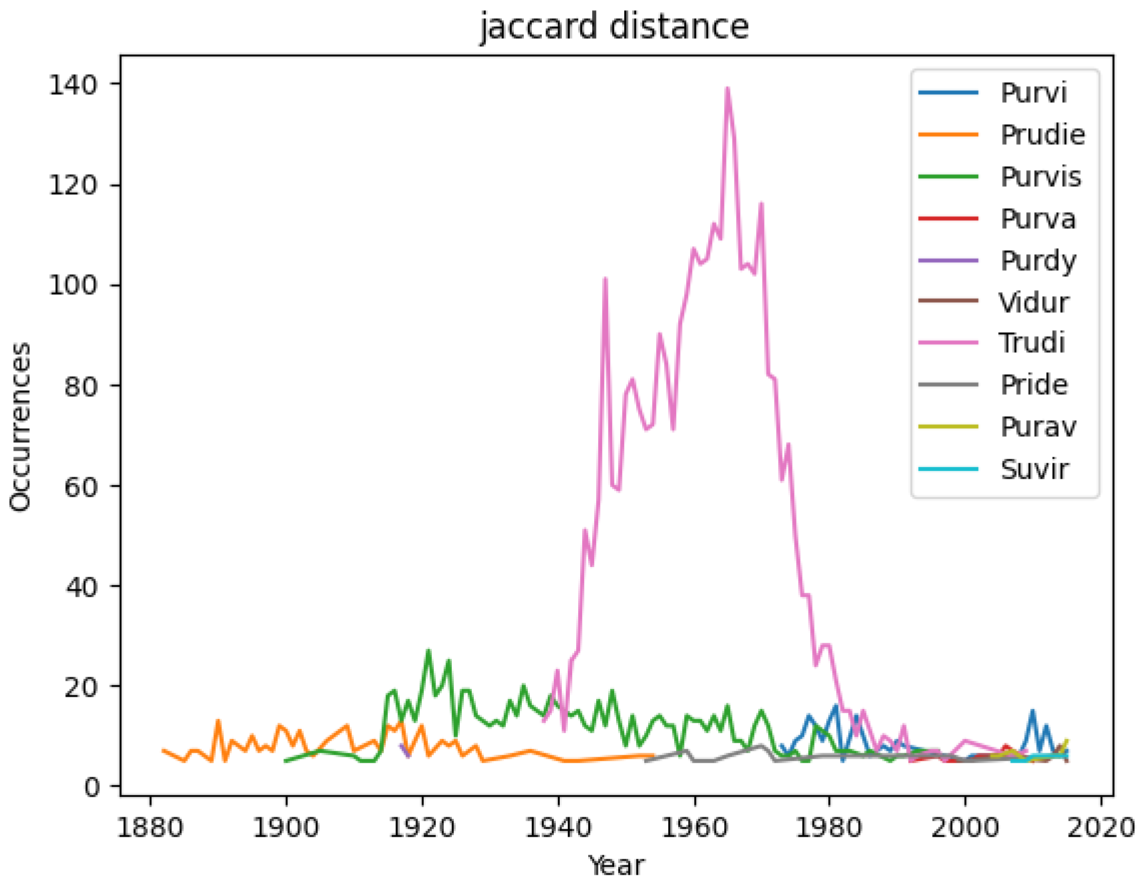
jaro distance
Occurrences
Year
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020
Prudie
Purvi
Trudi
Purvis
Pruda
Prudy
Prudencio
Prudencia
Pervie
Trudie

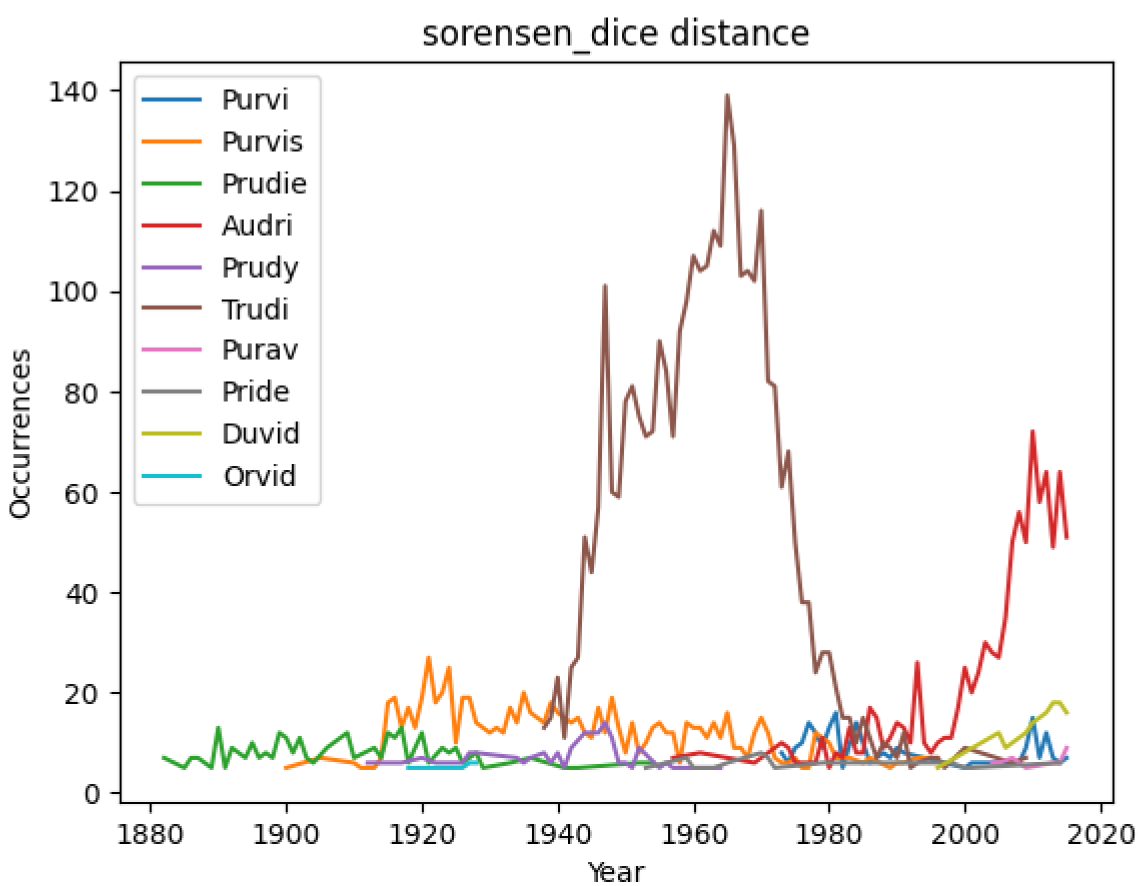


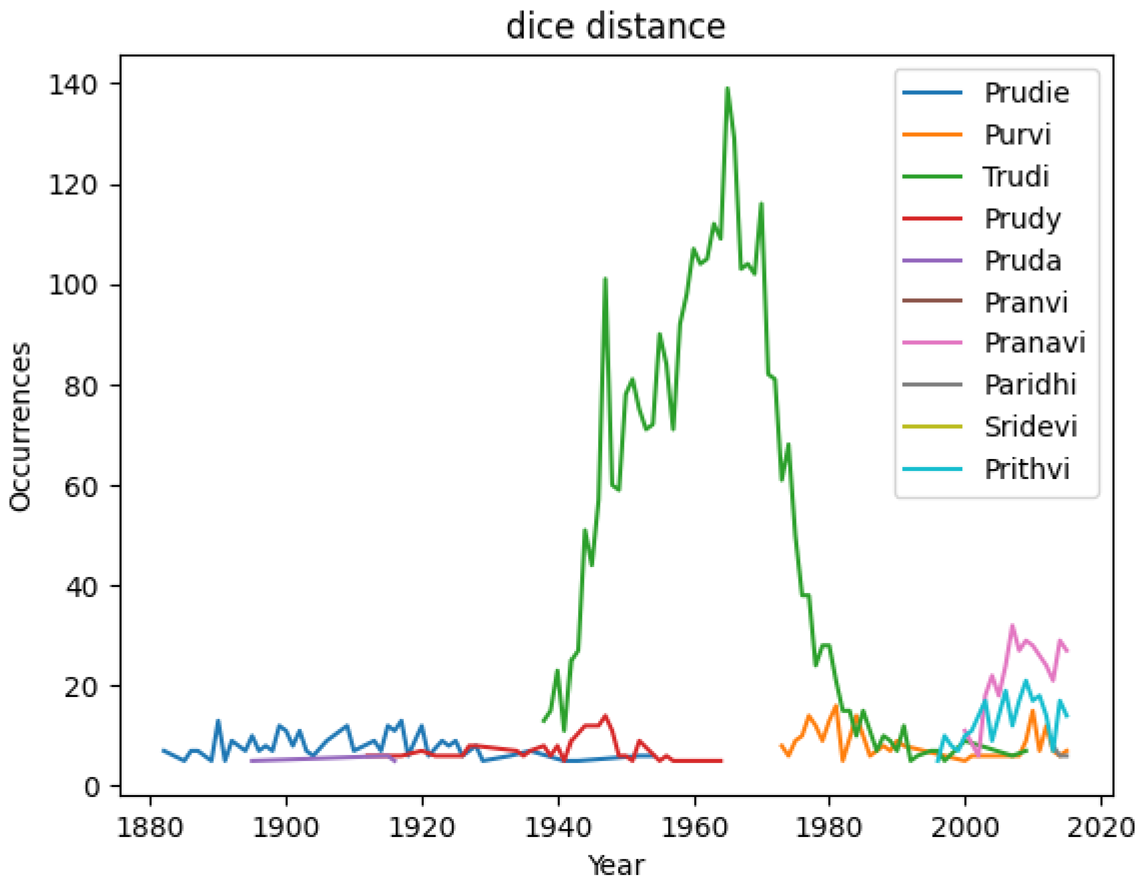
jaro_winkler distance
Occurrences
Year
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020
Prudie
Pruda
Prudy
Purvi
Prudencia
Prudencio
Pruitt
Purvis
Prudance
Prudence


cosine distance
Purvi
Prudie
Purvis
Gurvinder
Princedavid
Prudy
Audri
Orvid
Trudi
Suvir
Occurrences
Year
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020


manhattan distance
Or
Prue
Prudy
Mr
Po
Tru
Tr
Ott
Orr
Kru
Occurrences
Year
0
20
40
60
80
100
120
140
160
1880
1900
1920
1940
1960
1980
2000
2020


jaccard distance
Occurrences
Year
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020
Purvi
Prudie
Purvis
Purva
Purdy
Vidur
Trudi
Pride
Purav
Suvir


sorensen_dice distance
Occurrences
Year
0
20
40
60
80
100
120
140
1880
1900
1920
1940
1960
1980
2000
2020
Purvi
Purvis
Prudie
Audri
Prudy
Trudi
Purav
Pride
Duvid
Orvid



Step 3. Write a few sentences articulating the similarities and differences you notice about each metric.

1. It is apparent that my name "Prudvi" isn't a common name in the dataset. This might be because the dataset contains names mostly of US origin. However, it is interesting to note that some similar names, such as "Prudhvi" or "Praveen," do appear in the dataset.
2. It is to be noted that some string distance metrics give scores differently. For example, 'hamming', 'levenshtein', 'manhattan', 'sorensen_dice' metrics give the distance of the strings, meaning that the lower the value, the higher the similarity. On the other hand, all other metrics give the opposite, meaning that the higher the value, the higher the similarity.
3. Some metrics like levenshtein, jaro, jaro_winkler, jaccard, and dice give the best results. These metrics take into account the differences in character sequences and lengths between names and provide accurate similarity scores.
4. On the other hand, some metrics like hamming and manhattan were amongst the ones that gave poor results. These metrics do not take into account differences in string lengths, which may result in inaccurate similarity scores.
5. The time-series graphs of the distances usually show a higher occurrence of similar names from 1940 to 1980. This may be due to certain names being popular during those decades and then falling out of favor in later years.

6. Names like Trudy, Purvi, Prudie, Prudy, Prudie can be seen occurring more than once when using different metrics. This may be because these names have similar character sequences and are thus scored as more similar by certain string distance metrics.
7. Hamming and Manhattan distance have similar results, which are usually three-character names. This is because these metrics only measure the number of differences between characters in two strings and do not take into account differences in string length or character sequences.

In [ ]:
```

```