Lab 3: Search Terms with Pandas

Submitted By: Julian Singkham
Date: 01/07/2021

Abstract

The purpose of this lab is to familize ourselves with using panda data frames and functional programming for data analysis using our own dataset.

- The first objective was to derive names from a .txt file that contained rows of data with differing row lengths.
- The second objective was to compare the timings and memory used from this dataset to the Direct Supply dataset.

The data utilized in this lab is a character names list found in 617 movies by Cornell Movie Dialog Corpus. Due to the nature of name spelling, the filtering of non-alphabet characters and spellchecking was ommitted.

Parameters

In [1]:
```
from spellchecker import SpellChecker
import pattern.en
import csv
import pandas as pd
import sys

freq_dict = {}
freq_dict_spellchecked = []
```

Functions

Imports the movie_characters_metadata.txt file and creates a data frame of the second item of each row.

Param None **Return**:A data frame of the second item of each row of the csv

In [2]:
```
def import_file_df():
    df = pd.read_csv("movie_characters_metadata.txt", delim_whitespace=True,
                     names=['a', 'Names', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
    return df["Names"].to_frame()
```

Creates a frequency dictionary given a string list where the key is a string and the key-value is how many times the string appeared in the list.

Param input_list: String list
Return: A frequency dictionary

In [3]:
```
def list_to_freq_dict(input_list):
    freq_dict = {}
    for i in input_list:
        freq_dict[i] = input_list.count(i)
    return freq_dict
```

Creates a sorted frequency list given a frequency dictionary

Param freq_dict: Frequnecy dictionary
Return: A 2d list where the first row is frequency and the second row is the string

In [4]:
```
def sort_freq_dict(freq_dict):
    sorted_list = [(freq_dict[key], key) for key in freq_dict]
    sorted_list.sort()
    sorted_list.reverse()
    return sorted_list
```

Main

In [5]:
```
# Import csv to search term data frame
df = import_file_df()
```

In [6]:
```
%%time
# Thia section of code benchmarks the time it takes for the dictionary and li
# sorted frequency list of search terms
spellcheck_freq_dict = list_to_freq_dict(df["Names"].tolist())
spellcheck_freq_list = sort_freq_dict(spellcheck_freq_dict)
```

```
Wall time: 1.05 s
```

In [7]:
```
# Thia section of code benchmarks the time it takes for the data frame approa
# sorted frequency list of search terms
%time series_freq = df["Names"].value_counts(dropna=True)
series_freq.head(10)
```

```
Wall time: 7.8 ms
```

Out[7]:
```
MRS.     100
MR.       75
DR.       75
MAN       55
WOMAN     39
VOICE     39
THE       34
JACK      33
MARY      32
FRANK     32
Name: Names, dtype: int64
```

In [8]:
```
# This section of code benchmarks the size of the sorted frequency data frame
series_freq.memory_usage(deep=True)
```

Out[8]: 311362

In [9]:
```
# This section of code benchmarks the size of the sorted frequency list
sys.getsizeof(spellcheck_freq_list)
```

Out[9]: 38208

Conclusion

- Based on the fact that the characters dataset contains 9,035 items while the Direct Supply dataset contains 12,382 items, I believe there will be a linear relation between the two timings (27% faster). This is only valid for the data frame method as the list method does not contain empty values for the Direct Supply dataset. When compared to the list method of the Direct Supply dataset, 1407, the character dataset should take slightly longer to compute due to the time complexity of sort being n*log(n), where the difference in timing plateus very quickly.
- In terms of data size, the characters data frame and list should be vastly larger (3-4x) then the Direct Supply data frame due to larger string sizes and higher amount of unique terms. 3.17
- The characters dataset data frame method took 6.78ms while the Direct Supply data frame took 3.52ms. I suspect the reason for the this is due to the characters data et having larger string names than the Direct Supply dataset. This difference in word sizes should affect the timing. This difference in size will be discussed in the memory section. The characters dataset list took 3.31s while the Direct Supply dataset list took 2.14s. This difference alligns with my hypothesis and is supported by the difference in memory usage.
- The characters dataset data frame used 311,362B of data while the Direct Supply dataset data frame used 99,242B. Allthough I hypothesized the characters dataset would use more memory, I was off by a factor of 2 (4x vs 8x) This vast difference in memory usage surprised me, but after some time made sense as I skimmed the two datasets. I found that the characters dataset had a lot more chars than anticipated. The characters dataset list used 38,208B while the Direct Supply dataset list used 11,512B. This alligns with my inital hypothesis of 3x difference.