Comparison of various implementations of the Count Min Sketch Algorithm

Introduction of the Algorithm

Count Min Sketch is a streaming algorithm that can be used to track the frequency of events in real-time data streams. It uses a fixed amount of memory and operates with a small approximation error, making it suitable for big data problems where the data size is too large to fit into memory. The Count-Min Sketch algorithm uses a matrix of counters and a set of hash functions to compute an estimated frequency count for each item in the data stream. The estimated count is obtained by taking the minimum count across all the hash functions for each item. The algorithm has a theoretical guarantee of accuracy and can be used for various data analysis tasks, such as finding the most frequent items in a data stream or approximating the sum of frequencies of multiple items.

Problem Statement for this experiment

The main problem statement of this experiment is to check how the Count Min Sketch performs with various trade-offs between the number of hash functions used and the width of the matrix of counters (or buckets).

Approach Used

I performed an experiment to evaluate the effectiveness of the Count Min Sketch algorithm on both a real-world dataset and an artificially generated dataset. To calculate the actual frequencies of the items in the dataset, I used a simple dictionary in the BasicFrequencyCalculator.py file. The Count Min Sketch algorithm was then run on the dataset to estimate the frequency of the items using the CMSImplementation.py file. Finally, I analyzed the results of the algorithm by calculating the frequency of overestimation and the probability of estimation using the AnalysisOfCMS.py file. This experiment allowed me to gain a deeper understanding of the Count Min Sketch algorithm and its application to large datasets.

Details about the implementation

1. ArtificialDatasetGenerator.py
   In this file, a basic dataset is created using the pandas, string and random libraries. Two options are available for generating a list of random integers or ASCII characters, with the length of the list being determined. Once the list is created, the program shuffles it to ensure that similar items are not clustered together and then stores the list in the form of a data frame in a CSV file named "Dataset.csv".

2. BasicFrequencyCalculator.py
   It is an implementation of the basic frequency calculator which uses the pandas library in python. It creates a class called BasicFrequencyCalculator which takes in the path to the dataset as an argument in the constructor. The class has three methods:
   - read_dataset(): It reads the dataset from the given path and stores it in a data frame
   - calculate_frequency(): It is used to determine the number of times each unique item appears in the data frame. It loops through each item in the desired column of the data frame. For each item, it checks if it is already present in the frequency dictionary. If it is present, its count is incremented by 1. If it is not present, a new key is added to the frequency dictionary with the value 1. After the loop is finished, the frequency dictionary contains the number of occurrences of each unique item in the desired column of the data frame.
   - generate_output(): It converts the dictionary into a data frame and saves it to a CSV file named ActualFrequency.csv.

3. CMSImplementation.py
   This program is an implementation of the Count Min Sketch Algorithm using CSV, pandas, random, and math libraries of python. The CountMinSketch class is defined with two required parameters dataset_path and result_path along with two optional parameters epsilon and delta with default values of 0.01. Epsilon and delta are used to calculate N (number of hash functions used) and K (width of the matrix of counters) for the algorithm using the formulae:

   N = log(1/delta)

   K = 2/epsilon

   In this class, the generate_output is the driver method. It first calls the read_dataset method which reads the CSV file from the given dataset_path and stores it in the ‘data’ data frame. Then calls the generate matrix method for each item in the data frame and creates the matrix. The frequency count of each item is stored in the matrix using the generate_hash method which generates a hash value based on the item and seed. Then, it writes the estimated frequency count of each item to a CSV file at the specified result_path. If the file exists, it adds a new column with the frequency count, otherwise, it creates a new data frame and writes it to the file. This code also uses the get_frequency function which returns the minimum estimated frequency of a particular item.

   For the implementation of CMS, I used a custom hash function. This function uses the concept of linear lines to generate a hashed value. The generate_hash function takes in two arguments: item and seed. The hash value is computed using the formula: (((a * hash(item)) + b) % K) where "a" and "b" are randomly generated integers between 1 and 10. I am using mod K to ensure that the output is always within the range of my data structure. This hash function won’t be considered pair-wise independent as pairwise independent hash functions are functions that map two distinct inputs to two distinct outputs with high probability. This function uses a randomly generated coefficient 'a' and

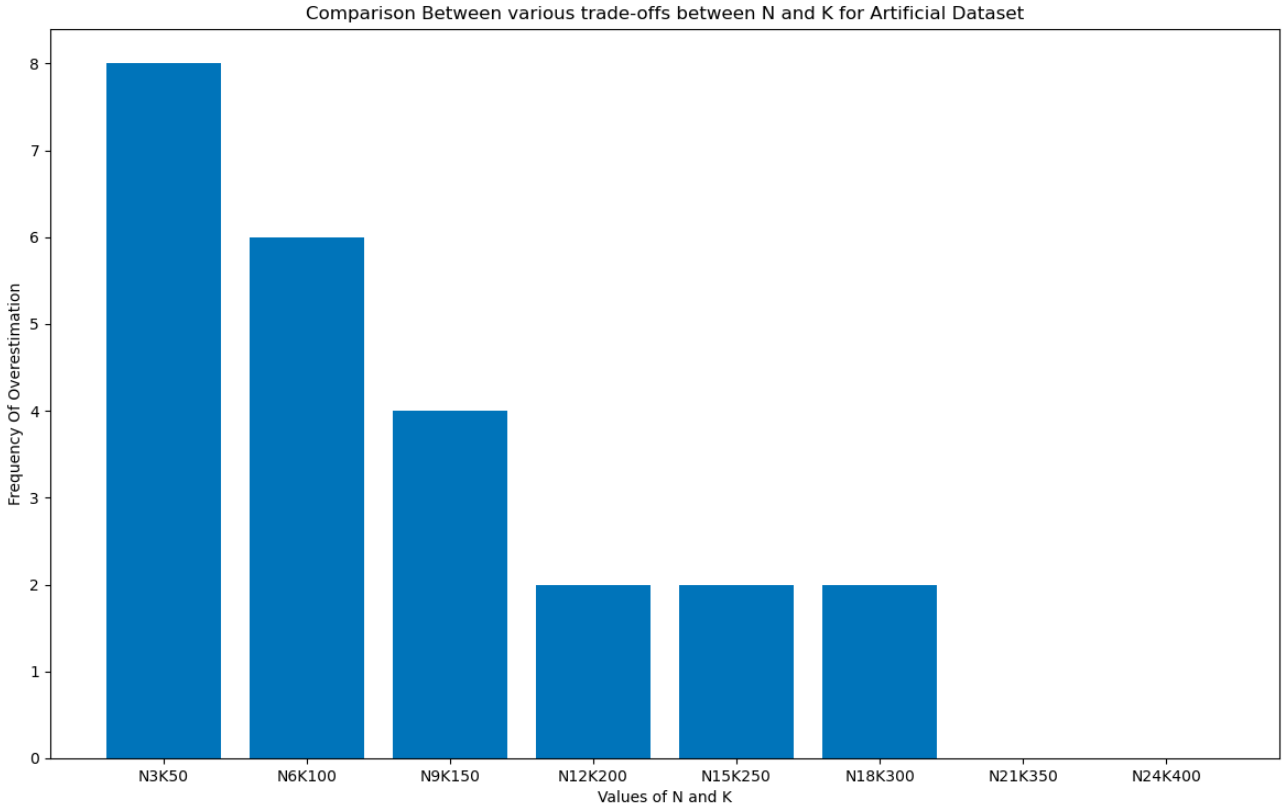
'b' to generate the hash value, which means it could produce the same output for different inputs, making it not pairwise independent.

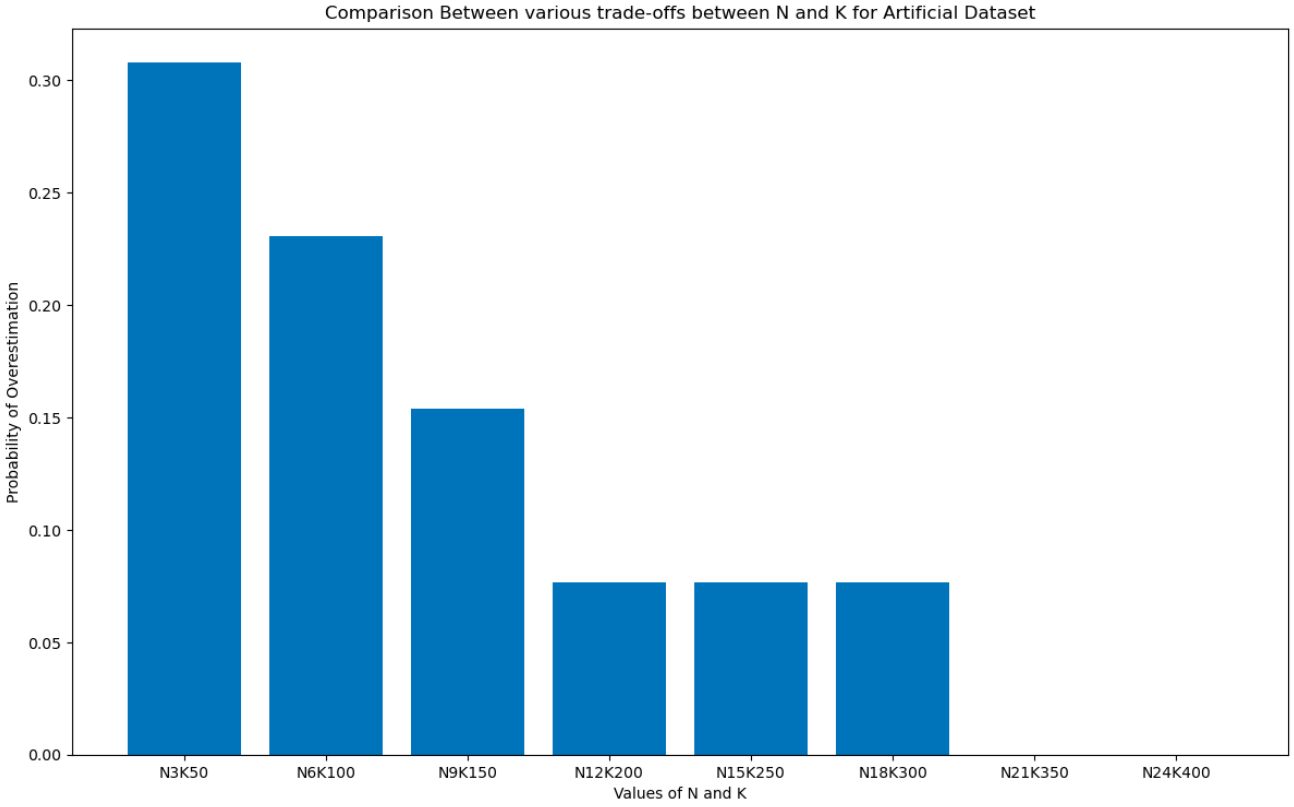
4. AnalysisOfCMS.py
   This code defines a class that performs a comparison between actual results obtained from a basic frequency calculator and estimated results obtained from the CMS Implementation and generates a graph to visualize the results. The class takes two arguments in its constructor - actual_results_path and estimated_results_path, which are the path of the corresponding .csv files.
   The code has two functions:
   - read_files(): This method reads the .csv files and stores the data in actual_results and estimated_results.
   - comparison(): This function performs the comparison between the two results. It first stores and calculates the deviation, frequency, and probability of overestimation into the result_df data frame then saves it as a .csv file. Then, this method creates two bar graph visualizations for both probability and frequency of overestimations and saves them as .png files
   - plot_deviation(): This function plots the top 10 deviations as a bar graph

Results

Artificial Dataset

- Various trade-offs of N and K for the artificial dataset (Answer to Question 4a)

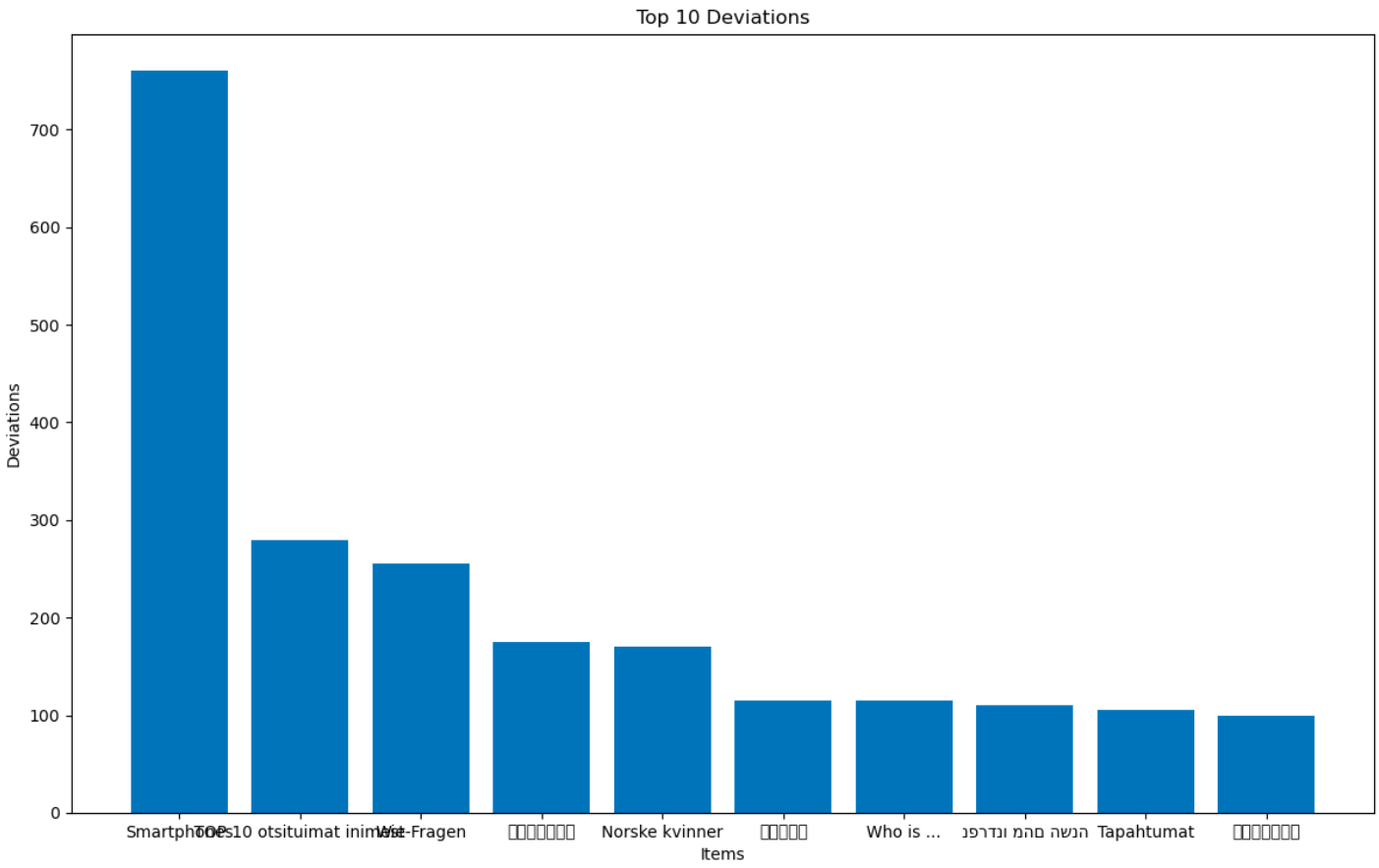

Comparison Between various trade-offs between N and K for Artificial Dataset
Frequency Of Overestimation
0
1
2
3
4
5
6
7
8
N3K50
N6K100
N9K150
N12K200
N15K250
N18K300
N21K350
N24K400
Values of N and K


Comparison Between various trade-offs between N and K for Artificial Dataset
Probability of Overestimation
0.00
0.05
0.10
0.15
0.20
0.25
0.30
N3K50
N6K100
N9K150
N12K200
N15K250
N18K300
N21K350
N24K400
Values of N and K

(Answer to Question 4b)
Here, we can easily see that as N and K increase the frequency and hence the probability of overestimation decreases. This is an expected behavior. The higher the N and K, the lower the probability of overestimation. The mathematical formula that relates the N and the desired maximum error epsilon is N = ceil(ln(1/delta)) where delta is the probability of overestimation.
So, if the desired maximum error (epsilon) is smaller, a higher N and K are required to achieve it. On the other hand, if the desired maximum error is larger, a lower N and K can be used. The trade-off between the N and K depends on the available memory and the desired accuracy of the estimation.
*Here by frequency of overestimation, I mean the total number of times any item was overestimated

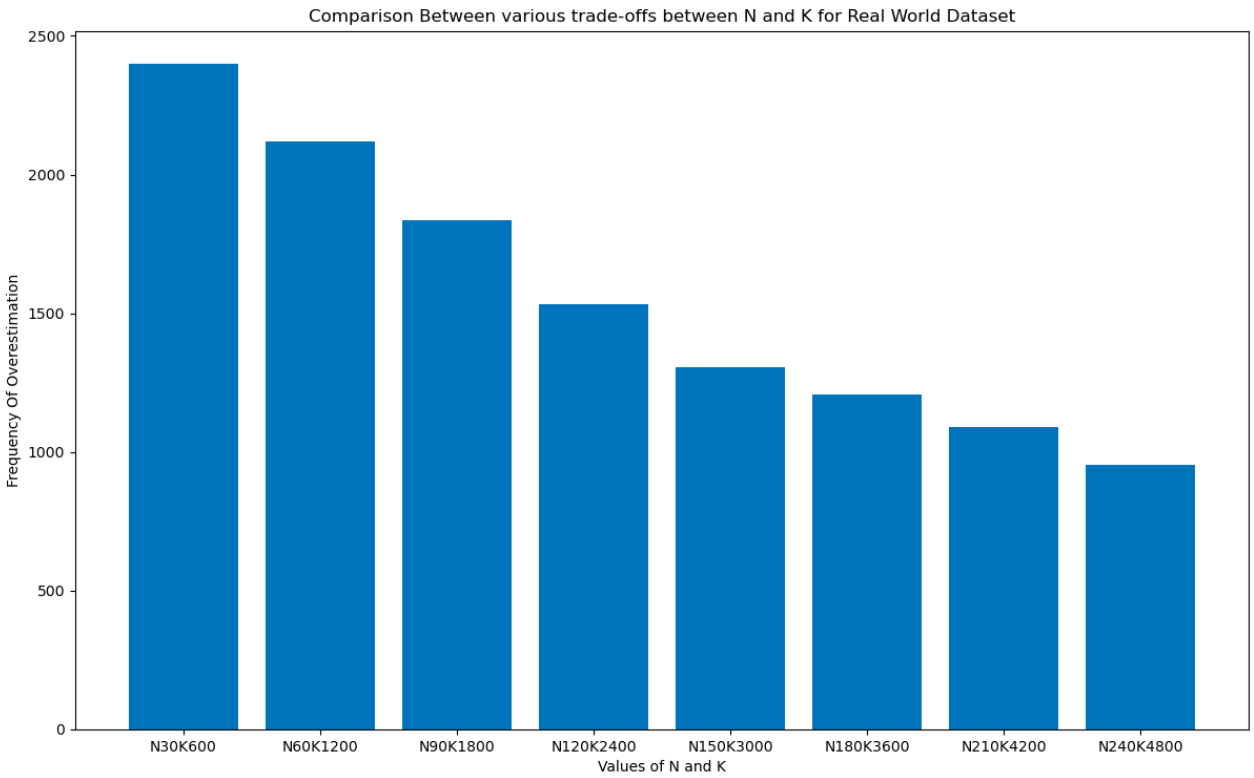
Real World Dataset

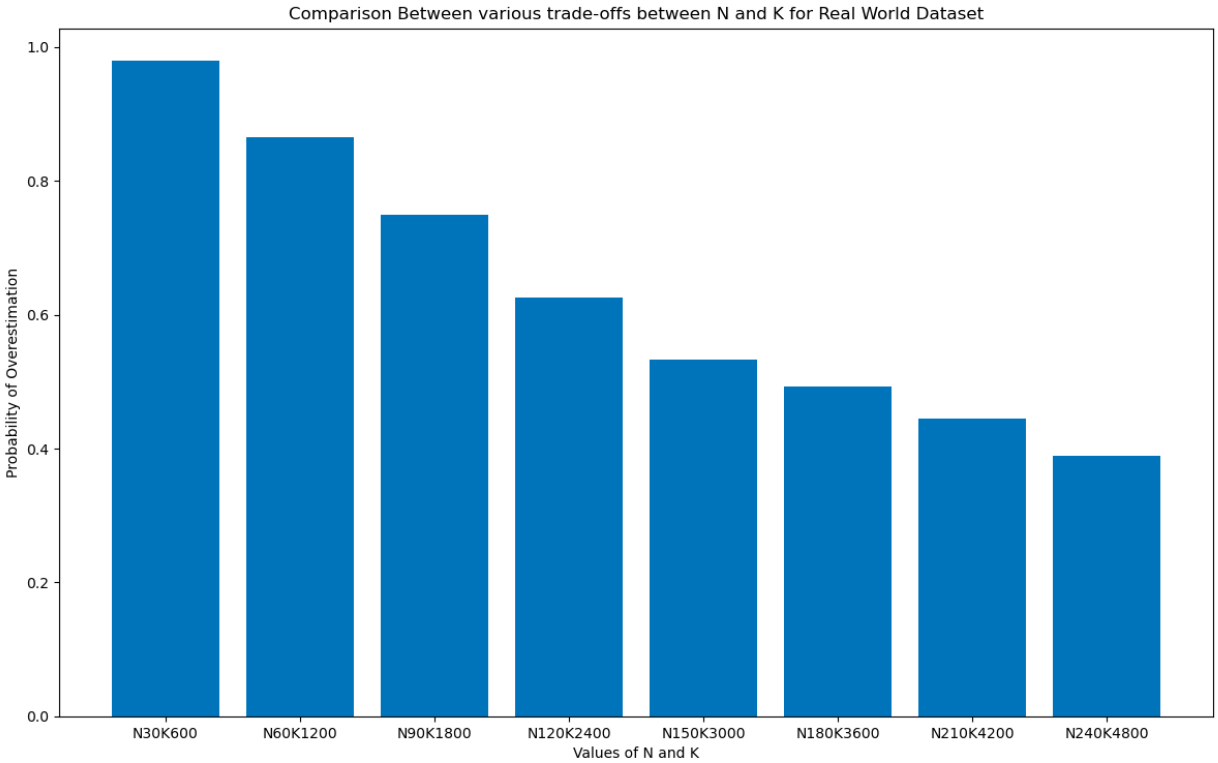
- Deviations for a Real World Dataset (Answer to Question 3)



The probability of overestimation, in this case, was 0.38 with N = 240 and K = 4800

- Various trade-offs of N and K for the Real world dataset (Answer to Question 4a)


Comparison Between various trade-offs between N and K for Real World Dataset
Frequency Of Overestimation
2500
2000
1500
1000
500
0
N30K600
N60K1200
N90K1800
N120K2400
N150K3000
N180K3600
N210K4200
N240K4800
Values of N and K


Comparison Between various trade-offs between N and K for Real World Dataset
Probability of Overestimation
1.0
0.8
0.6
0.4
0.2
0.0
N30K600
N60K1200
N90K1800
N120K2400
N150K3000
N180K3600
N210K4200
N240K4800
Values of N and K

(Answer to Question 4b and Question 5)

We can confidently say that the relationship that was shown for the artificial data set was also confirmed by the real-world dataset. Another thing to notice here is that the probability of overestimation is not always at most 1/2. The probability of overestimation depends on the values of N and K of the matrix used in the algorithm. The smaller the values of N and K, the greater the chance of overestimation.

In practice, a common trade-off is to set the values of depth and width such that the probability of overestimation is at most some desired value, such as 1/1000. This is accomplished by selecting appropriate values for N and K based on the desired accuracy, the size of the data, and the number of distinct items being counted.

Although, we can easily see that the analysis that probability is at most ½ holds true for greater values of K.

*Here by frequency of overestimation, I mean the total number of times any item was overestimated

How to run the code

To run this experiment on a Linux-based terminal:

1. cd to the python folder of your system
2. Paste the path of the python file you want to run and press enter

Things to keep in mind:

1. Give appropriate arguments to the class (like the path of the datasets etc)
2. Change the name of the column, whose item's frequency you want to estimate in the CMSImplementation.py and BasicFrequencyCalculator.py files.

Dataset

There is two types of datasets used in this program.

- Artificial Dataset: This is created using the ArtificialDatasetGenerator.py file. We can create two types of datasets using this: a dataset of ASCII characters and a dataset of integers.
- Real World Dataset: This dataset was downloaded from Kaggle, using this link. This is a curated dataset of Google Trends over the years. Every year, Google releases trending search queries all over the world in various categories. It has trends from 2001 to 2020.

Answer to Question 6

It took me around 24 hours to run this experiment. Although most of the time went into researching and understanding the relationship between N and K. I would say this was a medium-level assignment