TOWARDS INTELLIGENT DATA PROFILING AND AGGREGATION

Nidhi Menon
Sneha Venkatachalam

AGENDA

- INTRODUCTION
- RELATED WORK
- GOALS
- MOTIVATION
- IMPLEMENTATION
- PERFORMANCE MEASURE
- EVALUATION
- QUESTIONS & DISCUSSIONS

INTRODUCTION

- Data Aggregation
  - Useful in Data Science and Statistics
  - Eliminate repetitive Calculation of Statistics
- Data Profiling
  - Useful in Data pre-processing and analytics
  - Summarize data
- Key Idea: Intelligent data profiling and aggregation for faster query retrieval

RELATED WORK

Data Canopy: Accelerating Exploratory Statistical Analysis

A. Wasay, X. Wei, N. Dayan, and S. Idreos, "Data Canopy: Accelerating Exploratory Statistical Analysis," in ACM SIGMOD International Conference on Management of Data, 2017

Profiling relational data: a survey

Abedjan, Ziawasch, Lukasz Golab, and Felix Naumann. "Profiling Relational Data: A Survey." The VLDB Journal 24.4 (2015): 557–581

GOALS- ORIGINAL

- 75 - 80% : Implementing a data profiling system by replicating existing papers
- 100% : Computing and adding new metrics to the data profiling tool and building a visualization dashboard
- 125% : Leveraging the tool for ML models

GOALS- NEW

- 75 - 80% : Implementing a data profiling system by incorporating data aggregation for key metrics by pre-computing statistics by replicating the Data Canopy paper

- 100% : Computing and adding support for more complex data profiling tasks

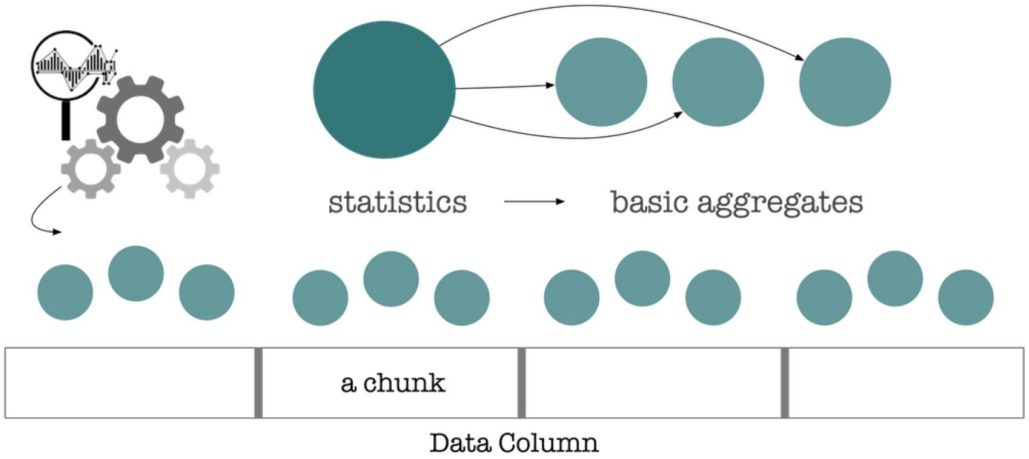
- 125% : Incorporating persistence for project by implementing LRU cache to aid efficient memory management and faster data retrieval

WHY DID THE GOALS CHANGE?

- With time, goals evolved based on feedback
- Tackling complex technical tasks using data aggregation is more challenging than working on a dashboard for visualization and leveraging the tool for machine learning tasks

STATISTICAL CALCULATIONS

- Data column divided into chunks
- Basic aggregates calculated for each chunk
- Statistics for the column calculated using pre-computed basic aggregates



REPETITIVE STATISTICS

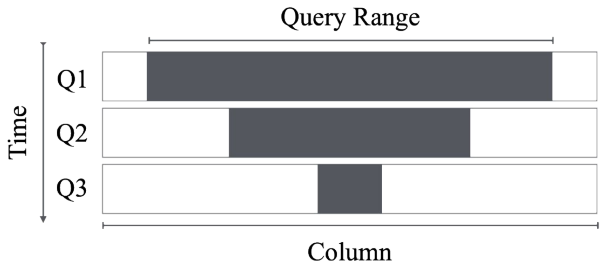


Fig.: Sub-range

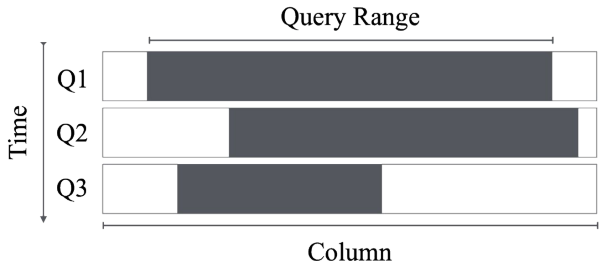


Fig.: Overlap

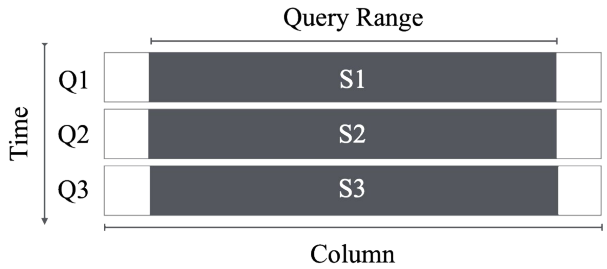


Fig.: Different Statistics

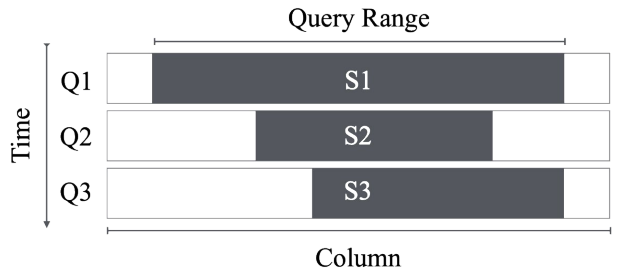
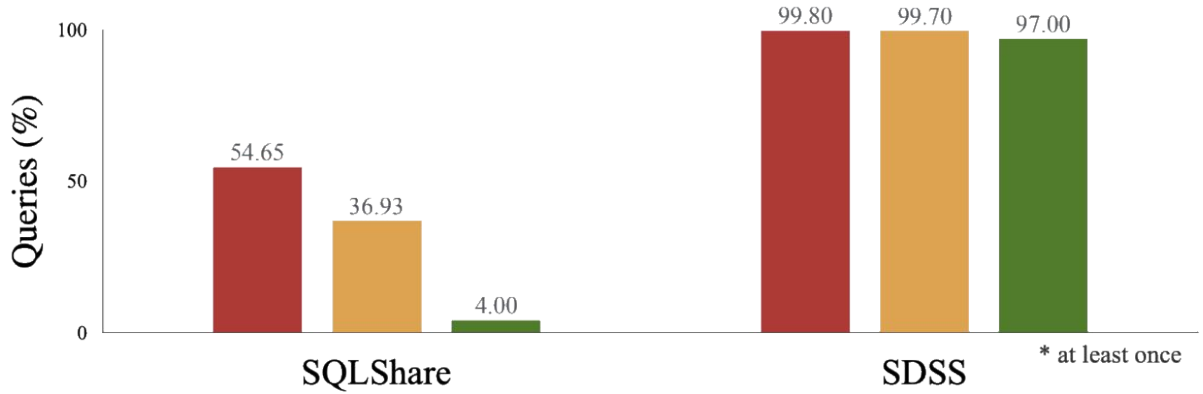


Fig.: Mixed

MOTIVATION

Exploratory Workloads Exhibit Repetition

- Repetition is everywhere - between 50% to 99%



SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment
Shrainik Jain, Dominik Moritz, Bill Howe, Ed Lazowska. SIGMOD 2016

IMPLEMENTATION

- **Language**: C++
- **Dataset (Numerical)**: Randomly generated; uniform distribution; ~10k rows
- **Query structure**: queryMethod (low, high, column_name)

SEGMENT TREE

A tree data structure that stores data about intervals, or segments

- Binary tree
- Leaves: data instances
- Internal nodes: union of elementary intervals

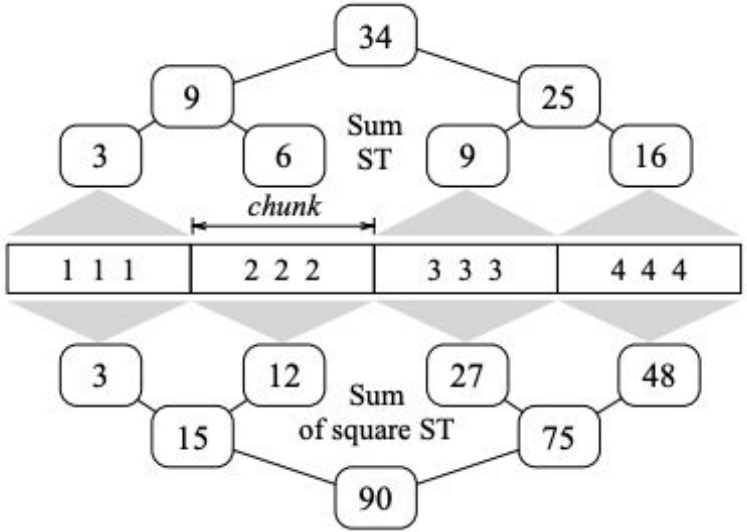


Image from the paper titled 'Data Canopy' by Wasay et. al.

APPROACH

- Data Aggregation
  - Segment tree implementation for caching
    - Build segment tree for entire dataset
    - Handles querying over continuous ranges
    - Handles updates to the data
  - Hash table implementation for mapping
    - Maps incoming query to corresponding segment tree
  - Statistics computation
    - Statistics mentioned in the ‘Data Canopy’ paper and ‘Data Profiling’ paper

Statistics		Basic Aggregates				
Type	***Formula***	$\sum x$	$\sum x^2$	$\sum xy$	$\sum y^2$	$\sum y$
Mean (avg)	$\frac{\sum x_i}{n}$	■				
Root Mean Square (rms)	$\sqrt{\frac{1}{n} \cdot \sum x^2}$		■			
Variance (var)	$\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}$	■	■			
Standard Deviation (std)	$\sqrt{\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}}$	■	■			
Sample Covariance (cov)	$\frac{\sum x_i \cdot y_i}{n} - \frac{\sum x_i \cdot \sum y_i}{n^2}$	■		■		■
Simple Linear Regression (slr)	$\frac{\text{cov}(x,y)}{\text{var}(x)}, \text{avg}(x), \text{avg}(y)$	■	■	■		■
Sample Correlation (corr)	$\frac{n \cdot \sum x_i \cdot y_i - \sum x_i \cdot \sum y_i}{\sqrt{n \cdot \sum x_i^2 - (\sum x_i)^2} \sqrt{n \cdot \sum y_i^2 - (\sum y_i)^2}}$	■	■	■	■	■

Table of statistics from the paper titled ‘Data Canopy’ by Wasay et. al.

APPROACH

- Segment tree implementation for caching
    - Build segment tree for entire dataset
    - Handles querying over continuous ranges
    - Handles updates to the data
  - Hash table implementation for mapping
    - Maps incoming query to corresponding segment tree
  - Statistics computation
    - Statistics mentioned in the ‘Data Canopy’ paper and ‘Data Profiling’ paper

APPROACH

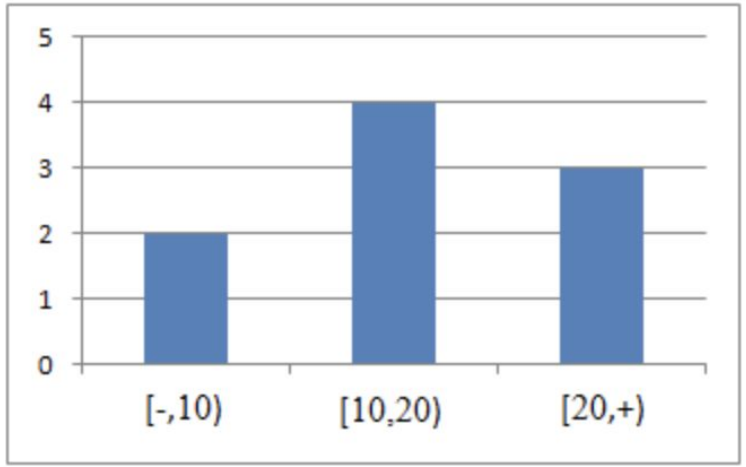
- Data Profiling
  - Task of reviewing data to understand its structure, content and relationships
  - Aids us in computing statistics or in collective informative summaries about the data
  - Data Profiling tasks include:
    - **Single-column tasks**
    - **Multi-column tasks**
    - Dependency detection
  - A set of results of these tasks gives a ***data profile*** or ***database profile***

Category	Task	Description
Cardinalities	num-rows	Number of rows
	value length	Measurements of value lengths (minimum, maximum, median, and average)
	null values	Number or percentage of null values
	distinct	Number of distinct values; sometimes called "cardinality"
	uniqueness	Number of distinct values divided by the number of rows
Value distributions	histogram	Frequency histograms (equi-width, equi-depth, etc.)
	constancy	Frequency of most frequent value divided by number of rows
	quartiles	Three points that divide the (numeric) values into four equal groups
	first digit	Distribution of first digit in numeric values; to check Benford's law
Patterns, data types, and domains	basic type	Generic data type, such as numeric, alphabetic, alphanumeric, date, time
	data type	Concrete DBMS-specific data type, such as varchar, timestamp.
	size	Maximum number of digits in numeric values
	decimals	Maximum number of decimals in numeric values
	patterns	Histogram of value patterns (Aa9…)
	data class	Semantic, generic data type, such as code, indicator, text, date/time, quantity, identifier
	domain	Classification of semantic domain, such as credit card, first name, city, phenotype

[Profiling relational data: a survey]: Overview of selected single column profiling tasks

APPROACH

- **Equal-width histogram**



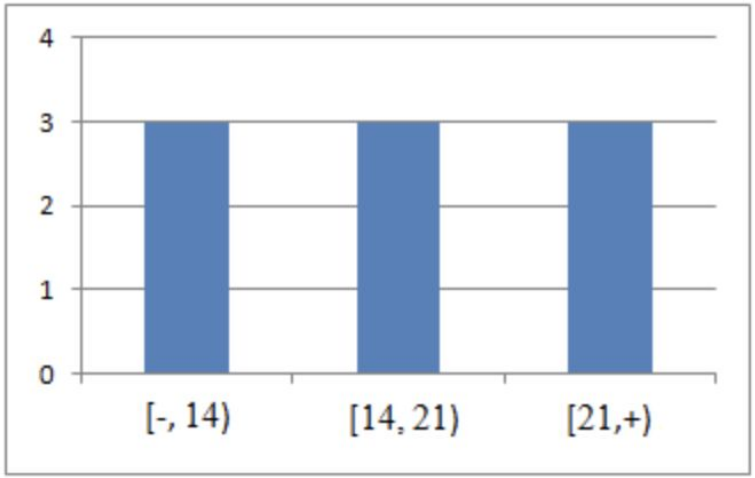
Example of an equal-width histogram

APPROACH

- Single column profiling
  - Category: Value Distribution
    - **Equal-width histogram I:** Aggregates for base width '*w*' and multiples of '*w*'
      - Supported for data types ***int*** and ***float***
      - Based on the concept of binning over base width 'w'
    - **Equal-width histogram II:** Aggregates over any bin-size i.e. width '*w*'
      - Supported for data type ***int*** only
      - Based on the concept of inverted index

APPROACH

- **Equal-height histogram**



Example of an equal-height histogram

APPROACH

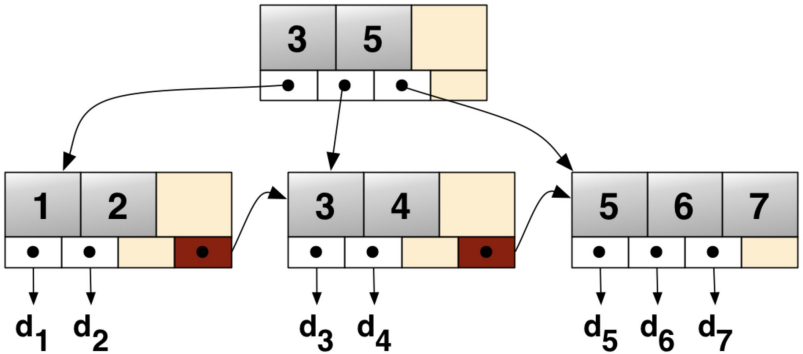
- Single column profiling
  - Category: Value Distribution
    - **Equal-height histogram:**
      - Involves sorting the entire column and creating bins representing dynamic ranges
      - Data aggregation is difficult due to dynamic-sized ranges

APPROACH

- Persistence
  - Method to efficiently store data structures such that they can continue to be accessed using memory instructions or memory APIs even after the end of the process that created or last modified them
  - Implemented using LRU (Least Recently Used) cache
  - Dataset divided into chunks, and only most recently used chunks are retained in memory to create segment trees for efficient memory management

APPROACH

- B+ Tree
  - An N-ary tree with a variable but often large number of children per node
  - Aids in storing data for efficient retrieval in a block-oriented storage context



Sample B+ tree

PERFORMANCE MEASURE

- Time Complexity
  - Tree Construction: *O(n)*
    - *2(n)* nodes, value of each node calculated once in tree construction
  - Tree Query: O(logn)
    - Number of levels: *O(logn)*
    - To query a range minimum, at most 2 nodes at every level processed
- Space Complexity
  - Tree: *O(n)*
    - For *n* data instances, segment tree uses a *2(n)* sized array

PERFORMANCE MEASURE

- MEMORY: Comparison of traditional implementation vs. our implementation
  - **Traditional**
    - No memory overhead assuming dynamic calculations for all statistics
  - **Our Implementation**
    - For int/float values (10k data instances)

      10,000 * 2 * 4 = 80,000 bytes = 80 KB

PERFORMANCE MEASURE

- Time taken for Segment Tree Construction (Array size = 10k)

Base aggregates	Construction time (micro-seconds)
Σx	170
Σsquare(x)	192
Σxy	82

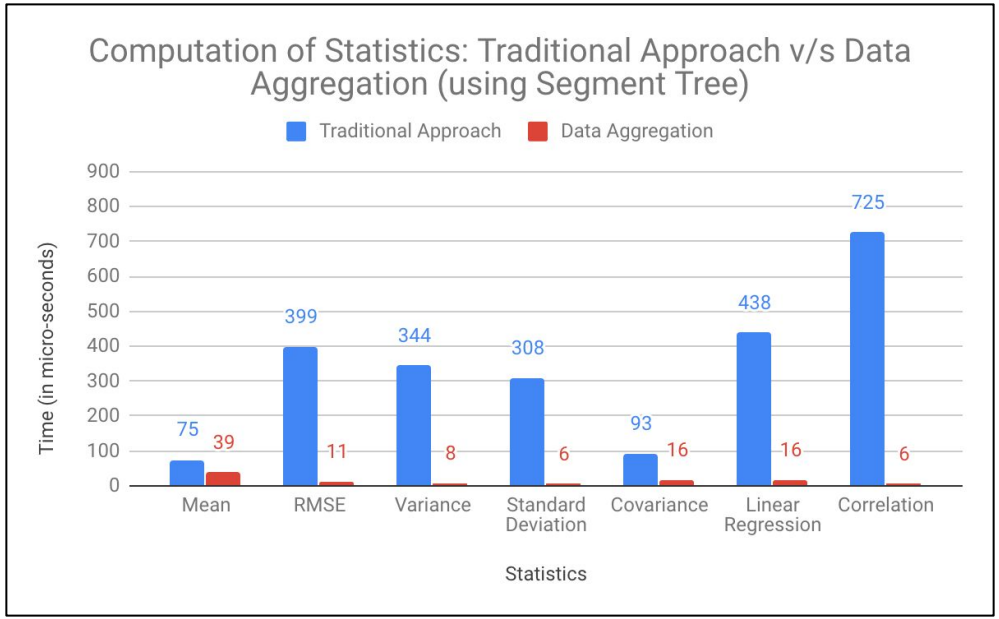
EVALUATION

- Time taken for Querying the Segment Tree (Array size = 10k)

Traditional Approach	Query Time (in micro-seconds)	Data Aggregation (Segment Tree)	Query Time (in micro-seconds)
Equal width	200281	Equal width	111
Equal height	681	Equal height	69

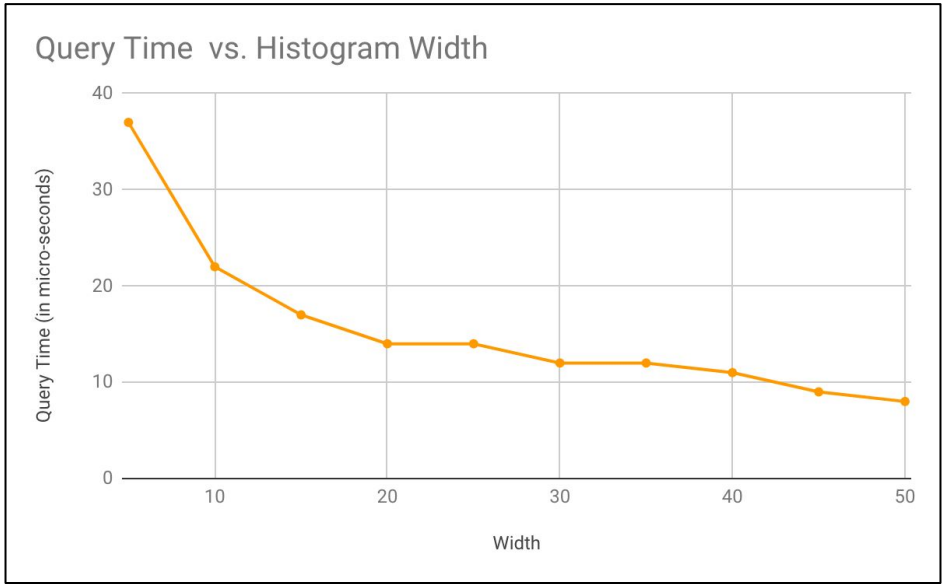
EVALUATION

- SPEEDUP: Comparison of our implementation vs. traditional implementation



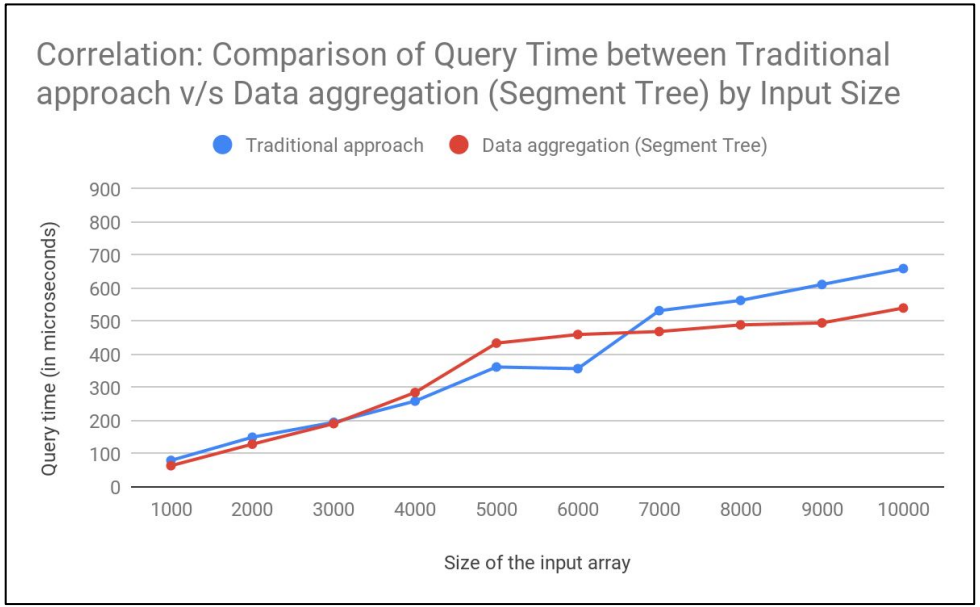
EVALUATION

- Decrease in query time with increase in histogram query width size
  - Using the Segment Tree Histogram implementation with base width = 5



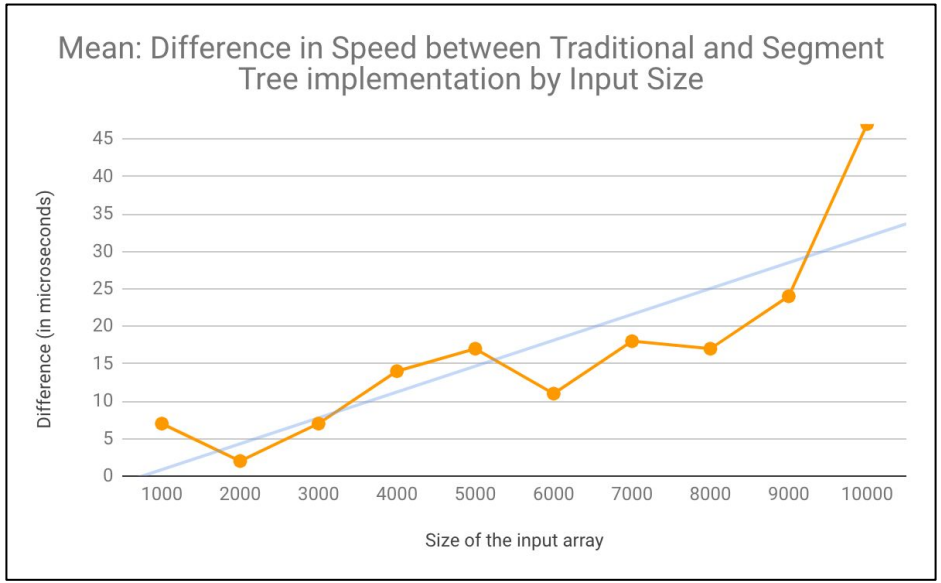
EVALUATION

- Comparing Query Time for the ‘Correlation’ operation for varying input size



EVALUATION

- Observing differences in speed for varying input size for the ‘Mean’ operation



CORRECTNESS

- Comparison of our implementation vs. traditional implementation

```
Mean: 5002.88
Time taken by function: 39 microseconds

RMSE: 5774.25
Time taken by function: 11 microseconds

Variance: 3.3342e+07
Time taken by function: 8 microseconds

STD: 5774.25
Time taken by function: 6 microseconds

Covariance: 86945.9
Time taken by function: 16 microseconds

Simple Linear Regression: 0.0026675
Time taken by function: 16 microseconds

Correlation: 0.0104589
Time taken by function: 6 microseconds
```

```
Mean: 5002.88
Time taken by function: 75 microseconds

RMSE: 5774.25
Time taken by function: 399 microseconds

Variance: 3.33419e+07
Time taken by function: 344 microseconds

STD: 5774.25
Time taken by function: 308 microseconds

Covariance: 86916.4
Time taken by function: 93 microseconds

Simple Linear Regression: 0.00266659
Time taken by function: 438 microseconds

Correlation: 0.0104553
Time taken by function: 725 microseconds
```

FUTURE SCOPE

- Inverted or adaptive indexing
- Dealing with approximate queries
- Implement for other statistics that use aggregation
- Persistent segment tree
- Multithreading

QUESTIONS & DISCUSSIONS