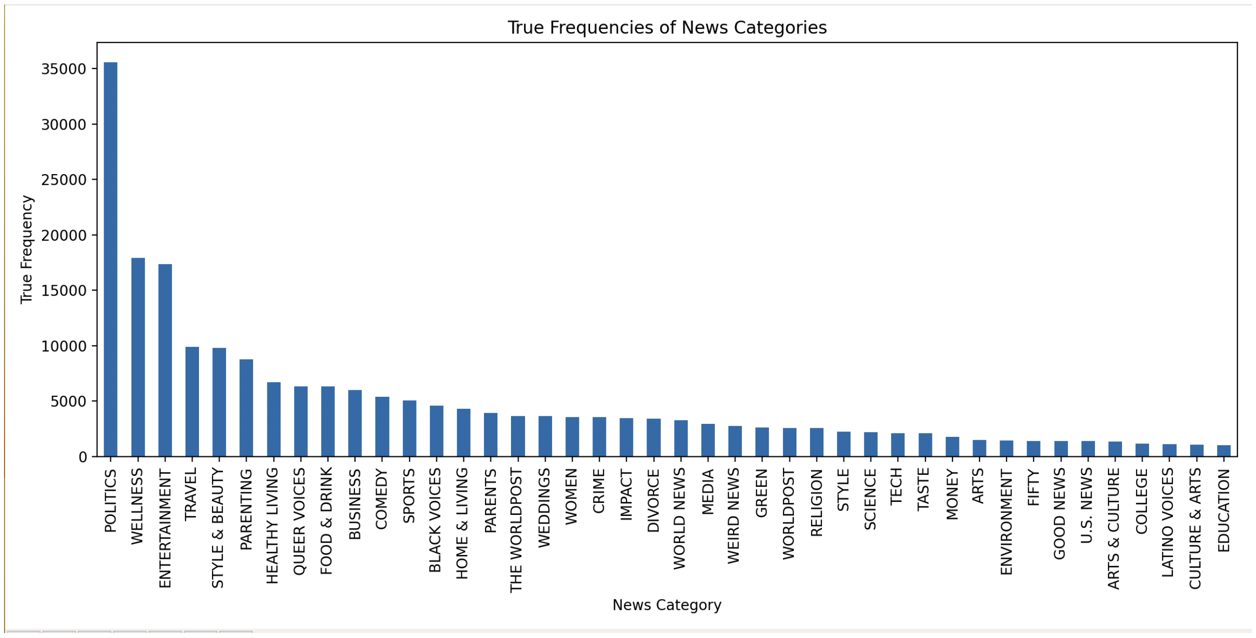
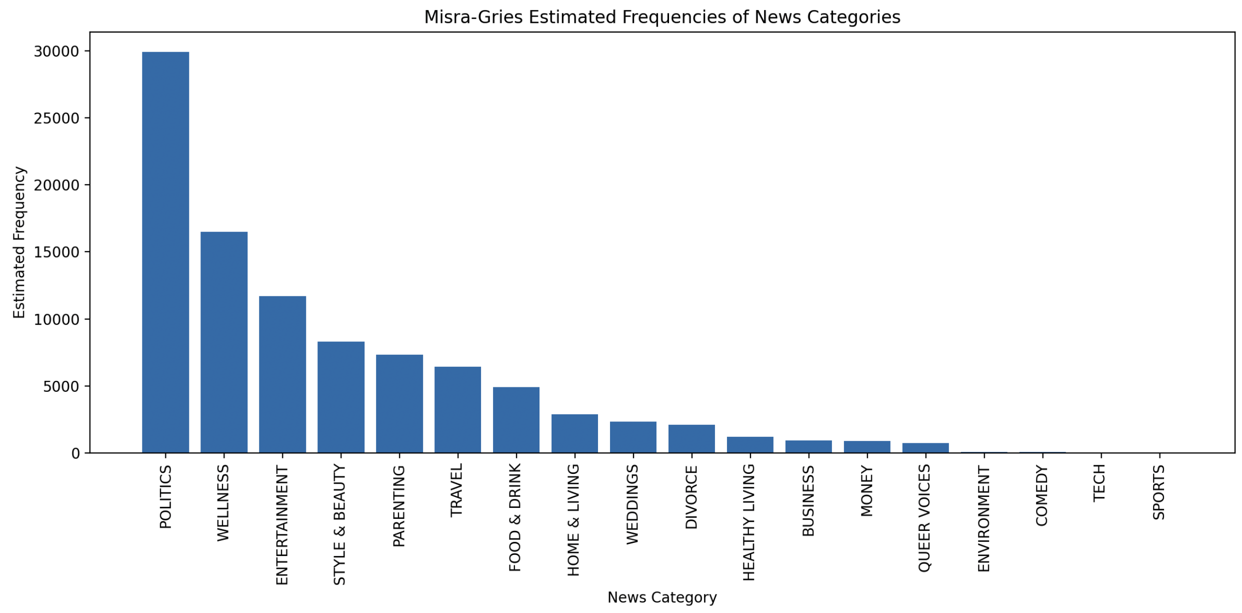
Assignment 2 Report

1-A: average frequency: 4988.738095238095

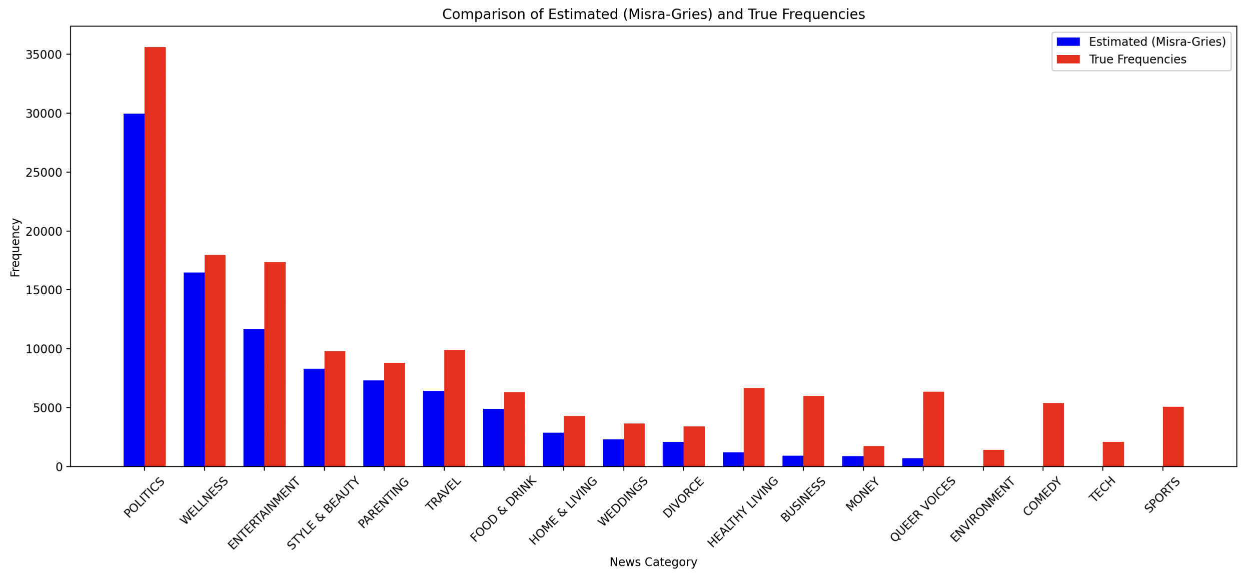
1-B:



2-A:

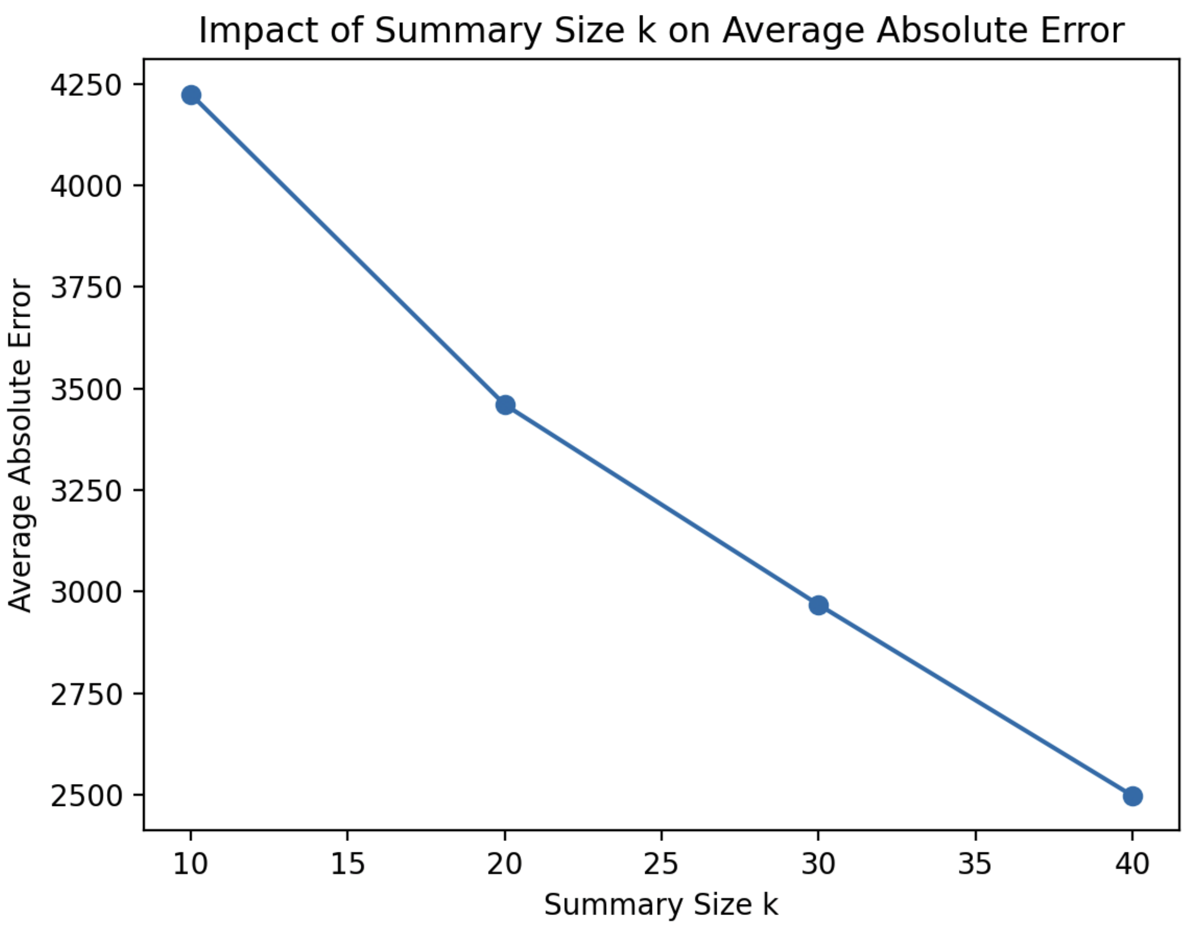


2-B



2-C: Number of decrement steps with k=20: 5664

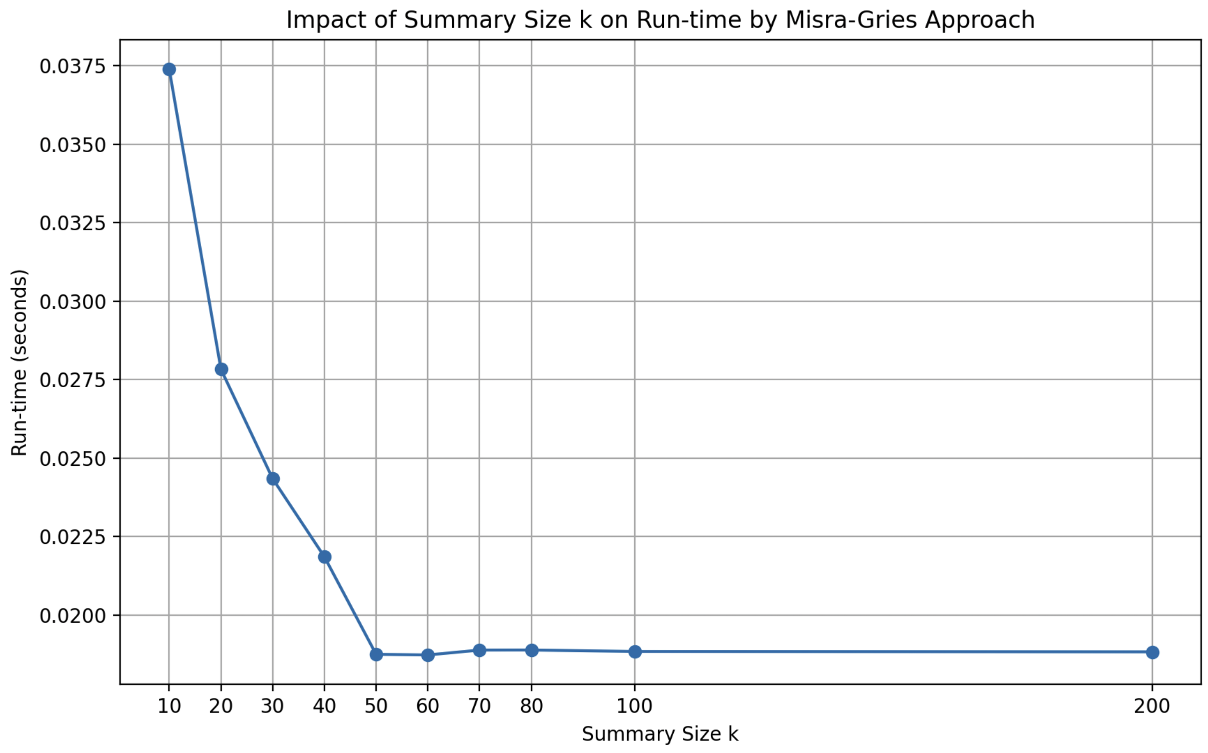
2-D:



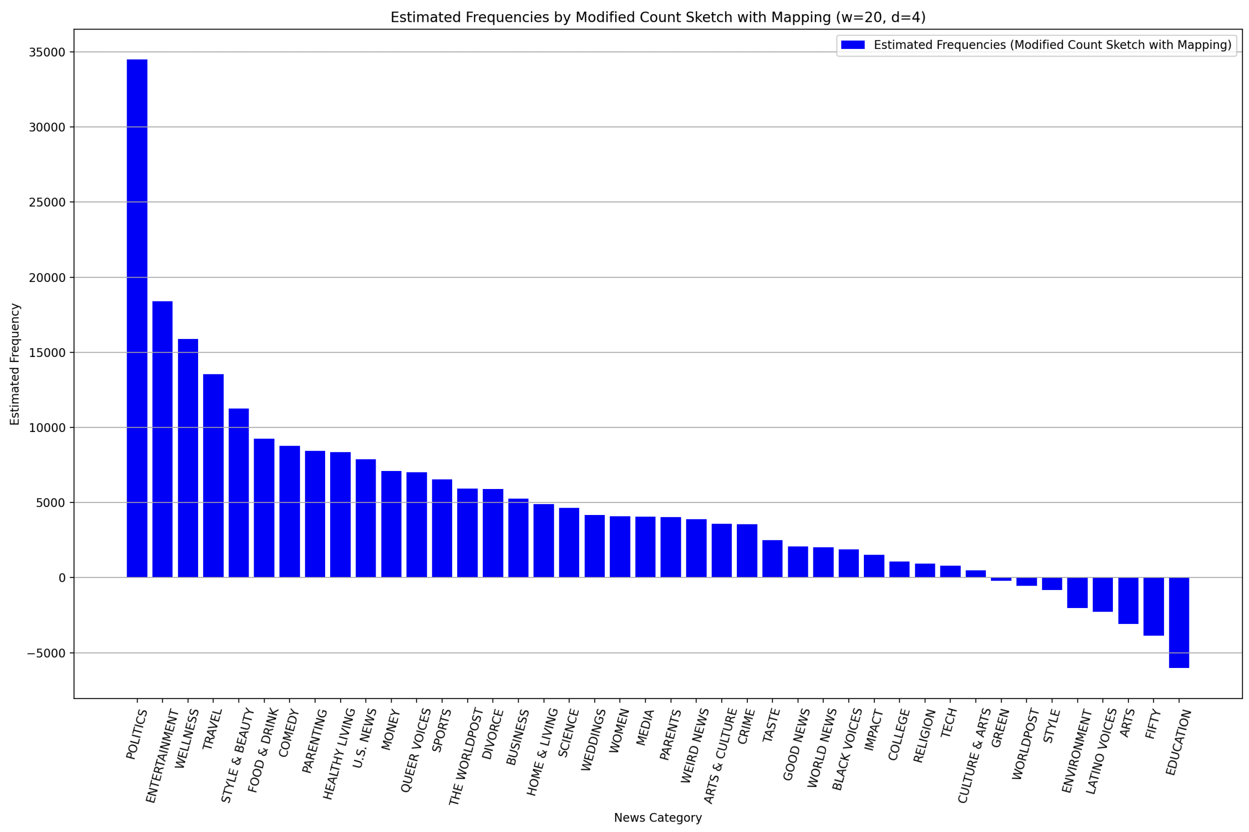
The graph above displays the impact of varying the summary size *k* on the average absolute error using the Misra-Gries algorithm. As observed, as *k* increases, the average absolute error decreases, suggesting that having more counters in the Misra-Gries summary leads to more accurate frequency estimations.

2-E

```
Number of decrement steps with k=10: 17739
Number of decrement steps with k=20: 5664
Number of decrement steps with k=30: 2777
Number of decrement steps with k=40: 1144
Number of decrement steps with k=50: 0
Number of decrement steps with k=60: 0
Number of decrement steps with k=70: 0
Number of decrement steps with k=80: 0
Number of decrement steps with k=100: 0
Number of decrement steps with k=200: 0
```

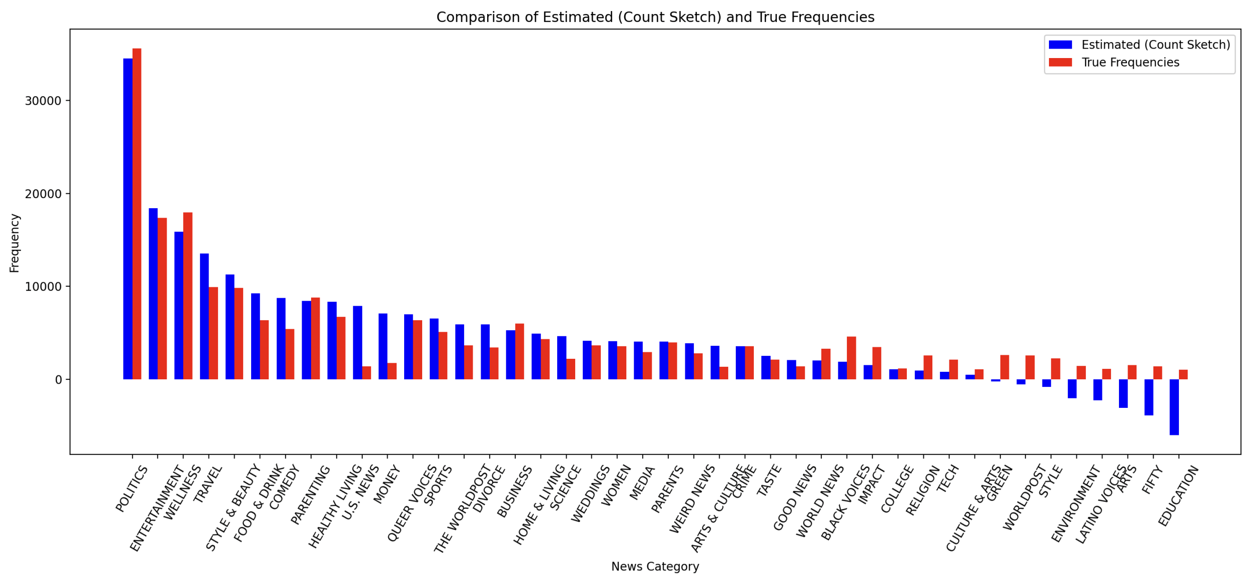


3-A:



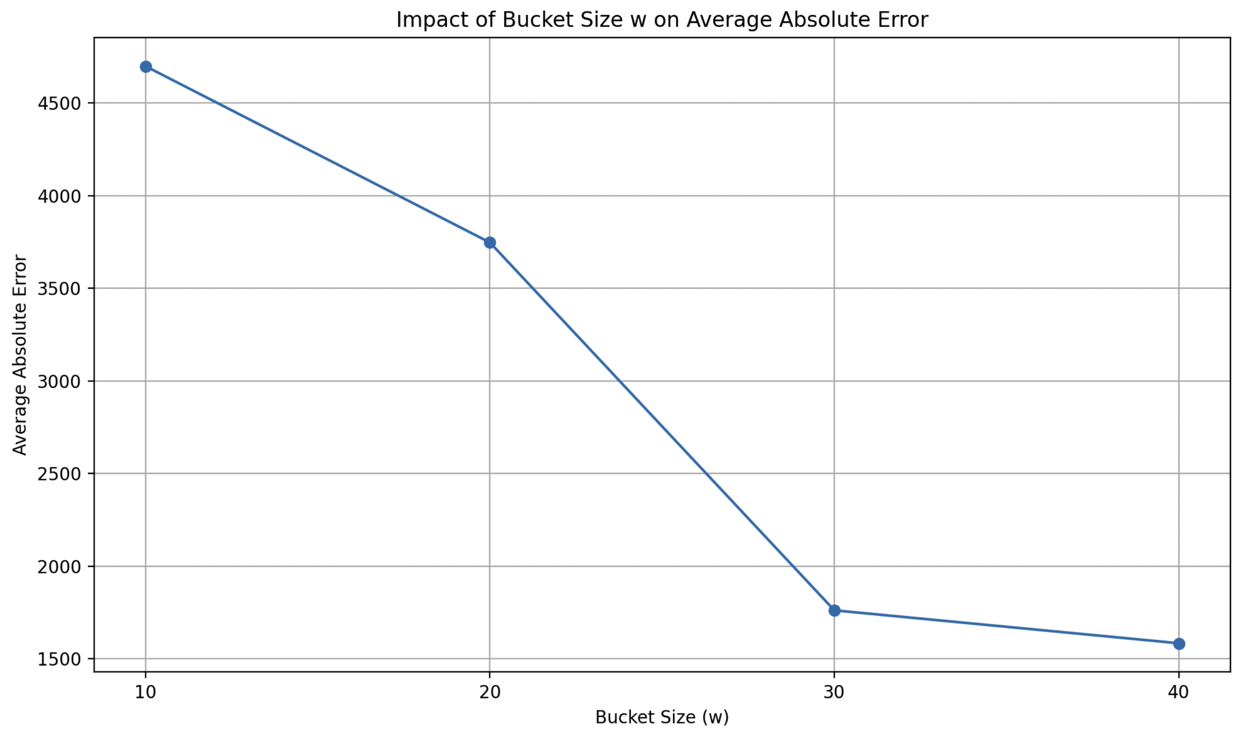
Estimated Frequencies by Modified Count Sketch with Mapping (w=20, d=4)
Estimated Frequencies (Modified Count Sketch with Mapping)
Estimated Frequency
News Category
POLITICS
ENTERTAINMENT
WELLNESS
TRAVEL
STYLE & BEAUTY
FOOD & DRINK
COMEDY
PARENTING
HEALTHY LIVING
U.S. NEWS
MONEY
QUEER VOICES
SPORTS
THE WORLDPOST
DIVORCE
BUSINESS
HOME & LIVING
SCIENCE
WEDDINGS
WOMEN
MEDIA
PARENTS
WEIRD NEWS
ARTS & CULTURE
CRIME
TASTE
GOOD NEWS
WORLD NEWS
BLACK VOICES
IMPACT
COLLEGE
RELIGION
TECH
CULTURE & ARTS
GREEN
WORLDPOST
STYLE
ENVIRONMENT
LATINO VOICES
ARTS
FIFTY
EDUCATION

3-B:



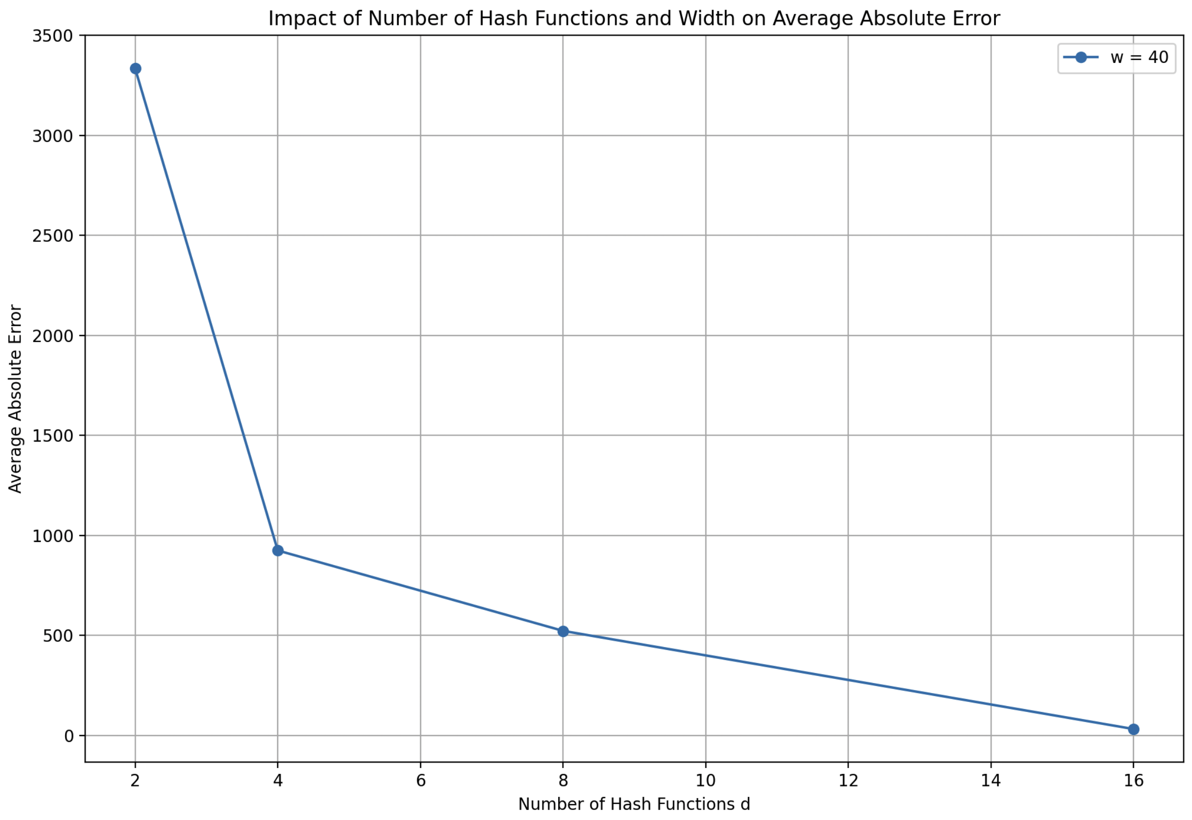
Comparison of Estimated (Count Sketch) and True Frequencies
Estimated (Count Sketch)
True Frequencies
Frequency
News Category

3-C:



The above curve illustrates how varying the bucket size w affects the average absolute error. As observed, as *w* increases, the average absolute error decreases. This means having a larger table (more buckets) in the Count Sketch algorithm provides more accurate frequency estimations.

3-D:



The graph above demonstrates the influence of varying the number of hash functions *d* on the average absolute error. As observed, increasing *d* leads to a decrease in the average absolute error, signifying that using more hash functions in the Count Sketch algorithm enhances the accuracy of frequency estimations.

Source code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time

# Column names
column_names = ['news_id', 'news_category', 'date']

# Load the dataset using comma as the delimiter and specify column names
df = pd.read_csv('news_stream.csv', delimiter=',',
                 header=None, names=column_names)

# Group by the 'news_category' column and count the occurrences
category_counts = df['news_category'].value_counts()

# 1.A Compute the average frequency
category_frequency = category_counts
```

```
average_frequency = category_frequency.mean()
print("average frequency: ", average_frequency)

# 1.B Compute the true frequencies of all categories
# Sort the frequencies in descending order
sorted_frequency = category_frequency.sort_values(ascending=False)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sorted_frequency.plot(kind='bar')
plt.ylabel('True Frequency')
plt.xlabel('News Category')
plt.title('True Frequencies of News Categories')
plt.tight_layout()
plt.show()

# 2 Misra-Gries Approach and Performance Evaluation


def misra_gries(stream, k):
    counters = {}
    decrement_steps = 0  # Initialize a counter for decrement steps

    for item in stream:
        if item in counters:
            counters[item] += 1
        elif len(counters) < k - 1:
            counters[item] = 1
        else:
            decrement_steps += 1  # Increment the counter when decrementing all items
            for key in list(counters.keys()):
                counters[key] -= 1
                if counters[key] == 0:
                    del counters[key]
    return counters, decrement_steps


# 2.A Apply Misra-Gries algorithm
k = 20
estimated_frequencies, num_decrements = misra_gries(df['news_category'], k)

# Sort the estimated frequencies in descending order
sorted_frequencies = dict(
    sorted(estimated_frequencies.items(), key=lambda item: item[1], reverse=True))

# Plot the estimated frequencies
plt.figure(figsize=(12, 6))
plt.bar(sorted_frequencies.keys(), sorted_frequencies.values())
plt.ylabel('Estimated Frequency')
plt.xlabel('News Category')
plt.title('Misra-Gries Estimated Frequencies of News Categories')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 2.B Compare the estimated and true frequencies
# Plot the estimated and true frequencies
plt.figure(figsize=(15, 7))
categories = list(sorted_frequencies.keys())
estimated_values = [sorted_frequencies[cat] for cat in categories]
```

```
# Multiply by len(df) to get counts instead of proportions
true_values = [category_frequency.get(cat, 0) for cat in categories]

bar_width = 0.35
index = range(len(categories))

bar1 = plt.bar(index, estimated_values, bar_width,
               label='Estimated (Misra-Gries)', color='b', align='center')
bar2 = plt.bar([i + bar_width for i in index], true_values,
               bar_width, label='True Frequencies', color='r', align='center')

plt.xlabel('News Category')
plt.ylabel('Frequency')
plt.title('Comparison of Estimated (Misra-Gries) and True Frequencies')
plt.xticks([i + bar_width / 2 for i in index], categories, rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

# 2.C Report the actual number of decrement steps
# Print the number of rows in the dataset
print(f"Number of rows in the dataset: {len(df)}")
print(f"Number of decrement steps with k={k}: {num_decrements}")

# 2.D
# Assuming you have a MisraGries class or function implemented.
# pseudo-code for the entire process


def compute_average_absolute_error(k_values, true_frequencies, stream):
    average_errors = []
    errors = []

    for k in k_values:
        # Run Misra-Gries algorithm with the given k
        estimated_counts, num_decrements = misra_gries(stream, k)

        # compute the absolute error for each category and accumulate
        total_error = 0
        for category, true_frequency in true_frequencies.items():
            # Multiply by len(stream) to get counts instead of proportions
            estimated_frequency = estimated_counts.get(category, 0)
            error = abs(estimated_frequency - true_frequency)
            errors.append(error)

        average_errors.append(sum(errors)/len(errors))

    return average_errors


k_values = [10, 20, 30, 40]
# Assuming true_frequencies is a dictionary with categories as keys and true frequencies as values.
# And stream is a list of items representing the data stream.
average_errors = compute_average_absolute_error(
    k_values, category_frequency, df['news_category'])

plt.plot(k_values, average_errors, marker='o')
plt.xlabel("Summary Size k")
plt.ylabel("Average Absolute Error")
plt.title("Impact of Summary Size k on Average Absolute Error")
plt.show()

# 2.E Investigate the impact of the size of summary k on the run-time by Misra-Gries Approach
k_values = [10, 20, 30, 40, 50, 60, 70, 80, 100, 200]
run_times = []

for k in k_values:
    start_time = time.time()
    estimated_counts, num_decrements_new = misra_gries(df['news_category'], k)
    end_time = time.time()
    run_times.append(end_time - start_time)
    print(f"Number of decrement steps with k={k}: {num_decrements_new}")

# Plotting the runtime against k values
plt.figure(figsize=(10, 6))
plt.plot(k_values, run_times, marker='o')
plt.xlabel('Summary Size k')
plt.ylabel('Run-time (seconds)')
plt.title('Impact of Summary Size k on Run-time by Misra-Gries Approach')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# 3. Count Sketch Approach and Performance Evaluation
# 3.A Implement Count Sketch Algorithm to find the most frequent categories. Please report

category_to_index = {category: idx for idx,
                     category in enumerate(category_counts.index)}


class CountSketchWithMapping:
    def __init__(self, w, d):
        self.w = w
        self.d = d
        self.table = np.zeros((d, w))
        self.prime = 2**31 - 1  # Using a large prime number
        self.coefficients = [(np.random.randint(
            1, self.prime), np.random.randint(0, self.prime)) for _ in range(d)]

    def _hash(self, x, function_idx):
        a, b = self.coefficients[function_idx]
        # Map the category to its unique integer index
        mapped_x = category_to_index[x]
        return ((a * mapped_x + b) % self.prime) % self.w

    def _sign_hash(self, x):
        return 1 if hash(x) % 2 == 0 else -1

    def add(self, x):
        for i in range(self.d):
            j = self._hash(x, i)
            self.table[i][j] += self._sign_hash((x, i))

    def estimate(self, x):
        estimates = []
        for i in range(self.d):
            j = self._hash(x, i)
            estimates.append(self.table[i][j] * self._sign_hash((x, i)))
        return np.median(estimates)
```

```
# Initialize Modified CountSketch with mapping with w=20 and d=4
mcs_mapped = CountSketchWithMapping(w=20, d=4)

# Add items to the Modified CountSketch table
for category in df.iloc[:, 1]:
    mcs_mapped.add(category)

# Estimate the frequencies using Modified CountSketch with mapping
estimated_frequencies_mcs_mapped = {category: mcs_mapped.estimate(
    category) for category in category_counts.index}

# Sorting the estimated frequencies for plotting
sorted_estimated_frequencies_mcs_mapped = {k: v for k, v in sorted(
    estimated_frequencies_mcs_mapped.items(), key=lambda item: item[1], reverse=True)}

# Plotting the estimated frequencies using Modified CountSketch with mapping
plt.figure(figsize=(15, 10))
plt.bar(sorted_estimated_frequencies_mcs_mapped.keys(), sorted_estimated_frequencies_mcs_mapped.values(
), color='b', label='Estimated Frequencies (Modified Count Sketch with Mapping)')
plt.title('Estimated Frequencies by Modified Count Sketch with Mapping (w=20, d=4)')
plt.xlabel('News Category')
plt.ylabel('Estimated Frequency')
plt.xticks(rotation=75)
plt.grid(axis='y')
plt.legend()
plt.tight_layout()
plt.show()

# 3.B Compare the estimated frequency of all categories with their true frequencies from Q1(B).
# Fetch the true frequencies from Q1(B)
sorted_true_frequencies = category_counts.sort_values(ascending=False)
# Plot the estimated and true frequencies
plt.figure(figsize=(15, 7))

# Extract ordered categories from estimated frequencies
ordered_categories = list(sorted_estimated_frequencies_mcs_mapped.keys())

# Create the estimated and true values lists
estimated_values = [sorted_estimated_frequencies_mcs_mapped[category]
                    for category in ordered_categories]
true_values = [sorted_true_frequencies.get(
    category, 0) for category in ordered_categories]

bar_width = 0.35
index = range(len(ordered_categories))

bar1 = plt.bar(index, estimated_values, bar_width,
               label='Estimated (Count Sketch)', color='b', align='center')

bar2 = plt.bar([i + bar_width for i in index], true_values,
               bar_width, label='True Frequencies', color='r', align='center')

plt.xlabel('News Category')
plt.ylabel('Frequency')
plt.title('Comparison of Estimated (Count Sketch) and True Frequencies')
plt.xticks([i + bar_width / 2 for i in index], ordered_categories, rotation=60)
plt.legend()
plt.tight_layout()
plt.show()
```

```
# 3(C) Impact of bucket size w on absolute error

bucket_sizes = [10, 20, 30, 40]
average_errors_w = []

for w in bucket_sizes:
    # Initialize Modified CountSketch with mapping for each w
    mcs_temp = CountSketchWithMapping(w=w, d=4)

    # Add items to the Modified CountSketch table
    for category in df.iloc[:, 1]:
        mcs_temp.add(category)

    # Estimate the frequencies using Modified CountSketch with mapping
    estimated_temp = {category: mcs_temp.estimate(
        category) for category in category_counts.index}

    # Calculate the absolute errors for each category and then the average
    errors = [abs(estimated_temp[category] - category_counts[category])
              for category in category_counts.index]
    # average_errors_w.append(np.mean(errors))
    average_errors_w.append(sum(errors)/len(errors))

# Plotting the impact of w on average absolute error
plt.figure(figsize=(10, 6))
plt.plot(bucket_sizes, average_errors_w, marker='o', linestyle='-')
plt.title('Impact of Bucket Size w on Average Absolute Error')
plt.xlabel('Bucket Size (w)')
plt.ylabel('Average Absolute Error')
plt.xticks(bucket_sizes)
plt.grid(True)
plt.tight_layout()
plt.show()

# 3(D) Impact of number of hash functions d on absolute error

hash_functions_counts = [2, 4, 8, 16]
average_errors_d = []

for d in hash_functions_counts:
    # Initialize Modified CountSketch with mapping for each d
    mcs_temp = CountSketchWithMapping(w=20, d=d)

    # Add items to the Modified CountSketch table
    for category in df.iloc[:, 1]:
        mcs_temp.add(category)

    # Estimate the frequencies using Modified CountSketch with mapping
    estimated_temp = {category: mcs_temp.estimate(
        category) for category in category_counts.index}

    # Calculate the absolute errors for each category and then the average
    errors = [abs(estimated_temp[category] - category_counts[category])
              for category in category_counts.index]
    # average_errors_d.append(np.mean(errors))
    average_errors_d.append(sum(errors)/len(errors))

# Plotting the impact of d on average absolute error
plt.figure(figsize=(10, 6))
plt.plot(hash_functions_counts, average_errors_d, marker='o', linestyle='-')
plt.title('Impact of Number of Hash Functions d on Average Absolute Error')
plt.xlabel('Number of Hash Functions (d)')
plt.ylabel('Average Absolute Error')
plt.xticks(hash_functions_counts)
plt.grid(True)
plt.tight_layout()
plt.show()
```