KTH ROYAL INSTITUTE OF TECHNOLOGY

II2202

RESEARCH METHODOLOGY AND SCIENTIFIC WRITING

A Review on Text Data Formats for Time Series Data

Author
Emil STÅHL

Author
Steven SHIDI ZHOU

October 27, 2022

A Review on Text Data Formats for Time Series Data

STEVEN SHIDI ZHOU EMIL STÅHL
shidi | emilstah @kth.se

October 27, 2022

Abstract

Data science pipelines process vast amounts of data stored in various data formats. For pipelines to be high performing, it is important to choose the right data format for the given application. Since data formats are implemented using different technologies, they may differ in how they handle different sizes of data. In this work, we investigate the read-and-write performance, file stability, as well as file storage size of some of the most popular data formats including csv, xlsx, Avro, and Parquet. The output of the thesis is a wide overview of how the performance of data formats is impacted by aspects such as file size and file corruption due to bit flips. The obtained results show that the four benchmarked data formats behave differently. Specifically, the discrepancy shows that Parquet offers the best read and write as well as file store performance. However, in the file stability benchmark, Parquet performed the worst while csv showed the most resilience against file corruption due to bit flips. This means that there is a trade-off between read/write performance, file store size, and file stability. This work shows that Parquet is the data format best fit for handling time series data. Suggestions for future work include repeating the read-and-write benchmark by utilizing the Apache Spark framework instead of Pandas as well as using a different data set other than time series data.

Contents

1 Introduction

Data science pipelines process vast amounts of data stored in various data formats. In the field of the Internet of Things (IoT) alone, there are billions of sensors and computing devices that are collecting and analyzing large amounts of personal data [1]. For pipelines to be high performing, it is important to choose the right data format for your application. Since different data formats are implemented using different technologies, they may differ in how they handle different sizes of data [2]. However, there are currently no conventions or general knowledge regarding when to use a particular data format for a given data size to optimize for performance, file stability, and file size. In this work, we are going to investigate how data science pipelines can be optimized by choosing the correct format when processing textual time series data.

1.1 Background

The modern society produces a vast amount of data every day across many different types of industries and pipelines. Until processed, these data are merely bits placed on a storage medium. The field of data science is focused on extracting useful knowledge from such data in order to gain a deeper understanding of a given field. Data can be stored in various formats such as csv, xlsx, Parquet, and Avro. Processing data requires many operations to be applied, such as reading, writing, and other mathematical operations. The value that data science brings places heavy demands on the performance and stability of the data formats used in data processing and storage pipelines. Related work has been done by Plase et al., who in their work "*A comparison of HDFS compact data formats: Avro versus Parquet*" have benchmarked the amount of disk space these formats consume, as well as their performance in the High Energy Physics (HEP) analysis, which uses a lot of numerical data. However, the work does not benchmark read-and-write speed, file stability, or performance on time series text data. Understanding the performance advantages of different data formats in terms of read and write speed when working with time series text data is therefore of paramount importance in the data science community[3, 2].

1.2 Problem

Analyzing the performance and stability of different data formats is key to addressing the scalability and performance challenges in which data science pipelines operate. Today, there exist more than 44 zeta bytes of data with approximately 2.5 Quintilian bytes of data being generated each day [4]. Data science pipelines must be highly optimized performance-wise to be able to process data at sufficient speeds and to ensure stability as well as job efficiency [2]. Since different data formats are implemented using different technologies, they may differ in how they handle different sizes of data, which affects the performance when processing and storing the data. There are currently no conventions for when to use a particular data format for a given data size to optimize performance, file stability, and file size. Since csv, xlsx, Parquet, and Avro are all widely used in data science, it is important to have an understanding of their behavior when processing and storing data [2].

1.3 Purpose

This paper aims to give the interested reader guidelines regarding how to choose a data format when working with textual time series data. The purpose of this work is to analyze the performance of the csv, xlsx, Parquet, and Avro data formats by quantifying their read/write speed, file stability, and file size to determine which data format is the most optimal when working with textual time series data. By analyzing the performance of these data formats under different pipeline conditions, such as file size, one can use this work to tailor data science pipelines to optimize performance. The results are potentially of interest to general data scientists, stock market traders, developers of big data frameworks, and the open-source community.

1.4 Research questions and hypothesis

Considering the background and problem discussion, this work will treat the following questions.

- **RQ1** How do the csv, xlsx, Parquet, and Avro data formats differ in terms of read and write performance when processing textual time series data?
- **RQ2** How do the csv, xlsx, Parquet, and Avro data formats differ in terms of resilience against file errors due to random bit flips when storing textual time series data?
- **RQ3** How do the csv, xlsx, Parquet, and Avro data formats differ in terms of file size when storing textual time series data?

With this work, we expect there to be an advantageous data format for a given file size when working with textual time series data. The row-based Avro format should provide advantages in performance with regard to simple operations such as read/write speeds when compared to csv, xlsx, and Parquet. For larger time series data sets, the Apache Parquet format should provide better performance. Regarding file stability we expect csv to be more resilient against file errors due to that it is uncompressed.

1.5 Benefits, Sustainability, and Ethics

Many possible benefits may result from understanding the performance and stability of different data formats in data science. Regarding sustainability, the biggest contribution in this matter would be to reduce the energy consumption of data science pipelines. Smaller file sizes result in less network bandwidth being used, as well as less waiting time for the cluster to load and process data. This would hopefully be able to free up more machines which results in less energy being consumed and combined with works such as ”Green network control” it would have a meaningful environmental impact [2]. Sustainable energy, meaning less consumption, is one of the Sustainable Development Goals created by the United Nations [5]. Therefore, this work could result in positive sustainability impacts. Regarding ethics, there exist several areas that are interesting from an ethics perspective. For example, showing limitations in how a specific data format or software performs and functions may cause harm to the developers behind it. This may deter people from using it which may cause harm to the copyright holder or third-party software that relies on the data format in question. However, benchmarking is widely accepted in the software community and is often welcomed by the users and developers behind the software that is being tested. Therefore, we argue that there are no major ethical problems that deter us from researching this area [5].

1.6 Methodology

The decision of methodology is important in a scientific study. In this work, the research method used is defined as *experimental* since the method focuses on finding aspects that affect the choice of data format when working with textual time series data [6]. Moreover, the research is also *applied* since it makes use of the code for the data formats that we benchmark as well as libraries and existing methodologies, which means that the methodology is based on work performed by other researchers and developers [6, 7]. With this in mind, the methodology used to perform this work consists of the choice and configuration of a benchmarking environment, a method for performing the experiments, and an evaluation method to analyze and evaluate the system itself. The chosen method should fulfill the purpose and goal of this thesis, which is to analyze the performance of different data formats [6].

1.7 Delimitations

Due to the scope of this project as well as our knowledge and available resources, a set of delimitations are needed for this work. The delimitations can be constricted to those regarding the application of the data formats to be benchmarked. As the application may influence the choice of data format, we have decided to delimit our work to only benchmark the data formats when working with textual time series data using the Pandas Python library.

1.8 Outline

This paper is organized as follows. Chapter 1 gives an introduction to the work where the background is briefly explained. In addition to this, the purpose, the goal, and the methodology used are shortly explained along with its connection to ethics and sustainability. Chapter 2 is a recess of the theoretical framework and literature used in our work. Further on, Chapter 3 describes the methodologies and methods used to perform the work, which includes the benchmarking environment and metrics. Next, Chapter 4 presents the results of the performed benchmarks, mainly focusing on the performance and stability of the data formats and how these differ in-between file sizes. Chapter 5 analyses and discusses the results. In other words, what the results mean and why they look the way that they do. We also discuss how the experiments could be changed to improve the accuracy and reliability of the benchmarks. Finally, Chapter 6 presents a conclusion of the work in relation to its purpose and goal, as well as potential future work that can be performed in order to improve the conclusions of this work.

2 Theoretical framework

There are two types of data storage, row-based and column-based data formats. Row-based data stores are considered the traditional approach where we data is written one row at a time. This makes it very easy to insert new data or modify an existing record. However, with a row-based store, more data is read than is needed resulting in overhead. Column-based data, on the other hand, writes the data vertically by column. The column-based approach ensures that you only need to read the data you are interested in. However, the downside of this method is that it requires multiple accesses in order to write the records since their columns are distributed in different sections. Figure 1 below is a visual representation of the description above (Graph replicated from the presentation ”The future of column-oriented data processing with Arrow and Parquet”) [8, 2].

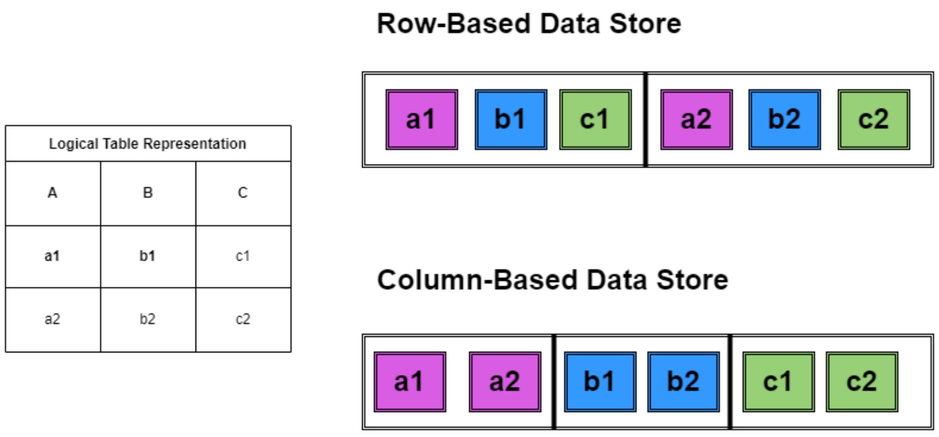


Figure 1: Row-based Data Store vs column-based data store. Replicate from [8]

2.1 Column-based data formats

The columnar data formats are a popular choice for fast analytic workloads. As opposed to row-oriented storage, columnar storage can significantly reduce the amount of data fetched from disk by allowing access to only the columns that are relevant to the particular query or workload. Moreover, columnar storage combined with efficient encoding and compression techniques can drastically reduce the storage requirements without sacrificing query performance [9]. It is worth noting that columnar data formats are suitable for large data volumes and read-only intensive workloads [8, 2].

2.1.1 Parquet

Apache Parquet is a column-based serialization format mainly used in the Hadoop ecosystem, regardless of the choice of data processing framework, data model, or programming language [10]. Parquet is built to support very efficient compression and encoding schemes. Instead of simply flattening the nested namespaces, it uses the record shredding and assembly algorithm described in the Dremel paper[11]. Multiple projects have demonstrated the performance impact of applying the right compression and encoding scheme to its data. Some projects, such as WikiParq, is an interesting project that builds on top of the Parquet format to tabulate and package the Wikipedia corpora[12]. Its content can then be extracted by querying the database to obtain the exact information the user needs. Besides the points above, Parquet also allows for compression schemes to be specified on a per-column level, and is future-proofed to allow adding more encoding as they are invented and implemented [10, 2].

2.2 Row-based data formats

Row-based data formats are ideal for visual data representation as well as transnational query. Examples of data formats used for visual data representation are csv and xlsx. Row-based formats are also very efficient and suitable for smaller data volumes. Below follows a presentation of each row-based data format that is to be benchmarked [13, 2].

2.2.1 Avro

Apache Avro is a compact binary data serialization format that provides various data structures and row-wise serialization[13]. Avro uses the JSON notation schema to serialize and deserialize data. Unlike some other similar systems, such as Google Protobuf, Avro does not require code generation and uses dynamic typing. Data are not tagged because the schema is accompanied with the data, resulting in a more compact data file. Avro supports versioning, so different versions of Avro data files can coexist along with their schemes. Since Avro is an Apache open-source project, many Apache projects support the Avro format such as Apache Hive, which provides support to store a table as Avro. Apache Flume supports Avro as input and output [10, 13, 2].

2.2.2 CSV

Comma Separated Value (CSV) is a very simple format where each record is located on a separate line, delimited by a line break (CRLF) [14]. Although simple, the format has been used to exchange and convert data between various spreadsheet programs. It is also very widely used as the format for storing and distributing data in the open-source data science community [14, 2].

2.2.3 Open Office XML-based spreadsheet format-XLSX

The Open Office XML-based spreadsheet format that uses .xlsx as a file extension has been the default format produced for new documents by Microsoft Excel versions since Excel 2007 [15]. The format was designed to be equivalent to the binary .xls format produced by earlier versions of Microsoft Excel [15, 2].

2.3 Previous comparative studies

Several previous studies have done research in this field. For example, Plase et al. compared Avro and Parquet with respect to text formats to evaluate the performance of the data queries [3]. Different data query patterns have also been evaluated in this study. Blomer performed a quantitative review on some common data formats, including Parquet and Avro, with respect to HEP datasets, which consist of n-tuples and Analysis Object Datas (AODs), and lots of numerical data[7]. Todor Ivanov and Matteo Pergolesi have in a journal article reasoned about the impact of using columnar-based file formats with SQL for Hadoop Engine performance, mainly comparing the ORC format vs Parquet [16, 2].

3 Methodology

In this chapter, we explain the different parts of the theoretical method used to benchmark the performance limits of the various data formats. We are going to execute three separate experiments, one corresponding to each of the research questions mentioned in 1.4. The data set that is to be used in this study is the Google Covid-19 Open Data [17]. The data is available in CSV and JSON formats. The source of this data set comprises authoritative sources, as well as volunteers and contributors. The Google Covid-19 dataset has multiple tables, but we are only going to use the aggregated table in this study. The aggregate table has a full version and subsets that provide only the data for the most recent day [17]. The full version, named aggregated_full.csv, is around 20 GB in size, and a subset version called v3-latest, named aggregated.csv, has a size of 24 MB. For a more rapid test and result collection, we are going to use the v3-latest version to do most of our benchmark operations. The v3-latest data set is going to be cut to have the size of 1 MB, 5 MB, 10 MB, 20 MB, and 24 MB and then be converted into the different file formats that are to be benchmarked in this work. Those files are then used for the read/write speed benchmark and file stability benchmark. For the benchmark of file store size, we will additionally convert the full version of the data, which weighs in at 20 GB, into other file formats and observe the file size changes.

3.1 Benchmarking environment

The chosen benchmark environment to perform the experiments is Google Colab, which allows anyone to write and execute arbitrary Python code through the browser and is especially well suited for machine learning, data analysis, and education [18]. We are going to write our benchmark scripts in the Python programming language since it is one of the most used languages in data science. Our Python code is going to automatically and empirically benchmark the read and write performance of the data formats by iteratively performing these operations 1,000 times and taking the average elapsed time that was required for the operation to finish. We will also be using some common Python packages such as the Pandas library for this experiment. The Google Colab environment has the following technical specifications [18]:

- CPU model name: Intel(R) Xeon(R) CPU @ 2.20GHz
- CPU cores: 2
- Available RAM: 12 GB
- Codec: Python_snappy-0.6.1-cp37
- Python Version: 3.7.14
- Python Pandas Version: 1.3.5
- Python Fastavro Version: 1.6.1
- Python Snappy Version: 0.6.1

3.2 Read & Write Performance

For the read-and-write performance tests, we take the v3-latest version of the Covid-19 data, as well as the subsets of the data generated from it. We use the Pandas library to help us with the read-and-write benchmark. For reading and writing Avro files, we combine Pandas and fastavro since Pandas does not provide direct support for Avro. Specifically, Avro files are read using the fastavro package and then converted to a Pandas dataframe afterward. Likewise, for writing, we first convert the Pandas dataframe to records, which is a list of dictionaries, and then use fastavro to help write those into an Avro file. It is worth noting that for Avro and Parquet file formats, they offer different compression codecs, in this study we are going to use the snappy codec for both formats. To address potential bias during data collection, the benchmark program is run 1,000 times in the Google Colab environment.

3.3 File stability

For the file stability tests, we are going to programmatically flip random bits in the files and check their resilience and ability to operate normally. This is to replicate the experiment of file stability done by Blomer with respect to HEP datasets, but for textual time-series data [7]. For flipping bits in the files, we make use of the bit-flipping software written by Antoine Grondin, and is available on GitHub [19]. The software lets us randomly flip one bit of a file. The range of possible outcomes of reading files with flipped bits consists of no change at all to a potential crash of the reading operation. For the cases where there are errors reading the file, we do another check where we compare the original file with the file that had a random bit flip; if these files have identical entries the outcome is reported as 'no effect'. However, if the file entries are not equal, the outcome is recorded as an 'undetected effect'. This procedure is executed iteratively 100 times, where we increment a counter for each possible outcome every time that specific outcome occurs, the results are saved to an excel spreadsheet that is shown in Table 4.2 of Section 4.2.

3.4 File store size

For this benchmark, since we have already created all the subset data we simply measure the size of the converted file formats. Additionally, we are also going to try to convert the 20 GB full version of the Covid-19 data into the other file formats, and observe the file size changes.

4 Results

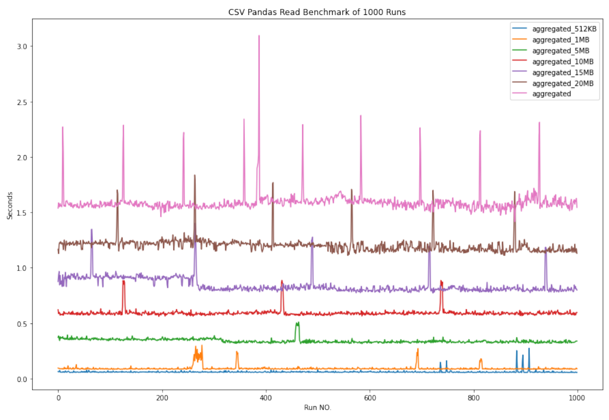
This chapter presents the results of the benchmarks performed.

4.1 Read/write performance

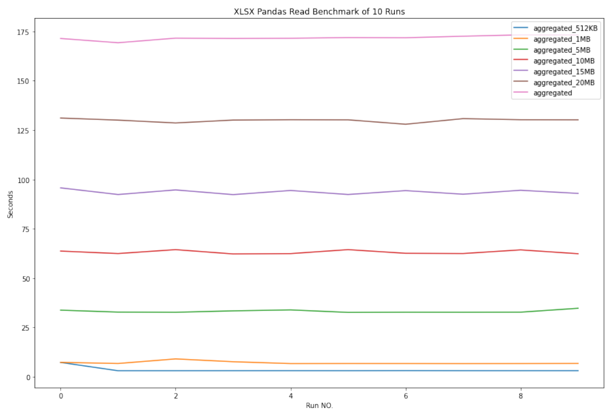
Due to the amount of data and the length of the tables, the results of the various read and write performance benchmarks for our files are shown in tables in Appendix **??** and **??**. The first row of each of these tables shows the filename of the benchmarked file, while each respective column shows the time in seconds it took to read the given file. These data are used to produce the summary tables and figures below. For the read benchmark shown in figure 2, the sub-figure of xlsx benchmark in section *b* is a lot smoother compared to the rest of the sub-figures, this is due to that the benchmark time for xlsx is significantly longer compared to the other file format benchmarks. For example, each read of the xlsx file named "aggregated(v3-latest)" which is around 24.1 MB in size takes an average of 172 seconds to finish. This means that if we proceed with the planned 1,000 iterations of each benchmark, it would take 48 hours to benchmark only the largest file. Therefore, we have chosen to benchmark it only 10 times for xlsx only. In the same situation as with the Avro benchmark in sub-figure *c*, the benchmark for the larger files, named aggregated_20MB and aggregated(v3-latest), are only run 100 times instead of 1,000. Figure 3 presents the summary of the results of the read benchmark using the Python Pandas library. As shown in sub-figure *a*, the xlsx file format shows significantly more time to finish reading compared to the other file formats, and the trend continues as the file size increases, suppressing the other file format metrics. After excluding the xlsx format, it becomes clear that the Parquet format has the best read performance compared to the rest of the benchmarked file formats since the read time increases at a very slow rate as the file size grows bigger.

Table 1: Pandas Read Benchmark

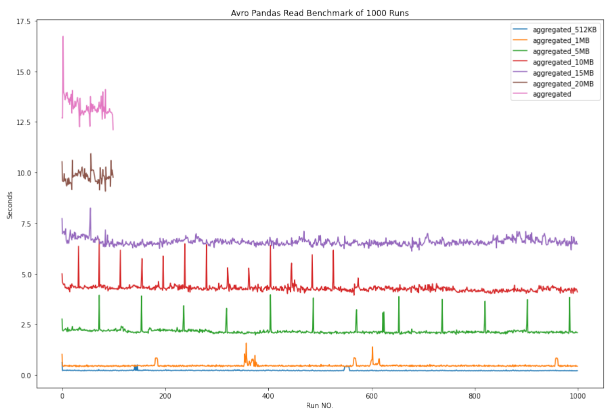
File Size	CSV Mean s	XLSX Mean s	Avro Mean s	Parquet Mean s
512KB	0.0603	3.4877	0.2238	0.0772
1MB	0.0925	7.0893	0.4659	0.0783
5MB	0.3404	33.1888	2.1599	0.1119
10MB	0.5921	63.1211	4.2898	0.1526
15MB	0.8415	93.6020	6.5762	0.1783
20MB	1.2019	129.9450	9.7963	0.2104
aggregated(v3-latest)	1.5918	171.7802	13.2546	0.2471



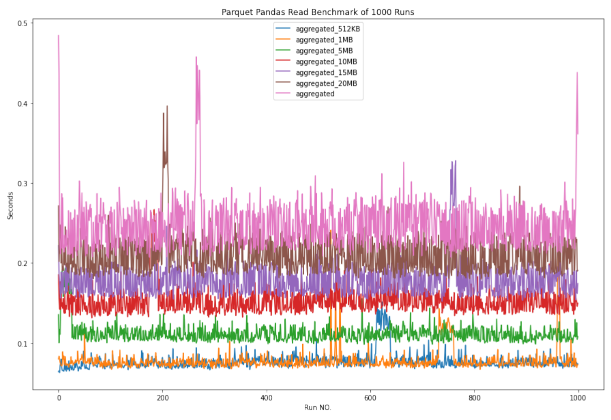
(a) CSV



(b) XLSX

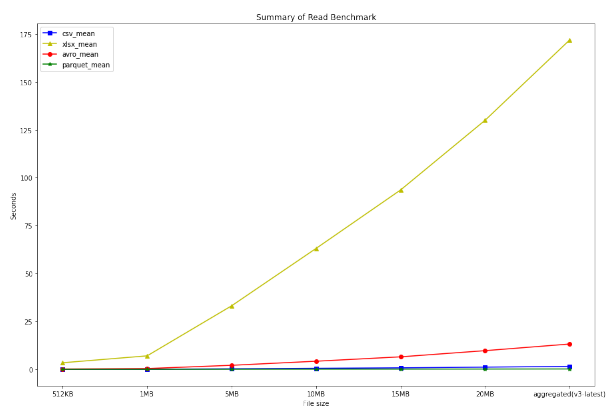


(c) Avro

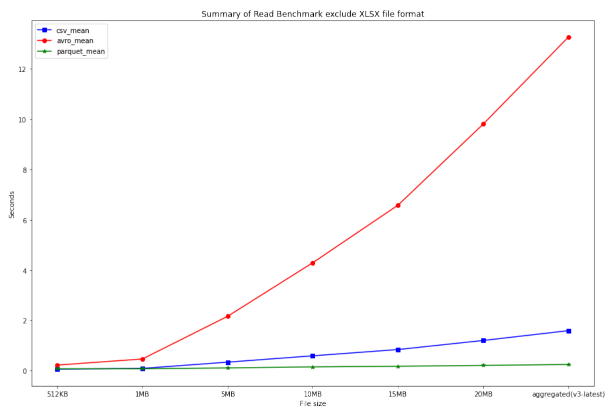


(d) Parquet

Figure 2: Pandas Read Benchmark for 4 different file formats, X-axis shows each runs up to 1000th run, Y-axis shows the time it takes for each run, measured in seconds. Sub-graph b looks smooth due to the number of iterations being 10 instead of 1000, and sub-graph c has cut off for bigger file sizes because for those it was only run the benchmark 100 times instead of 1000 due to time limitation



(a) Read Benchmark Summary



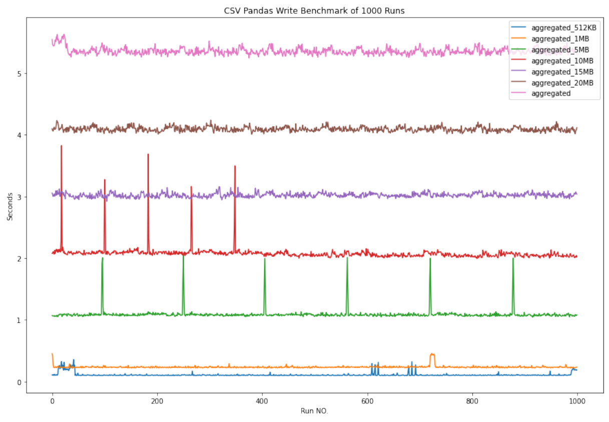
(b) Read Benchmark Summary excluding XLSX file format

Figure 3: Summary of read benchmark using Panda for 4 different file formats, the X-axis shows the file size, starting with 0.5MB up to 24.1MB (aggregated v3-latest), while the Y-axis shows the average time it took to read the file measured in seconds

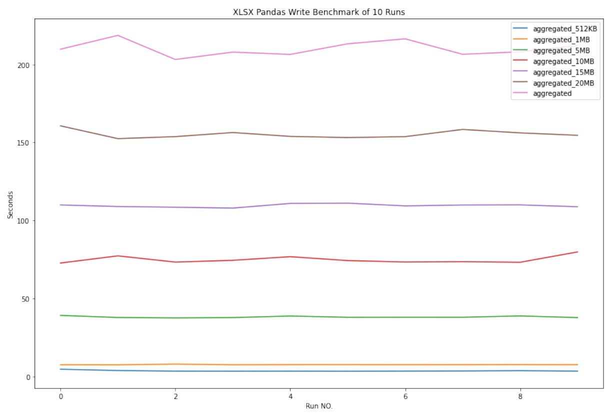
For the write benchmark, in figure 4, the xlsx benchmark in section *b* shows the same situation as its read counterpart. Meaning that we have chosen to benchmark it only 10 times for xlsx only. Due to time limitations for the Avro benchmark in sub-figure *c*, the benchmarks for all sizes are only run 100 times instead of 1,000. Figure 5 presents a summary of the results of the write benchmark using the Python Pandas library. As shown in sub-figure *a*, the xlsx file format takes significantly more time to finish writing compared to the other file formats, and the trend is only getting worse as the file size increases, suppressing the other file format's metrics. After excluding the xlsx format it becomes clear that Parquet format also has the best write performance compared to the rest of the benchmarked file formats, surprisingly the Parquet write time does not suffer as the file size increases.

Table 2: Pandas Write Benchmark

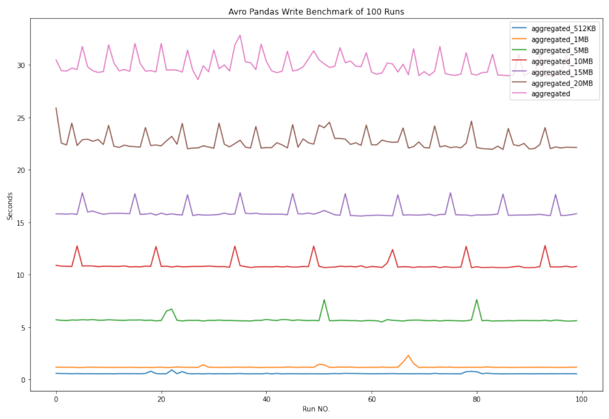
File Size	CSV Mean	XLSX Mean	Avro Mean	Parquet Mean
	s	s	s	s
512KB	0.1076	3.6311	0.5548	0.4839
1MB	0.2335	7.6040	1.1762	0.4801
5MB	1.0895	38.1422	5.6798	0.4818
10MB	2.0765	74.8746	10.8823	0.4852
15MB	3.0205	109.5140	15.9373	0.5005
20MB	4.0912	155.2620	22.6856	0.4890
aggregated(v3-latest)	5.3560	210.1604	29.9129	0.4852



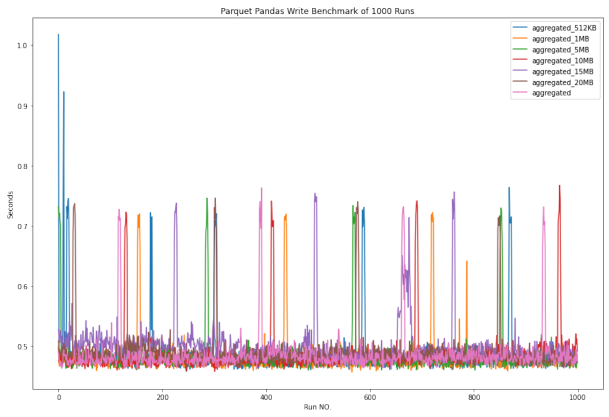
(a) CSV



(b) XLSX

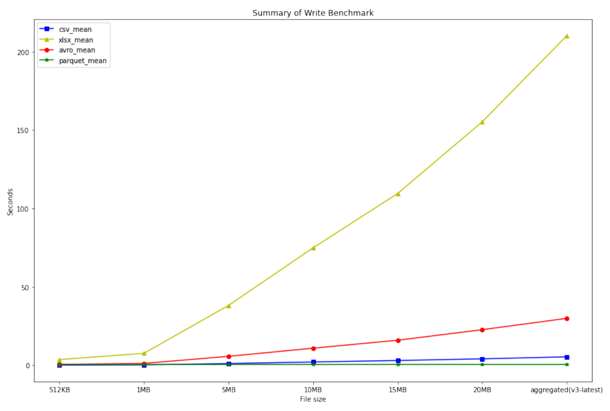


(c) Avro

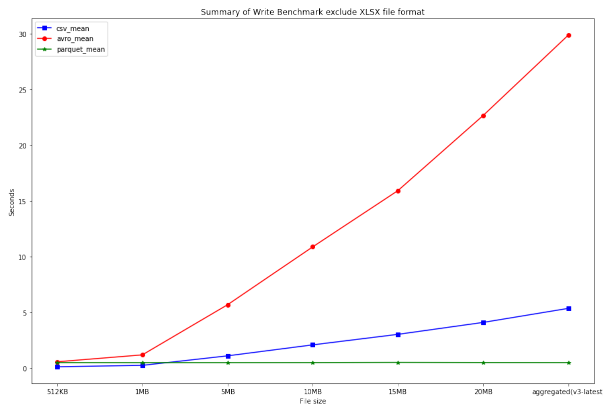


(d) Parquet

Figure 4: Pandas write benchmark for 4 different file formats, the X-axis shows each run up to the 1,000th run. The Y-axis shows the time it takes for each run measured in seconds. Sub-graph *b* looks smooth due to that the number of iterations is 10 instead of 1,000. Sub-graph *c* also looks smoother compared to the sub-graph *a* and *b* because the number of iterations are 100 times instead of 1,000.



(a) Write Benchmark Summary



(b) Write Benchmark Summary excluding XLSX file format

Figure 5: Summary of write benchmark using Pandas for 4 different file formats, the X-axis shows the file size starting with 0.5MB up to 24.1MB (aggregated v3-latest). The Y-axis shows the average time it took to write the file measured in seconds

4.2 File stability

In this section, we present data from our benchmark experiment regarding the file stability explained in Section 4.2. Due to the significant time it takes to benchmark the xlsx format, we have excluded it from this benchmark. Table 3 shows the result of flipping a random bit in each of the files we benchmark. The first column shows the filename, while the remaining columns specify the outcome of the benchmark, ranging from *Error*, *No effect*, and *Undetected effect*. When examining the data in Table 4.2, we notice that the csv files experience considerably fewer errors compared to Avro and Parquet. This is discussed in further detail in Section 5.

Table 3: Results of random bit flip when reading file

File Name	File Type	Error	No effect	Undetected effect
aggregated_512KB	csv	18	82	0
aggregated_512KB	avro	54	0	46
aggregated_512KB	parquet	22	0	78
aggregated_1MB	csv	14	86	0
aggregated_1MB	avro	59	0	41
aggregated_1MB	parquet	19	0	81
aggregated_5MB	csv	15	85	0
aggregated_5MB	avro	49	0	51
aggregated_5MB	parquet	22	0	78
aggregated_10MB	csv	13	87	0
aggregated_10MB	avro	58	0	42
aggregated_10MB	parquet	29	0	71
aggregated_15MB	csv	17	83	0
aggregated_15MB	avro	52	0	48
aggregated_15MB	parquet	31	0	69
aggregated_20MB	csv	12	88	0
aggregated_20MB	avro	60	0	40
aggregated_20MB	parquet	31	0	69
aggregated	csv	10	90	0
aggregated	avro	50	0	50
aggregated	parquet	24	0	76

4.3 File Store Size

The file size benchmark shown in Table 4 shows the file sizes of each file format when converted from csv format. Figure 7 provides a visualization of the table. Since the file size difference is too large, "aggregated_full" is therefore visualized in sub-figure *b* instead of merging it with sub-figure *a*. Furthermore, xlsx for "aggregated_full" is not shown in sub-figure *b* because of the lack of an effective method to convert a 20 GB csv file into xlsx format.

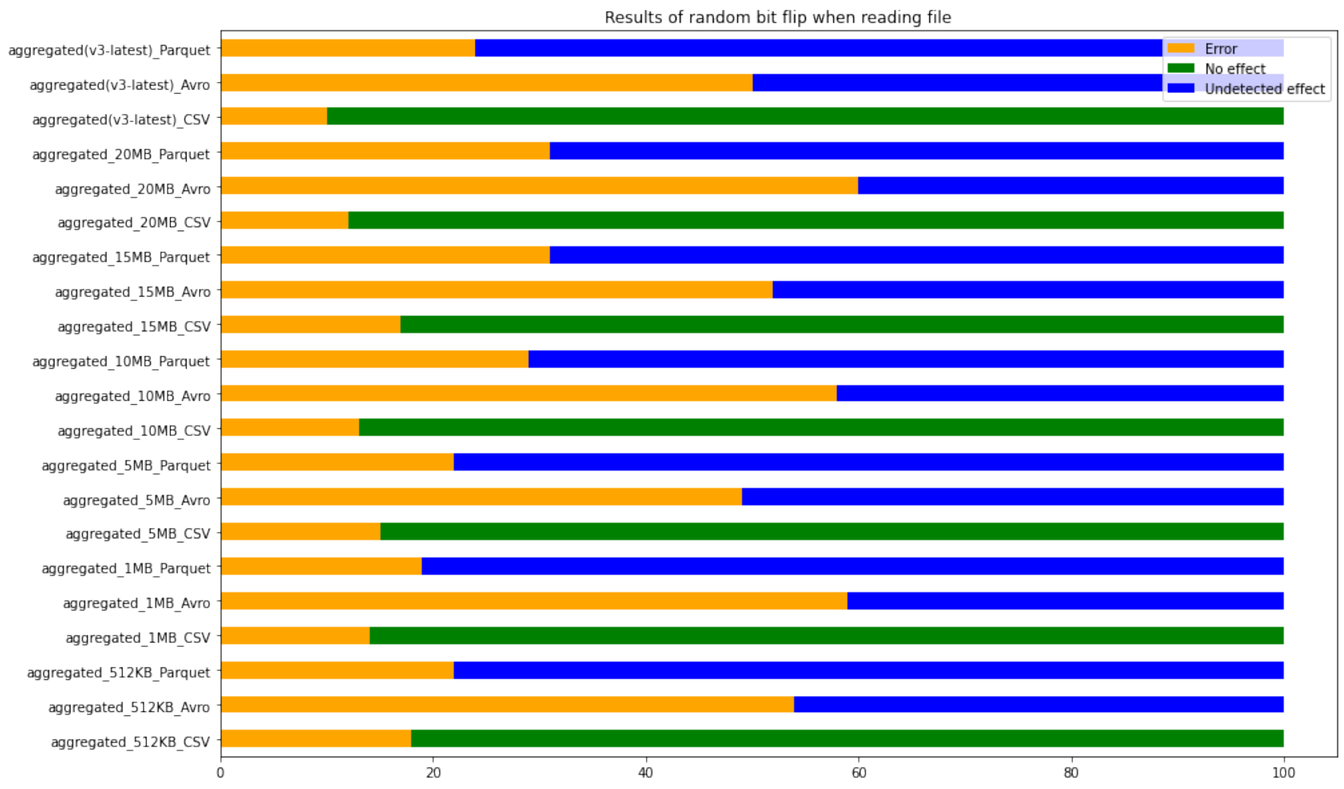
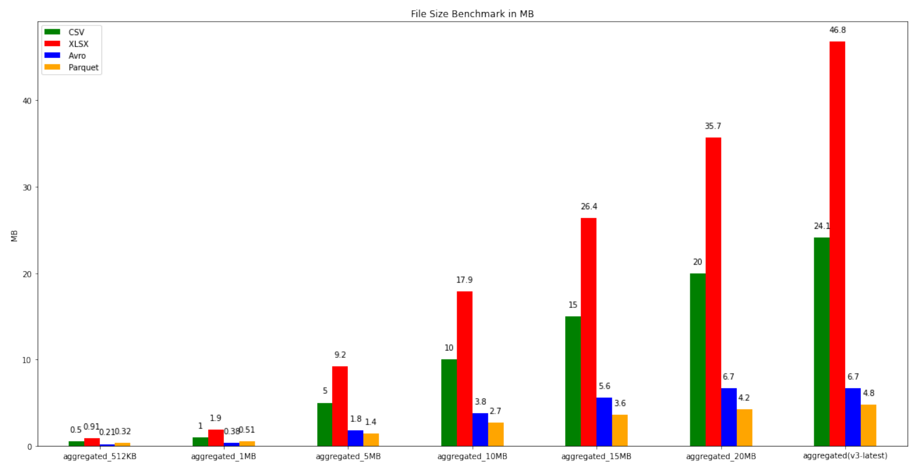


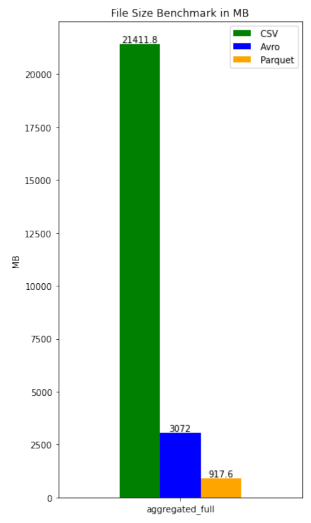
Figure 6: Summary of file stability benchmark using Pandas for 3 different file formats including Parquet, Avro, and csv). The X-axis shows the number of runs, as well as the percentage since total of 100 runs, that were performed for each file. The Y-axis shows the file size and format, starting with 0.5MB up to 24.1MB (aggregated v3-latest)

Table 4: File Store Size Benchmark

File Name	CSV (MB)	XLSX (MB)	Avro (MB)	Parquet (MB)
aggregated_512KB	0.5	0.90625	0.209961	0.318359
aggregated_1MB	1	1.9	0.383789	0.509766
aggregated_5MB	5	9.2	1.8	1.4
aggregated_10MB	10	17.9	3.8	2.7
aggregated_15MB	15	26.4	5.6	3.6
aggregated_20MB	20	35.7	6.7	4.2
aggregated (v3-latest)	24.1	46.8	6.7	4.8
aggregated_full	21411.84 (20.91GB)	Not convertible	3072(3GB)	917.6



(a) File Size Benchmark Summary



(b) File size benchmark summary for full data format

Figure 7: File size benchmark summary of the 4 file formats in different sizes listed on the X-axis. The file size is measured in MB and shown on the Y-axis)

5 Discussion

This section consists of two major sections that provide an analysis of the results presented in section 4 and respectively, an analysis regarding the different parts of the project as well as how they may have impacted the results.

5.1 Analysis of results

This section presents an analysis of the results shown in Section 4.

5.1.1 Read & Write Benchmark

For both the read and write benchmarks with the Pandas library for textual time series data, the Parquet format has the best performance. For reading, Parquet format read time only increases by around 0.04 seconds for every 5MB increase in file size, whereas the second format in place, the csv format, has an increase of around 0.25 seconds for every 5MB increase in file size. This is most likely due to the effective columnar storage style and the compression that is done with it. Interestingly, even thus the Avro file has the same compression codec, the file size is reduced but not as much as with the Parquet file format. For a discussion on file size and compression, see 5.1.3. The write performance of the Parquet format is astounding since the write time does not increase with the amount of data written. Parquet also manages to give the best compression rate compared to the rest of the file formats which is shown in Figure 6 7. This makes it a clear winner in this benchmark.

5.1.2 File stability

The data presented in Section 4.2 clearly shows that the csv data format experience considerably fewer reading errors and undetected effects when compared to Avro and Parquet. This is potentially due to that csv is a very redundant and uncompressed data format, which means that the randomly flipped bit is unlikely to have any meaningful impact if the flipped bit happens on the metadata or the comma used as separation. For files with compression, there is a higher probability of flipping an important bit that needs to be used to recreate and read the data correctly [13, 14, 2]. The results of the file stability benchmark show that the csv data format is more resilient against file errors and undetected effects due to bit-flip when compared with Avro and Parquet. Furthermore, when comparing the number of errors and undetected effects for Avro and Parquet it can be seen that Avro experiences more errors than Parquet. Regarding the cases where there were no file reading errors, Avro resulted in more undetected effects when compared to Parquet. Since Parquet performed the best in read/write and file size performance, our research shows that there is a trade-off between file stability and read/write performance as well as file size.

5.1.3 File Store Size

As shown in Figure 7, xlsx suffers the most in size when converted from the csv format while both Avro and Parquet show a huge file size reduction. Parquet seems to use the time series data the best since it managed to compress the 20.91 GB file into only merely 917.6MB. This is less than $\frac{1}{20}$ when compared to the original csv file, and less than $\frac{1}{3}$ when compared with the Avro format. This means that Parquet is once again the preferred format to its columnar storage style and the snappy compression codec.

5.2 Discussion

Although Parquet scored the lowest point out of the 3 file formats that were benchmarked in the file stability benchmark, the columnar-based storage format that is Parquet seems to be the best fit for textual time-series data. It has the best file size compression and read & write performance. Therefore, one can argue that the probability that a bit flip happens in a Parquet file transfer of 917.6 MB is less likely when compared to a transfer of a csv file with a size of 20.91 GB. Not to mention the savings on hard disk space as well as network bandwidth. One may argue that when it comes to big data analysis, it is implausible that people only use Pandas for data analysis, transformation, and visualization. Tools such as Apache Spark are more likely to be used in place. Furthermore, both Avro and Parquet formats support schema in their file format as well as schema evolution. Therefore, with Spark, we can more easily and effectively check for file corruption as well as get a valid schema from the file itself. This means that for future research purposes, one should consider running the same read/write and file stability benchmarks but with Apache Spark instead of Pandas [10, 2].

6 Research conclusion

Data formats are a critical component of data science pipelines to ensure high performance. Since different data formats are implemented using different technologies, they may differ in how they handle different sizes of data, which affects the performance when processing and storing the data. This means that it is of interest of understanding the potential limitations of various data formats used in data science [2]. This research has performed a set of benchmarks to reveal potential limitations in how various data formats perform in read/write speeds, resilience against potential file errors, and file storage size. The benchmarking methodology consisted of empirically and programmatically performing benchmarks with a set of Python scripts that iteratively executes the desired operations multiple times. The results were saved for further analysis and show that Parquet is the most suitable data format when it comes to handling textual time series data. This is because Parquet achieved the lowest read/write time as well as the smallest file storage size. However, due to the compressed nature of Parquet, it performed the worst in file stability benchmarks. Therefore, there exists a trade-off between read/write performance as well as file storage size and resilience against bit-flips that cause file errors. For future research purposes, one should consider running the same read/write benchmark but with Apache Spark instead of Pandas. Additionally, other applications rather than textual time series data may be researched to better understand the differences between the data formats when used in different data science applications.

References

[1] "State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally," May 2022. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[2] L. Cao, "Data Science: A Comprehensive Overview," *ACM Computing Surveys*, vol. 50, no. 3, pp. 43:1–43:42, Jun. 2017. doi: 10.1145/3076253. [Online]. Available: https://doi.org/10.1145/3076253

[3] D. Plase, L. Niedrite, and R. Taranovs, "A Comparison of HDFS Compact Data Formats: Avro Versus Parquet," *Science future of Lithuania*, vol. 9, no. 3, pp. 267–276, 2017. doi: 10.3846/mla.2017.1033 Place: Vilnius Publisher: Gediminas Technical University.

[4] J. Howarth, "30+ Incredible Big Data Statistics (2022)," Feb. 2022. [Online]. Available: https://explodingtopics.com/blog/big-data-stats

[5] M. Bexell and K. Jönsson, "Responsibility and the United Nations' Sustainable Development Goals," *Forum for Development Studies*, vol. 44, no. 1, pp. 13–29, Jan. 2017. doi: 10.1080/08039410.2016.1252424 Publisher: Routledge _eprint: https://doi.org/10.1080/08039410.2016.1252424. [Online]. Available: https://doi.org/10.1080/08039410.2016.1252424

[6] A. Håkansson, "Portal of Research Methods and Methodologies for Research Projects and Degree Projects." CSREA Press U.S.A, 2013, pp. 67–73. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-136960

[7] J. Blomer, "A quantitative review of data formats for HEP analyses," *Journal of Physics: Conference Series*, vol. 1085, p. 032020, Sep. 2018. doi: 10.1088/1742-6596/1085/3/032020 Publisher: IOP Publishing. [Online]. Available: https://doi.org/10.1088/1742-6596/1085/3/032020

[8] D. J. L. D. R. J. Jack, Jacques Nadeau, "The future of column-oriented data processing with Arrow and Parquet - ppt download." [Online]. Available: https://slideplayer.com/slide/14421705/

[9] A. Floratou, *Columnar Storage Formats.* Springer, May 2018. [Online]. Available: https://www.microsoft.com/en-us/research/publication/columnar-storage-formats/

[10] D. Vohra, "Apache Parquet," in *Practical Hadoop Ecosystem.* Berkeley, CA: Apress, 2016, pp. 325–335. ISBN 978-1-4842-2198-3

[11] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[12] M. Klang and P. Nugues, "WikiParq: A Tabulated Wikipedia Resource Using the Parquet Format," 2016, pp. 4141–, book Title: Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016). [Online]. Available: https://lup.lub.lu.se/record/5e2a7be2-9bc2-4d72-bec3-618939d0b729

[13] D. Vohra, "Apache Avro," in *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, D. Vohra, Ed. Berkeley, CA: Apress, 2016, pp. 303–323. ISBN 978-1-4842-2199-0. [Online]. Available: https://doi.org/10.1007/978-1-4842-2199-0_7

[14] Y. Shafranovich, "Common Format and MIME Type for Comma-Separated Values (CSV) Files," Internet Engineering Task Force, Request for Comments RFC 4180, Oct. 2005, num Pages: 8. [Online]. Available: https://datatracker.ietf.org/doc/rfc4180

[15] "XLSX Transitional (Office Open XML), ISO 29500:2008-2016, ECMA-376, Editions 1-5," May 2022. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000398.shtml

[16] T. Ivanov and M. Pergolesi, "The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet," *Concurrency and computation*, vol. 32, no. 5, p. n/a, 2020. doi: 10.1002/cpe.5523 Place: Hoboken Publisher: Wiley Subscription Services, Inc.

[17] "COVID-19 Open Data — Google Health." [Online]. Available: https://health.google.com/covid-19/open-data/

[18] "Google Colab," Jun. 2022. [Online]. Available: https://research.google.com/colaboratory/faq.html

[19] A. Grondin, "bitflip," Feb. 2022, original-date: 2020-04-09T19:05:11Z. [Online]. Available: https://github.com/aybabtme/bitflip

A Benchmark Results

Benchmark results are available in the results folder in the GitHub project, you can access the data via URL github.com/emilstahl97/Research-Methodology-and-Scientific-Writing-II2202