The Group Project of COMP30770

Programming for Big Data

Terrorism Trends

22435374 Cian Cockerill / 22529016**:** Sean Kilcullen / 22313726: Taisia Danilova
https://github.com/seankcn/TerrorismData

Section 1. Introduction

The main dataset for this project we are using is the Global Terrorism Database, which includes a list of various types of terrorist attacks in the period of 1970-2017 throughout the world.
The datasets we are using to compare with and find trends between these attacks are:

- Stock Price Data in US
- USA Real Estate Dataset
- Disneyland Reviews

Volume:
The volume of our project is just over 2GB of data total to process.

- Global Terrorism 162.81MB including 181,692 entries during 1970-2017. With 135 columns including dates, locations and details on attacks, etc.
- Stock Prices 1.64 GB total ETFs and Stocks, processing 7195 files of, specifically, stocks over past years. Includes 7 columns of data for each business over the years with the closing and opening stock prices, etc.
- Real Estate 178.86MB 2,226,383 entries of US Estate Listings as of 2024. Includes 12 columns of data about the prices, locations, status and details about the listings.
- DisneyLand reviews 32.06MB 42,000~ reviews of 3 disney branches over past years. Includes 6 columns of data, review ID, month, review details, rating, disney branch, etc.

Our biggest data would be the Stock Price dataset, with 585 seconds taken to process only the 1 Day/Stocks files with the traditional solution, with just over 1GB of data to process. This was run on a Ryzen 5 3600 Processor with 16GB RAM.

Variety:
Most of the data we use is structured. We generally aggregate the data by either the locations or dates from the terrorism dataset. We get values of prices from the real estate and stocks datasets and we get ratings from the reviews dataset. For the stocks dataset we used the unstructured data relating to attack types to calculate a separate structured data column for the severity of the attacks to make a better analysis on the trends of the stock changes for attack days. This allows

us to get a better understanding on how stock prices are affected by various attacks that have happened over the years. There are differences in the ways the datasets write their locations comparatively to the main terrorism dataset. In the Disney Reviews dataset, to get the specific disney branch in the area we are using for the attacks, the location is formatted like "Disney_Location", which has to be reformatted into just the location so the data can be merged seamlessly into the resulting dataset. Similarly, dates are formatted differently, for example, in the Disney Reviews dataset, the month and year are included together in a single string, with no date, whereas the terrorism dataset contains individual columns containing integers for day, month and year.

In general, while most data was structured, mostly quantitative for prices and ratings, we used unstructured data to gain more information about the attacks for a better understanding of the trends. Not only that but some qualitative data had to be reformatted to be able to aggregate and merge datasets.

Section 2. Project Objective

Value of our project:
The main value of our project is to find trends within terrorism attacks to gain better insights for decision making and planning for the future. Our project attempts to understand how terrorism affects public sentiment and the economy through real estate and stock markets.

We are interested in finding:

- Correlations between attacks and stock price changes for those days. This information is relevant for researchers, businesses and investors.
- Correlations between sentiment of reviews on days/months around time of the attacks for better information on how the public may be affected would allow businesses to better understand consumer sentiment and cut out noise.
- Correlations between real estate housing prices in affected areas over the years. This would be useful for researchers, businesses and investors in approaching new actions in these areas.

With these trends, stakeholders can get a better understanding in how to approach and plan for these days that can cause major disruption and changes to our daily lives.

Section 3. Traditional Solution

We mostly used python for our processing, but also used bash for parts of our stock price analysis and used google sheets for some of our visualisations.

Step 1: preparing the terrorism dataset

We first find relevant information from the main terrorism dataset, so that we only have the columns we need. For example, filtering data by only locations in the United States and getting rid of unnecessary columns like attack types, etc. Our remaining dataset will contain just the dates, locations and any other necessary information. We reformat data in columns if needed, to match other dataset, and calculate other necessary values into new columns, e.g. a severity value for each attack entry.

CODE:

```
Python
terror_infile = "globalterrorismdb_0718dist.csv" #terrorism file
with open(terror_infile) as terrorfile, open("clean.csv", 'w') as outfile:
#reads the terrorism file and outputs to the clean.csv file
    terrorReader = csv.reader(terrorfile)
    writer = csv.writer(outfile)
```

Basic cleaning process of the terrorism database, in this simple example it takes only the column country and filters entries only in the United States.

```
Python
    terrorHeader = next(terrorReader, None) #skips the header row
    for row in terrorReader:
        if len(row) < 12: #skips rows that are missing column data
            continue
        try:
            country = row[8].strip()
            if country != "United States": #filter by "United States"
                continue
```

In our Stock Analysis we created a new data column, calculating severity of each attack.

```
Python
       nkill_str = row[98].strip()
       nwound_str = row[101].strip()
       nkill = int(nkill_str) if nkill_str else 0
       nwound = int(nwound_str) if nwound_str else 0

       severity = nkill + nwound * 0.5 # Calculate severity by giving a weight
```

Then we wrote it into a new cleaner terrorism file that will be processed later.

```python
        writer.writerow([
                date, country, city, latitude, longitude,
                success, suicide, attacktype1_txt, nkill, nwound, severity
                ]) #writes row for each entry with these columns only
```

Step 2: preparing the comparison dataset

We do the same for the other datasets we are comparing, keeping only relevant columns and processing data into new columns if needed, i.e. averaging price for each location. We then perform any necessary formatting to match the terrorism dataset.

CODE:

The code for step 2 is almost identical to step 1 but applied to our other dataset.

Step 3: merging the datasets

We merge and compare the two datasets. We match the relevant columns, e.g. dates or locations, and form one aggregate dataset with the key information from each dataset.

CODE:

Data is taken from these files and then a new csv file is created merging data together to be later visualised with graphs. An example below shows data, taken from processed real estate csv file, taking total price per state and getting average and the number of attacks per state, taken from terrorist file, combined into one merged file with state, average price and number of attacks per state.

```python
        #merge the databases
    writer.writerow(["state", "avg_price", "num_attacks"])
        #merge terrorism and realestate by "state"
    for state, data in state_price.items(): #per state
        price = data["total_price"]/data["count"] #avg price
        attacks = state_attacks.get(state, 0)  #num attacks
        #write avg price and attack count
        writer.writerow([state, round(price, 2), attacks])
        #dataset for average price vs number of attacks per state built
```

Step 4: analysis and results

We visualise the information in the merged dataset and attempt to identify any trends relating to attack days/locations, i.e. average stock changes on attack days, average prices for attack days, etc. We measured the execution time for running our programs and generated our results

CODE:

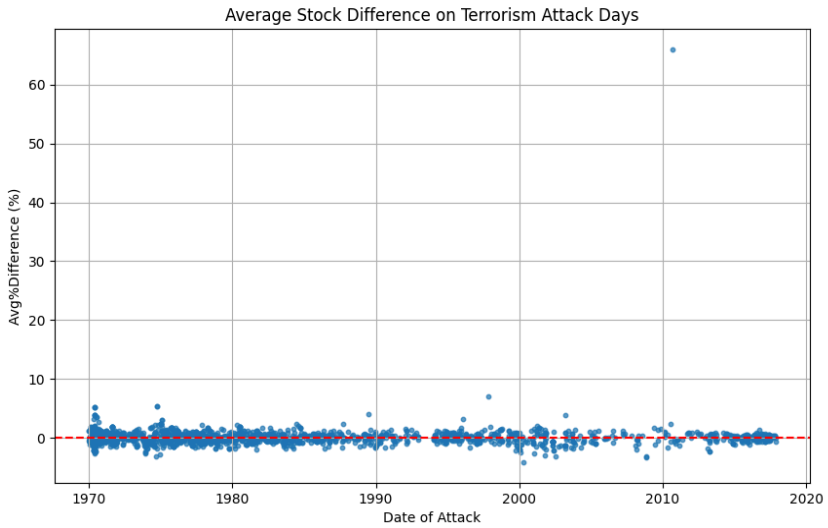
We did not need to use any code for visualisation, as we were able to simply import the data into Google Sheets or Excel and generate a chart showing any correlations between relevant columns. Execution time was measured using the time library.

```python
# top of script
import time
starttime = time.time()
# ...
# bottom of script
print(f"Elapsed Time: {time.time() - starttime} seconds")
```

RESULTS:

Stock Prices Analysis:

The Stock Price dataset took 585 seconds taken to process only the 1 Day/Stocks files with the python and bash script. This was run on a Ryzen 5 3600 Processor with 16GB RAM. The overall resulting trend for "Average Stock Difference on Terrorism Attack Days" is graphed throughout the years. As shown we don't see any trends during these days for stock prices.

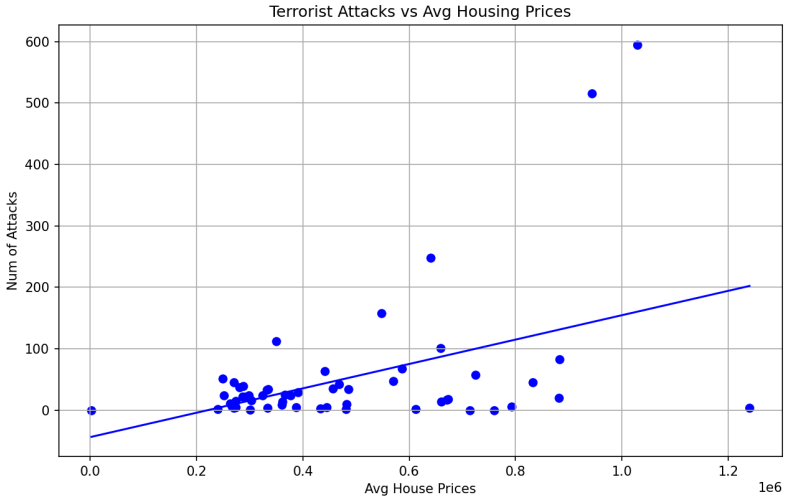


However, while this may seem unremarkable, it deviates from the typical behaviour. Average stock prices usually trend upwards. The NYSE has increased by an average of 0.89 points since its creation in 1965, so the lack of any discernible trend on this graph likely means that stocks perform worse on days where terrorist attacks occur.

Real Estate Pricing Analysis:

Real Estate dataset took around 9.4 seconds to process due to aggregation by states in America, creating a small dataset to sort and merge with only 50 entries. It was run on Intel N200 Processor with 4 cores and threads with 8GB RAM, written using python only. It took 15

seconds to run with the i5-7300U Processor. The “Terrorist Attacks vs. Average Housing Prices” plot graph below shows a upwarding trend with the amount of attacks correlating to higher housing prices in the area.

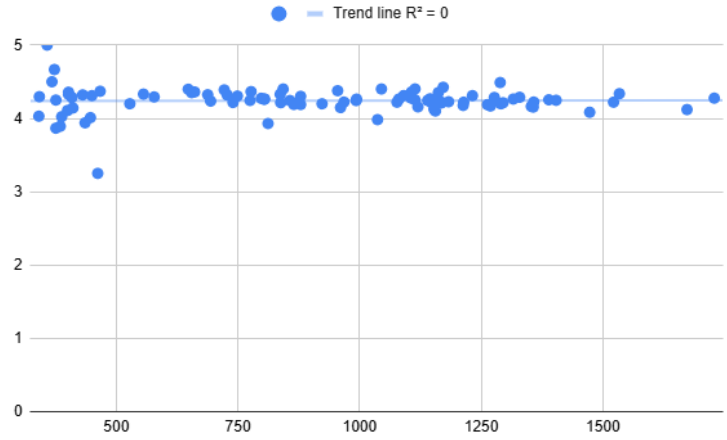


This could indicate that terrorist attacks happen in richer areas, but this trend might also be coincidental, since the real estate dataset is based on only the year 2024 and the attacks we compare against are within a range of years through 1970 to 2017. It would have been better to get pricings within a 5 to 10 year range after terrorist attacks happened in affected regions. Not only that, but population of the states was not a considered factor, which could mean this trend is not as reliable as we would have hoped.

Review Analysis:

Disney reviews dataset took 13.4 seconds to run on i5-7300U Processor with 2 cores, 4 threads and 8GB RAM. The plot graph below shows the trend for review ratings in Disneyland against the number of attacks during months between 2010-2017. There is no significant trend that shows, indicating that reviews are not correlated to terrorist attacks, and remain consistent.

Star rating vs number of terrorist attacks in a given month



Section 4. MapReduce Optimisation

The most time-consuming steps in Section 3 that could be optimised by MapReduce:
One of the main challenges when processing stock datasets is that each file must be read, preprocessed, and transformed individually, specifically steps 1 and 2 Operations such as skipping headers, parsing CSV records, and computing the difference between closing and opening prices for stocks are computationally expensive when executed sequentially. In addition, sorting the results by date becomes a significant bottleneck when dealing with a dataset of around 1GB. The merging step between two big datasets (step 3) can also be quite computational intensive.

How can it be optimised using MapReduce and our expectations:
By applying a MapReduce approach, we can overcome these challenges. The Map phase allows us to process each file in parallel, performing tasks like filtering, parsing, and calculating price differences simultaneously. The Reduce phase then efficiently merges and sorts the processed records by date. This distributed model leverages parallelism to significantly reduce the heavy computational load, particularly during sorting. For the Stock Analysis we expected around a 6x improvement in time with the Ryzen 5 3600 processor with 6 core CPU assuming complete equal parallelization across all cores. For the other datasets, Real Estate and Reviews, we didn't expect a significant change in the execution time, due to the datasets being a lot smaller than the Stocks dataset, being just over 1GB to process.

MapReduce Solution:
Both reviews and real estate datasets used PySpark to use MapReduce, while Stocks analysis used Spark using Java. We've included the PySpark code for the real estate dataset and explanation here, and the other code can be found on GitHub.

Initialisation:

```python
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.1-bin-hadoop3"

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True)
spark
```

```
import time
starttime = time.time()

sc = spark.sparkContext
real_infile = "realtor-data.zip.csv"
terror_infile = "globalterrorismdb_0718dist.csv"
output_file = "output"

#load files into rdd
real_rdd = sc.textFile(real_infile)
terror_rdd = sc.textFile(terror_infile)
```

Mapping and Reducing Data:

Python

```
#(state and price aggregation) for real estate
def process_real_estate(line):
    try:
        row = line.split(',')
        state = row[8].strip()
        price_str = row[2].strip()
        if state and price_str:
            price = float(price_str)
            return (state, price)  #output: (state, price)
        return None
    except Exception:
        return None

#rdd for real estate
real_estate_data = (
    real_rdd.map(process_real_estate)  #extract pairs
    .filter(lambda x: x is not None)   #filter empty vals
)
#aggregate prices per state and calculate average
state_price_rdd = (
    real_estate_data
    .mapValues(lambda price: (price, 1))  #map to (price, count)
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))  #sum price, counts
    .mapValues(lambda x: x[0] / x[1])  #get average price
)

#(state and attack count aggregation) for terrorism
def process_terrorism(line):
    try:
```

```
        row = line.split(',')
        country = row[8].strip()
        if country != "United States":
            return None
        state = row[11].strip()
        return (state, 1)  #(state, 1 attack)
    except Exception:
        return None

#rdd for terrorism
terrorism_data = (
    terror_rdd
    .map(process_terrorism)  #get pairs
    .filter(lambda x: x is not None)   #filter empty vals
    .reduceByKey(lambda a, b: a + b)   #sum attack count per state
)

#join both rdds
merged_rdd = state_price_rdd.leftOuterJoin(terrorism_data)
```

Joining and Creating Data files:

```
Python
#join both rdds
merged_rdd = state_price_rdd.leftOuterJoin(terrorism_data)

try:
    (
        merged_rdd #format "state,avg_price,attack_count"
        .map(lambda x: f"{x[0]},{x[1][0]},{x[1][1] or 0}")
        .coalesce(1)  #combine into one file
        .saveAsTextFile(output_file)
    )
    print(f"Data successfully saved to {output_file}")
except Exception as e:
    print(f"Error saving data: {e}")

print(f"Elapsed Time: {time.time() - starttime} seconds")

sc.stop()
```

Our MapReduce results:

Stock Price MapReduce Results:

When compared to a sequential Bash script, which took about 585 seconds to process the dataset, the MapReduce solution reduced the processing time to approximately 75.3 seconds, a 7.5x improvement.

Real Estate Pricing MapReduce Results:

When comparing the Python script, which took about 9-10 seconds to process the compressed dataset. The MapReduce solution using PySpark actually increased the processing time to approximately 19-22 seconds, a 2x deterioration. When ran on

Review MapReduce Results:

When using the Python script, processing took about 13.4 seconds to produce its output file, whereas using MapReduce, the process only took about 6.9 seconds, a 2x improvement.

Our results vs. our expectations:

The Spark solution uses parallel processing during the Map phase and distributed sorting during the Reduce phase, making it ideally suited for big data environments. This is shown for the Stocks dataset. The difference from the expected 6x improvement can be due to a few factors. First, Spark's efficient resource utilization and task scheduling resulted in better performance, improved cache usage and reduced I/O overhead. Second, the sequential Bash script has extra overheads in file I/O and sorting operations, which worsens its inefficiencies compared to a distributed processing model. Lastly, processing the 1GB dataset in memory further accelerates computations in Spark, reducing disk dependency and improving performance.

Unlike the Stocks dataset, the Real Estate dataset analysis with MapReduce showed proof of the reason Spark is better used for bigger data environments. Spark's overhead, such as the initialisations, setting up tasks and shuffling data, outweighs the benefits of parallel processing. The traditional solution using Python is a lot faster since it loads data into memory and loads it directly without any setup. This would explain the traditional solution processing the data in just 10 seconds, while MapReduce takes around 20 seconds.

Surprisingly, this was not the case for the Reviews dataset. We would expect it to take longer as it is the smallest dataset. One potential explanation for this is the configuration and specifications of the computers running these tasks. We noticed that when we ran the Real Estate code on the same computer, the 2 core 4 thread Intel i5 7300U, that it took 15 seconds to run in pure Python and 7 seconds to run on PySpark. This might mean that a traditional Python solution makes better use of multiple cores than PySpark, or simply that PySpark wasn't misconfigured to properly take advantage of the additional resources.