COMP1204: Data Management
Coursework One: Hurricane Monitoring

Shaho Zagrosi
33381836

March 17, 2023

1 Introduction

The ability to conduct an efficient, meaningful analysis and interpretation of information is crucial in the realm of academic research and often relies on data hygiene. This necessitates presenting facts and figures in a manner that enables automated processing, sorting, and formatting for large-scale research purposes. Data cleaning, a process that entails transforming raw data by isolating it from irrelevant details and technicalities, plays a pivotal role in achieving this goal. The significance of data hygiene cannot be emphasized enough; incomplete or inconsistent data presentation may result in an inaccurate comprehension of the information, ultimately leading to erroneous conclusions. Researchers depend on intelligent data handling to comprehend data on a large scale, and by effectively managing data processing, larger data-sets can be understood more expediently.

This study aimed to enhance the comprehension of data cleaning and processing techniques within the context of hurricane data. The primary objective was to develop a `bash` script capable of efficiently parsing Keyhole Markup Language (KML) files. These KML files comprise valuable storm data, alongside extraneous metadata and tags. The success of this endeavor hinged on the ability to identify and extract pertinent information encoded within the KML files while eliminating the irrelevant sections. To accomplish this, an analysis of data patterns was conducted, followed by the employment of the `bash` scripting language to automate the scanning, scraping, and parsing of the KML files. This procedure resulted in the generation of well-structured data tables, which can be utilized to produce storm map plots from any similar KML file.

The methodology implemented in this study integrates theoretical concepts and practical skills in data processing and analysis. The process encompassed data exploration, manipulation, and cleaning—fundamental aspects of data science. Through this project, I have acquired invaluable experience in handling real-world data and deepened my understanding of data quality's importance and its influence on subsequent analyses.

The findings of this coursework possess applicability to a broad spectrum of data processing scenarios beyond the hurricane data used in this project. The techniques and tools employed are vital for extracting valuable insights from intricate data-sets, particularly in situations where data quality may be a concern. Additionally, the development of the `bash` script showcased in this project offers an efficient and practical solution for KML file processing, which may prove advantageous for researchers and practitioners operating in fields that utilize KML files as a data source.

2 CSV Script Generator

The following report outlines the development of a script for scraping, processing, and tabulating hurricane data. My script was designed to recognize repeating patterns in `kml` files and extract relevant information such as timestamps, latitude, longitude, minimum sea level pressure, and maximum intensity. In this report I will elaborate on my work on the script, including explanations of relevant code snippets and a brief overview of its functionality, implemented using the `bash` shell scripting language.

2.1 Scraping Input File

A `create_csv.sh` file was used as the main script file, in which the `bash` shell script searches for relevant data tags found in the KML files alongside relevant pieces of data. Input and output files were defined as variables based on the arguments parsed, using the terminal command format

```
./create_csv.sh <inputkmlfile.kml> <outputcsvfile.csv>
```

The script started by using the `echo` pattern of `bash` to define the table header, ensuring that column headings would always appear on the first line of the output document and data as data within would be displayed in the same order.

```
#!/bin/bash

input_file="$1"
output_file="$2"

echo "Timestamp,Latitude,Longitude,MinSeaLevelPressure,MaxIntensity" > "$output_file"
```

A while loop with an Internal Field Separator (`IFS`) was used to read the input document line-by-line. Scraping, trimming, and tabulating data were undertaken using a series of `if` statements that checked for relevant data tags in the KML files, subsequently using `sed` to remove excess code, tags, and white-space. Multiple `if` and `elif` statements were used for this purpose, using the same general principle for each tag (`<dtg>`, `<lat>`, `<lon>`, `<minSeaLevelPres>`, and `<intensity>`). The values in which the data was stored were recognized and stored in corresponding variables for use later.

```
while IFS= read -r line; do
    if [[ $line =~ "<dtg>" ]]; then
        dtg=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
```

In the case of latitude and longitude, `sed` trimming was especially required as a nested ternary operator was used to print corresponding cardinal directions (N/S and E/W respectively). This is due to the use of a ternary operator, which would not properly operate if the data is not parsed purely numerically. In other words, the tags would interfere with the ternary operator and always return false if the numerical data is presented with tags and white-space included.

```
elif [[ $line =~ "<lat>" ]]; then
        lat=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
        lat=$(awk -v lat="$lat" 'BEGIN {printf "%.1f %s", lat, (lat >= 0 ? "N" : "S")}')
```

2.2 Producing Output File

A final `if` statement was used to print rows of data in corresponding order to the header column. The script ensured that all data was present in each row, thus preventing inconsistent output. Finally, variables for each data category were reset, their respective contents cleared for the script's next row iteration.

```
if [[ -n $dtg && -n $lat && -n $lon && -n $pres && -n $intensity ]]; then
        echo "$dtg,$lat,$lon,$pres,$intensity" >> "$output_file"
        unset dtg lat lon pres intensity
    fi
```

The client is then notified of successful script operation with the use of AWK's `END` pattern, printing the number of lines processed (`NR`) to the console.

```
echo "Finished processing $(wc -l < "$input_file") lines."
```

Thus, an output in the following format can be expected:

```
Timestamp,Latitude,Longitude,MinSeaLevelPressure,MaxIntensity
1800 UTC MAY 16,28.0 N,-78.7 W,1008 mb,30 knots
0000 UTC MAY 17,28.9 N,-78.0 W,1006 mb,35 knots
0600 UTC MAY 17,29.6 N,-77.6 W,1004 mb,35 knots
1200 UTC MAY 17,30.3 N,-77.5 W,1003 mb,35 knots
1800 UTC MAY 17,31.0 N,-77.3 W,1003 mb,40 knots
0000 UTC MAY 18,31.9 N,-77.0 W,1003 mb,40 knots
0600 UTC MAY 18,33.1 N,-76.7 W,1002 mb,40 knots
1200 UTC MAY 18,34.4 N,-75.9 W,1000 mb,45 knots
1800 UTC MAY 18,35.5 N,-74.7 W,993 mb,45 knots
0000 UTC MAY 19,36.2 N,-73.1 W,991 mb,50 knots
0600 UTC MAY 19,36.8 N,-71.4 W,990 mb,50 knots
1200 UTC MAY 19,37.0 N,-69.5 W,989 mb,55 knots
1800 UTC MAY 19,36.9 N,-67.8 W,991 mb,50 knots
0000 UTC MAY 20,36.2 N,-66.8 W,993 mb,50 knots
0600 UTC MAY 20,35.5 N,-66.0 W,997 mb,50 knots
1200 UTC MAY 20,34.6 N,-65.6 W,1002 mb,50 knots
1800 UTC MAY 20,33.7 N,-65.3 W,1006 mb,45 knots
0000 UTC MAY 21,32.8 N,-65.0 W,1008 mb,40 knots
```

2.3 Final Bash Script

The final `bash` script (`create_csv.sh`) that produced this output utilising all the aforementioned steps. Note that `chmod +x create_csv.sh` was used to make the script executable.

```
#!/bin/bash

input_file="$1"
output_file="$2"

# Print header
echo "Timestamp,Latitude,Longitude,MinSeaLevelPressure,MaxIntensity" > "$output_file"

while IFS= read -r line; do
    if [[ $line =~ "<dtg>" ]]; then
        dtg=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
    elif [[ $line =~ "<lat>" ]]; then
        lat=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
        lat=$(awk -v lat="$lat" 'BEGIN {printf "%.1f %s", lat, (lat >= 0 ? "N" : "S")}')
    elif [[ $line =~ "<lon>" ]]; then
        lon=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
        lon=$(awk -v lon="$lon" 'BEGIN {printf "%.1f %s", lon, (lon >= 0 ? "E" : "W")}')
    elif [[ $line =~ "<minSeaLevelPres>" ]]; then
        pres=$(echo "$line" | sed 's/<minSeaLevelPres>//; s/<\/minSeaLevelPres>//; s/^[[:space:]]\+//; s
        pres="${pres} mb"
    elif [[ $line =~ "<intensity>" ]]; then
        int=$(echo "$line" | sed 's/<[^>]*>//g; s/^[[:space:]]\+//; s/[[:space:]]\+$//')
        intensity="${int} knots"
    fi

    if [[ -n $dtg && -n $lat && -n $lon && -n $pres && -n $intensity ]]; then
        echo "$dtg,$lat,$lon,$pres,$intensity" >> "$output_file"
        unset dtg lat lon pres intensity
    fi
done < "$input_file"
```

```
echo "Finished processing $(wc -l < "$input_file") lines."
```

3 Storm Plots

With correct output of `csv` files from the `create_csv.sh` script, the `create_map_plot.sh` script provided allows for the generation of graphical storm plots corresponding to the `kml` files provided.

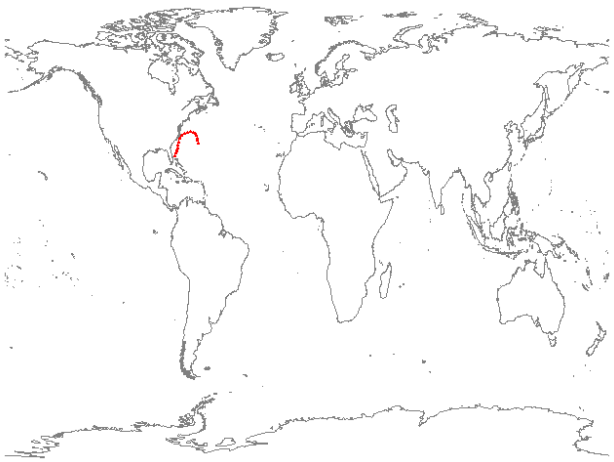

Figure 1: Map Plot (AL012020)

The above map shows the propagation of a storm along the east coast of North America, possibly migrating northwards along the state of Florida. Historically speaking, this appears to be data from Tropical Storm Arthur, a cyclone measuring speeds up to 60 miles per hour and affecting vast areas of the United States of America and the Caribbean. This storm was observed between the 16th to the 19th of May 2020.

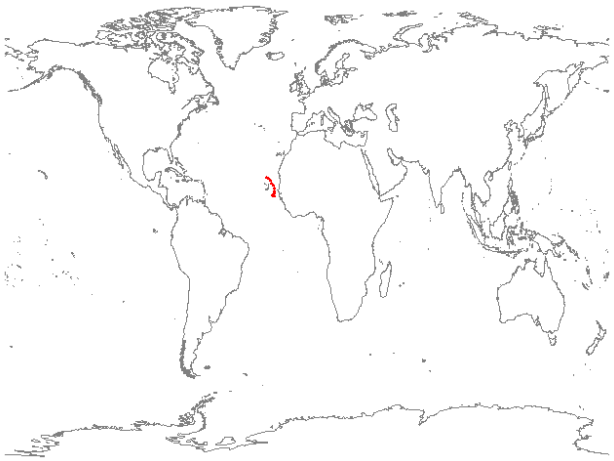

Figure 2: Map Plot (AL102020)

This hurricane is seen moving westward to the west coast of Africa, possibly in the direction of Mauritania. This was named Tropical Depression Ten, a storm originating off the west coast of Western Africa and later dissipating offshore between the 31st of July and the 1st of August 2020.

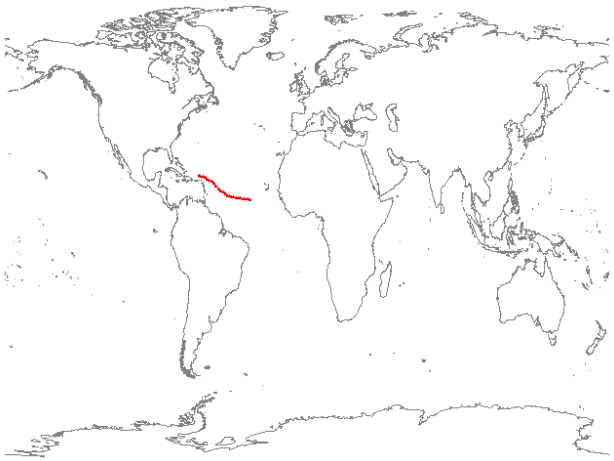

Figure 3: Map Plot (AL112020)

The cyclone pictured above appears to show a wide storm front pushing through the Central Atlantic. This was actually documented as Tropical Storm Josephine, a short-lived storm storm observed between the 11th to the 16th of August 2020.

It is evident from my analyses above that I am no expert in meteorology nor paleotempestology, but my script was successful in transforming cluttered markup text into easily understandable tabulated data and, with the help of the aforementioned script supplied, present this data graphically in such a way that can be generally understood and commented upon by both laymen and professionals alike.

4 Git Usage

Appropriate use of Git for both version control and incremental updates is helpful for project management, especially in projects in which multiple people are collaborating. In utilising `git`, I organised and tracked updates to my code, primarily using `git commit` and `git push` commands.

```
commit 0ff0342419db2ff8d93daa090afb0bd8ee179081
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Tue Mar 14 03:16:48 2023 +0000

Commit 10: Output respective csv tables for kml files, alongside png map
plots.

commit b73c6512631c1f9ada745a2bc2e059256871b415
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sun Mar 12 20:20:54 2023 +0000

Commit 9: Updated python-plot-script.py file to remove branching error
code.

commit 05825c26f53a00add258c3ab781c9077965f2989
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sun Mar 12 20:20:29 2023 +0000

Commit 8: Initial creation of python-plot-script from working branch.

commit 524ccf98a1997ae3a18fb49fa8a43427b2b85cfc
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sun Mar 12 20:20:28 2023 +0000

Commit 7: Cleaning old scripts.

commit cfbf3db3bffdf62207a92bca3e2ac4df694bf9c3
Merge: 5f0d4cc 2a82ea0
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 19:17:52 2023 +0000

Commit 6: Merge branch 'python-addon'.

commit 5f0d4cc478e628f3a98bd804b6d96252f273203f
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 19:17:04 2023 +0000

Commit 5: Attempting to resolve git merge conflict between master and
python-addon branches respectively.

commit 994338f09d8d6e19b87aad1a27844f434041c2c5
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 19:05:22 2023 +0000

Commit 4: Initial creation of python-plot-script from working branch.

commit 2a82ea0ae7977ebbf660a43d340306b169ba1db3
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 19:05:21 2023 +0000
```

```
Commit 3: Added new function to the python-plot-script from python-addon.

commit b67944af045bc66d9e28a6faf2ca0dea9c8c8b53
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 19:05:21 2023 +0000

Commit 2: Cleaning old scripts.

commit d32176e8ef18d6e08d56e23bf17bdbb597c02366
Author: Zagrosi <sz10g21@soton.ac.uk>
Date:   Sat Mar 11 18:59:37 2023 +0000

Commit 1: Completed create_csv.sh file to perform required function.
```

By keeping a numbering scheme for each `commit`, I was able to easily sort each update and revert accordingly if required. No reversions were required in this project, however major changes such as merges were carried out which resulted in conflicts to be resolved; should I have encountered any serious problems, it would have been easy to roll-back my project to an earlier version. This was especially important considering the `git` conflicts that arose when merging the `python-addon` branch with the `master` branch in which the main project files were located. This resulted in both errors in the terminal and in the `python-plot-script.py` file, which I later cleaned up. The conflict resulted in `git` errors pointing between the `master` and `python-addon` branches respectively, materialising with `<<<<<<< HEAD`, `=======`, and `>>>>>>> python-addon` markers inside the `python-plot-script.py` file. The following code block shows the final file contents after the cleanup.

```
import pandas as pd
import matplotlib.pyplot as plt
import os
import glob
import math

user_key = 18429


def plot_all_csv_pressure():
    path = os.getcwd()
    csv_files = glob.glob(os.path.join(path, '*.csv'))

    for f in csv_files:
        storm = pd.read_csv(f)
        storm['Pressure'].plot()
        plt.show()
```