COMP1204: Data Management
Coursework One: Hurricane Monitoring

PROKOPIS GEORGIOU
STUDENT ID: 34077499

NATIONAL OCEANOGRAPHIC AND ATMOSPHERIC ADMINISTRATION CENTRE TROPICAL CYCLONE TRACKING SYSTEM

Source: Adobe Stock *https://stock.adobe.com*

1 Introduction

This report explains how to convert raw data from tropical cyclone reports, initially in KML format, into structured CSV files using UNIX commands. This conversion is crucial for making the data usable and accessible for further analysis and visualization. The goal is to extract specific information from the reports, such as timestamps, storm latitude, longitude, minimum sea level pressure, and maximum intensity. This information is then formatted into CSV files. Additionally, the report includes visual representations generated by StormPlots, with each plot captioned to specify the corresponding KML file. The report details the approach taken to achieve this task, including the scripting process, conflict resolution using Git, and a summary of the work undertaken. This process ensures that the data is easy to understand and useful for researchers and analysts, enabling them to explore tropical cyclone patterns effectively. Continuous exploration and improvement of UNIX-based data manipulation techniques are recommended for ongoing enhancements.

2 Create CSV Script

This script takes two parameters: an input file (tropical cyclone report) and an output file (CSV file). It extracts specific information from the input file, such as timestamps, latitude, longitude, minimum sea level pressure, and maximum intensity. Then, it combines this data and formats it into CSV format with appropriate headers. Finally, it writes the formatted data into the output file specified. The script also includes error handling to ensure proper usage.

Listing 1: create_csv.sh

```
#!/bin/bash

# Check if input and output file parameters are provided
if [ "\$#" -ne 2 ]; then
    echo "Usage: \$0 input_file output_file.csv"
    exit 1
fi

echo "Converting \$1 --> \$2"

# Extract data from input file
timestamp=$(cat "\$1" | grep 'UTC' | sed 's/.*[0-9]\{4\} UTC [A-Z]\{3\}
    ↪ [0-9]\{2\}.*/\1/' | uniq)
latitude=$(grep -Eo '[-]?[0-9]+\.[0-9]+ N' "\$1")
longitude=$(grep -Eo '[-]?[0-9]+\.[0-9]+ W' "\$1")
minSeaLevel=$(grep -Eo '[0-9]+ mb' "\$1")
knots=$(grep -Eo '[0-9]+ knots' "\$1")

# Combine data
output=$(paste -d "," <(echo "\$timestamp") <(echo "\$latitude") <(echo "\
    ↪ $longitude") <(echo "\$minSeaLevel") <(echo "\$knots"))

# Add commas and spaces after each variable
output=$(echo "\$output" | awk 'BEGIN { FS=","; OFS="," } {print \$1, \$2, \$3, \
    ↪ $4, \$5}')

# Add header
header="Timestamp,Latitude,Longitude,MinSeaLevelPressure,MaxIntensity"

# Write data to output CSV file
echo "\$header" > "\$2"
echo "\$output" | sed 's/\t/,/g' >> "\$2"

echo "Done!"
```

2.1 Explanation of the script

- Line 12: This line starts by reading the contents of the input file (`$1`) using `cat`. It then uses `grep` to filter out lines containing the string 'UTC'. Next, `sed` is used to extract the timestamp information from the filtered lines, which is assumed to be in a specific format (e.g., '2024 UTC JAN 01').Finally, `uniq` is used to remove duplicate entries, and the resulting timestamp is stored in the variable `timestamp`.
- Line 13: This line uses `grep` with extended regular expressions (`-Eo`) to search for latitude values in the input file (`$1`). It extracts latitude values in the format of decimal degrees followed by 'N' (e.g., '34.052 N') and stores them in the variable `latitude`.
- Line 14: Similar to line 13, this line extracts longitude values from the input file in the format of decimal degrees followed by 'W' (e.g., '-118.244 W'). It uses `grep` with extended regular expressions to perform the extraction and stores the values in the variable `longitude`.
- Line 15: This line searches for minimum sea level pressure values in the input file (`$1`) using `grep` with extended regular expressions. It extracts numerical values followed by 'mb' (millibars) and stores them in the variable `minSeaLevel`.
- Line 16: Similarly, this line searches for knots information in the input file using `grep` with extended regular expressions. It extracts numerical values followed by 'knots' and stores them in the variable `knots`.
- Line 19: It combines the data fields that were extracted from the input file earlier. It takes the timestamp, latitude, longitude, minimum sea level pressure, and knots information and merges them into a single string. This string contains all the data fields without any specific separators between them. The paste command is used for this purpose, which essentially concatenates the data fields together. The resulting string is then stored in the variable output for further processing in the script.
- Line 22: Processes the concatenated data in the output variable, using awk to format it with commas between each data field; in this command, FS specifies the input field separator as a comma, and OFS specifies the output field separator also as a comma
- Line 25: Defines a header containing column names for the CSV file
- Line 28-31: Write the header to the output file specified as the second parameter, append the formatted data stored in the variable output to the same output file, and finally output "Done!" to indicate the completion of the script.

3 Storm Plots

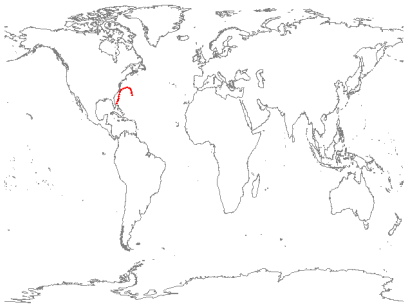

Figure 1: The output plot for al012020.kml file. Shows the map representation of the cyclone from May 16 to May 21.

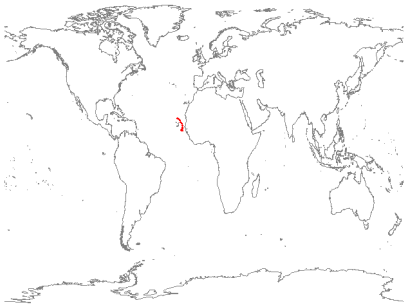

Figure 2: The output plot for al102020.kml file. Shows the map representation of the cyclone from July 29 to August 02.

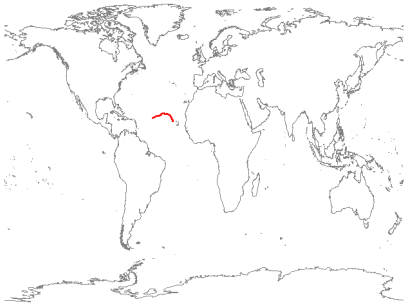

Figure 3: The output plot for al212020.kml file. Shows the map representation of the cyclone from September 14 to September 19.

4 Git usage

During the development process, a conflict arose when merging the changes made in the `python-addon` branch with the main working branch. This conflict occurred in the `python-plot-script.py` file, which had been edited in both branches.

To resolve the conflict, the following steps were taken:

1. Identified the conflicting sections within the `python-plot-script.py` file.
2. Manually reviewed the changes made in both branches to understand the conflicting edits.
3. Made necessary adjustments to the file to resolve the conflict, ensuring that the final version of the script incorporated the desired changes from both branches.
4. Used Git commands to complete the merge and resolve the conflict:
   - `git checkout main`: Switched to the main working branch.
   - `git merge python-addon`: Merged the changes from the `python-addon` branch into the main branch.
   - Resolved any merge conflicts within the `python-plot-script.py` file manually.
   - `git add python-plot-script.py`: Added the resolved file to the staging area.
   - `git commit -m "Resolved conflict in python-plot-script.py"`: Committed the changes with an appropriate message.
   - `git push origin main`: Pushed the changes to the remote repository.

The final version of the `python-plot-script.py` file, which incorporated the resolved conflict, was successfully pushed to the remote repository and is provided below: This resolved version of the script ensured that all necessary changes were retained and integrated seamlessly into the project.

4.1 python-plot-script.py Script

Listing 2: python-plot-script.py

```
import pandas as pd
import matplotlib.pyplot as plt
import os
import glob
import math
user_key= 27702

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()

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

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