ISS LIS and ECMWF Lightning Data Analysis

Khizer Zakir

SUMMER INTERNSHIP REPORT

Supervised by

Conrad Bielski, Gunter Zeug, & Jakob Lobensteiner

April- July 2023

Contents

1 Introduction

This report presents a comprehensive analysis of lightning data collected from two distinct sources: the ISS LIS sensor for real-time lightning observations and ECMWF forecasts for lightning predictions. The study highlights the potential and complementary nature of these data sources in understanding lightning activity. The analysis involved the utilization of GDAL, QGIS, R, and Python tools to read, clean, aggregate, and analyze the Near Real-Time (NRT) data from the ISS LIS sensor and the forecast data from ECMWF.

1.1 Analysis of ECMWF Forecast Data

The ECMWF forecasts data were processed using GDAL and QGIS, renowned tools for handling geospatial data. GDAL, in combination with QGIS, allowed for the efficient extraction and manipulation of forecast data, especially the most relevant parameters, such as "Litota" and "Cape" values enabling meaningful insights into lightning predictions. QGIS helped in visualizing the forecast data on maps, aiding in the identification of high-risk areas prone to lightning occurrences.

1.2 Analysis of ISS LIS Lightning Data

The ISS LIS lightning data was analyzed using a combination of R and Python, two versatile programming languages for data manipulation and analysis. Using Python, the ISS LIS lightning data in NetCDF format was efficiently read into memory, enabling seamless data handling. The data was then cleaned and preprocessed to eliminate any anomalies and inconsistencies. Converted ".nc" files into gridded GeoTIFF files, following the data recipe of [2] hosted on NASA - Data Recipes. Subsequently, R was employed to aggregate and summarize the cleaned lightning data. Temporal patterns of lightning occurrences were explored to identify any trends or seasonality in lightning activity.

For all the codes and explanation refer to Appendix B6

1.3 ISS LIS Lightning Data

The Lightning Imaging Sensor (LIS) is a space-based instrument designed to detect and study total lightning (cloud-to-cloud, intra-cloud, and cloud-to-ground lightning) and its distribution and variability over a large region of the Earth's surface. There were two LIS instruments built in the 1990s, one for the Tropical Rainfall Measurement Mission (TRMM) and a spare LIS, which remained stored for over 20 years. The TRMM LIS operated successfully from 1997 to 2015, while the spare LIS was placed on the International Space Station (ISS) in February 2017 and is still ongoing.

The LIS instrument uses a calibrated lightning sensor with a wide field-of-view expanded optics lens and a narrow-band filter to detect lightning at millisecond timing and storm-scale resolution. A high-speed charge-coupled device (CCD) detection array and a Real Time Event Processor (RTEP) help identify lightning flashes, even in the presence of bright sunlit clouds and the night light. The RTEP removes background signals to detect weak lightning, achieving up to a 90 % detection efficiency. The instrument records the time of lightning occurrences, measures the radiant energy, and determines the location of flashes within its field-of-view [1].

ISS lightning data:

- The ISS LIS instrument records the time of occurrence, radiant energy, and location of each lightning event
- Near-real time (NRT) data are available within two minutes of observation

2 Data Analysis

During my internship, I worked on lightning forecast information by ECMWF and ISS LIS lightning data hosted by GHRC. I utilized the GDAL library to process lightning forecast data for "Litota" and "Cape" values. Additionally, I employed Python and R to analyze and convert ISS LIS lightning data from NetCDF format to a dataframe data structure, enabling the production of dynamic web maps. Furthermore, I attempted to develop a RandomForest model to predict wildfires, incorporating temperature, lighting, fire warnings, and two other variables to enhance hazard information and predictability.

2.1 Lightning Parameters for the ECMWF Model

The FMI dataset has different datasets. We are interested in the Litota and Cape values. The Litota subset is made up of forecast lightning so there are up to 85 raster bands which are temporal. To extract these for further processing use the GDAL commands available in *Appendix A*6 given a netCDF file

`fcyyyymmddssss_ens_lightning.nc`

- cape: convective available potential energy (J/kg),
- litota: averaged total lightning flash density in the last 3 or 6 hours (1/km2/day)

b **Lightning parametrization**

In the version planned for operational implementation in IFS Cycle 45r1, the lightning parametrization does not discriminate between CG and IC flashes. It calculates total (i.e. CG+IC) flash density f_T (in flashes/km^2/day) as

$$f_T = \alpha\, Q_R \sqrt{CAPE}\, [\min(z_{base}, 1.8)]^2 \qquad (1)$$

where α is a tunable coefficient, currently set to 37.5 to match the annual global mean flash rate from the LIS/OTD climatology. The variable z_{base} is the convective cloud base height (in km). The term Q_R denotes a proxy for the charging rate resulting from the collisions between graupel particles and other types of hydrometeors inside the charging layer and is computed as

$$Q_R = \int_{z(0°C)}^{z(-25°C)} q_{graup}\,(q_{cond} + q_{snow})\rho(z)\,dz \qquad (2)$$

In Equation (2), q_{cond}, q_{graup} and q_{snow} denote the amount of convective cloud condensate, graupel and snow, respectively (in kg/kg), while $\rho(z)$ is the ambient air density (in kg/m^3) at altitude z. The amount of graupel and snow at each model level is diagnosed from the convective frozen precipitation flux P_f by writing

$$q_{graup} = \beta \frac{P_f}{\rho V_{graup}} \qquad (3)$$

$$q_{snow} = (1-\beta) \frac{P_f}{\rho V_{snow}} \qquad (4)$$

where β is set to 0.7 over land and 0.45 over sea, while constant fall speeds V_{graup} and V_{snow} for graupel and snow are assumed to be 3.0 and 0.5 m/s, respectively.

Figure 1: CAPE Equation Explanation

2.2 Near Real-Time Lightning Analysis and Web Mapping

During the course of this project, I utilized both R and Python to download NetCDF files from GHRC (Global Hydrology Resource Center) based on specific dates.

The first phase of the analysis involved data cleaning and processing to extract relevant information from the NetCDF files. By parsing the dates and times associated with each lightning flash, I transformed the raw data into a structured format suitable for further analysis. This enabled me to build dataframes that could be easily queried based on date and time. *for code refer to Appendix B6*.

In the subsequent stages of analysis, I improved the data processing pipeline to read values from the dataframes based on specified dates and times. This enhancement facilitated better filtering and analysis of near real-time lightning flashes for specific temporal intervals. The dataframes were instrumental in efficiently retrieving lightning data corresponding to user-defined periods, making the analysis more flexible and versatile. *for code refer to Appendix B 6*.

The derived dataframes provided the desired output, which was crucial for the successful implementation of web mapping. To visualize the near real-time lightning flashes on webmaps, my colleagues leveraged technologies such as pmtiles and MapLibre. These tools enabled seamless integration of lightning data into interactive and user-friendly webmaps.

3 Sample Outputs

Table 1: Table with date/time of lighting - ISS LIS data

Date	Time	Latitude	Longitude	Orbit Start	Orbit End
4/1/2023	12816	4.8999815	137.88545	954466106.4	954471679.4
4/1/2023	12816	4.9965777	137.90236	954466106.4	954471679.4
4/1/2023	12816	4.996413	137.90112	954466106.4	954471679.4
4/1/2023	12816	4.9977455	137.90723	954466106.4	954471679.4
4/1/2023	12816	5.839443	138.36586	954466106.4	954471679.4
4/1/2023	12816	26.752033	151.30055	954466106.4	954471679.4
4/1/2023	12816	26.748571	151.38591	954466106.4	954471679.4
4/1/2023	12816	39.039104	-88.07466	954466106.4	954471679.4

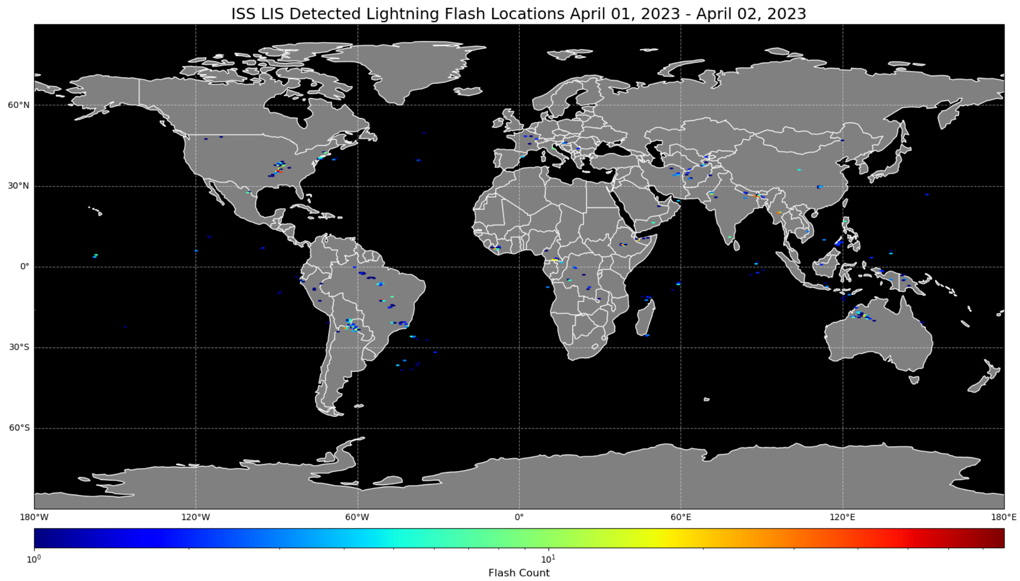

Figure 2: Lightning plot for a date in April

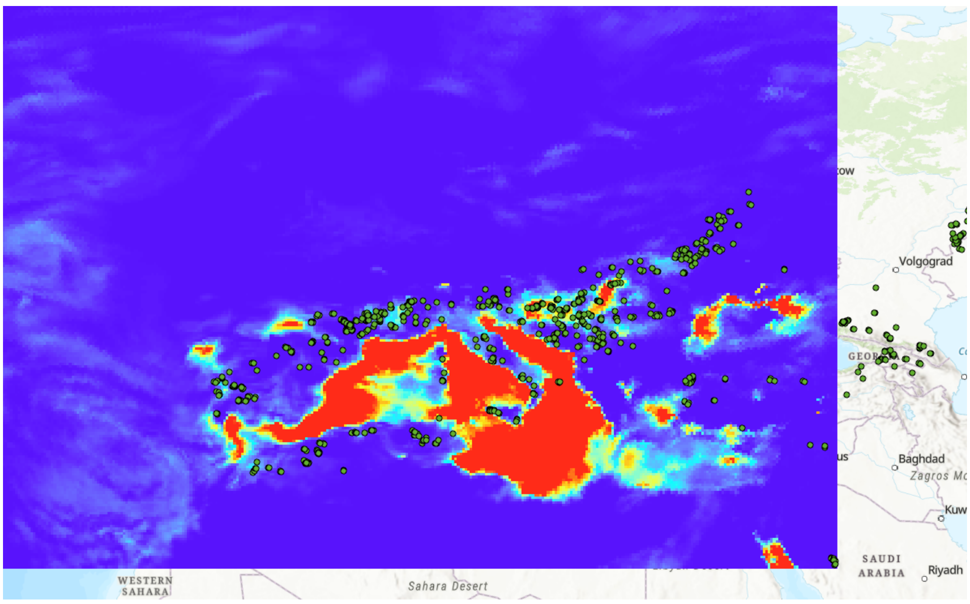

Figure 3: Litota values mapped in GIS

4 Risk Assessment Model & Future Potentials

4.1 Risk Assessment Model

There was an attempt to develop the risk assessment model that aims to analyze near real-time lightning data obtained from the ISS LIS sensor and ECMWF forecasts. The model combines data (lightning, fire alerts, wildfire occurences, forest cover, temperature, and cape values)and processing techniques using GDAL, QGIS, R, and Python to extract and clean the respective data, enabling the identification of lightning hotspots and trends over specific regions and time intervals.

By aggregating and summarizing the cleaned data, the model can assess temporal patterns of lightning occurrences and respective entities, allowing for improved hazard identification and preparedness. The dynamic webmaps created using pmtiles and MapLibre visualize the lightning activity and wildfire occurences, enhancing situational awareness for better risk management. *Please refer to the code in Appendix: C .0.3.*

4.2 Future Potentials

The data analysis and the sample risk assessment model developed in this project holds promising future potentials for wildfire hazard assessment and predictability. As data collection techniques and forecast models continue to advance, the model can be expanded to incorporate additional variables, such as temperature, fire warnings, and other environmental factors relevant to wildfire occurrence.

Furthermore, advancements in machine learning techniques, particularly random forest models, can be integrated into the risk assessment process. By utilizing historical lightning and wildfire data, the model can be trained to predict wildfire risk based on various factors, leading to more accurate and reliable predictions.

Additionally, the web mapping application can be enhanced to include real-time data updates and interactive features for users to customize risk assessments based on specific criteria and regions. Such improvements would empower emergency responders and policymakers to make informed decisions and implement timely mitigation strategies to reduce wildfire hazards.

5 Personal Experience and Growth

Working with lightning data for wildfire hazard assessment has been an immensely enriching and rewarding experience, encompassing various technical skills and tools across Python and R. From the outset, I was excited about the potential of this project to contribute to wildfire management and emergency preparedness. As I delved into the world of geospatial data analysis, I encountered numerous opportunities to enhance my data acquisition, manipulation, and analysis skills in both Python and R.

In the early stages of the project, I honed my Python programming skills to efficiently download NetCDF files from the Global Hydrology Resource Center (GHRC) based on specific dates. Utilizing Python's requests library, I successfully retrieved the required data, showcasing my proficiency in web scraping and data acquisition techniques.

As I transitioned to data analysis, R emerged as a powerful tool for processing and visualizing lightning data. Working with various R libraries, such as gdal, caret, and randomForest, allowed me to gain a deeper understanding of geospatial data processing techniques, machine learning algorithms, and model evaluation methods. I successfully created dataframes from NetCDF files, conducted exploratory data analysis, and trained a random forest model for wildfire risk prediction.

Throughout the project, I found myself continually researching and exploring new possibilities for utilizing lightning data for wildfire hazard assessment. This journey of continuous learning not only expanded my technical skills but also broadened my perspective on the potential applications of geospatial data in environmental monitoring and disaster management.

Moreover, the project involved handling NetCDF files, which was a novel experience for me in both Python and R. Learning to read, extract, and process data from these files demonstrated my adaptability and willingness to acquire new data manipulation skills across different programming languages.

Reflecting on this multifaceted journey, I am filled with gratitude for the opportunity to work on such a meaningful project. The exposure to geospatial data analysis, machine learning, and wildfire management has not only enhanced my technical proficiency but also deepened my passion for making a positive impact through data-driven solutions.

As I move forward, I am eager to continue my exploration of hazard assessment and predictability models for wildfires, leveraging the diverse knowledge and experiences gained from working with lightning data in Python and R. This endeavor has reinforced my commitment to contributing to environmental conservation and community safety through innovative and data-driven approaches.

6 Appendix

Appendix A: Converting / extracting NetCDF lightning data from FMI

To inspect the contents of the NetCDF file `fc202304270000_ens_lightning.nc`, you can use the `gdalinfo` command:

```
gdalinfo fcyyyymmdd0000_ens_lightning.nc
```

This command will provide information about the NetCDF file, including its metadata and structure.

To extract the "litota" variable from the NetCDF file and save it as a GeoTIFF file, you can use the `gdal_translate` command:

```
gdal_translate -of GTIFF -b 1

NETCDF:"fcyyyymmdd00000_ens_lightning.nc":litota

fcyyyymmdd0000_ens_lightning_litota_b1.tif
```

In this command, the `-of GTIFF` flag specifies the output format as GeoTIFF, and the `-b 1` flag indicates to extract the first band, which corresponds to the "litota" variable. The output will be saved as `fcyyyymmdd0000_ens_lightning_litota_b1.tif`.

Appendix B: R and python Code for Data Analysis

To begin the data processing, we need to set up our Python environment with the necessary libraries. Make sure to install the following libraries using `pip`:

- `os`
- `pandas` (as `pd`)
- `matplotlib.pyplot` (as `plt`)
- `xarray` (as `xr`)

.0.1 Save All the NetCDF Files According to Their Dates

Next, we will download and save all the NetCDF files from GHRC based on their respective dates. We can use the `requests` library to download the files.

.0.2 Read All of the NetCDF Files, Store the Output in Respective Subfolders with CSV Files and PNG Plots

Now that we have downloaded the NetCDF files, we can proceed with reading them and performing data processing. We will create subfolders for each date and store the output CSV files and PNG plots in their respective subfolders.

Python Code

```
#loading libraries

import sys
import os
import glob
from netCDF4 import Dataset, num2date
import numpy as np
import csv
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.ticker as mticker
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER,
    ↪ LATITUDE_FORMATTER
import re
import datetime

#Initial file path. It can be changed by passing a different
    ↪ path as an argument
#to the main() function
file_path = 'E:/ERASMUS/Internship/Data/0104_2023'
os.path.exists(file_path)

def main(file_path):

    #Define the directory of the files
```

```python
    dataDir = os.path.join(file_path, '')

    #Identify all the ISS LIS NetCDF files in the directory and
        ↪ their paths
    raw_files = glob.glob(os.path.join(dataDir, 'ISS_LIS_*.nc'))
    files = [os.path.normpath(i) for i in raw_files]


    #Extract the dates for the files
    #Create empty lists to hold the orbit start and end times
    orbit_start = []
    orbit_end = []

    #Loop through the NetCDF files and for each file, extract
        ↪ the start and end time of the ISS LIS
    #orbit, adding them to the respective empty list (
        ↪ orbit_start and orbit_end)

    for i in files:
        datafile = Dataset(i,'r')

        start_value = datafile.variables['
            ↪ orbit_summary_TAI93_start'][:].data.tolist()
        start_value_units = datafile.variables['
            ↪ orbit_summary_TAI93_start']
        end_value = datafile.variables['orbit_summary_TAI93_end'
            ↪ ][:].data.tolist()
        end_value_units = datafile.variables['
            ↪ orbit_summary_TAI93_end']
        orbit_start.append(start_value)
        orbit_end.append(end_value)

    #From the start and end times, calculate the minimum and
        ↪ maximum date of the files
    start_dates = num2date(orbit_start[:], start_value_units.
        ↪ units)
    stop_dates = num2date(orbit_end[:], end_value_units.units)

```

```
    begin_date_value = min(start_dates)
    end_date_value = max(stop_dates)

    #Create text and numerical dates to use in file names and plot title
    begin_date = begin_date_value.strftime("%B %d, %Y")
    end_date = end_date_value.strftime("%B %d, %Y")
    begin_int = begin_date_value.strftime("%Y%m%d")
    end_int = end_date_value.strftime("%Y%m%d")

    #Create CSV file and destination
    csvfile = os.path.join(dataDir, 'isslis_flashloc_'+ begin_date + '_' + end_date +'.csv')

    #Extract lightning flash locations
    #Create empty arrays to populate lightning flash location coordinates
    flash_lat = np.array([]) #latitude
    flash_lon = np.array([]) #longitude
    dates = np.array([])

    #Loop through list of NetCDF files and for each file, extract the lightning flash latidude
    #and longitude, adding them to the respective empty array (flash_lat and flash_lon)
    for i in files:
        datafile = Dataset(i)

        flash_lat = np.concatenate([flash_lat,datafile.variables['lightning_flash_lat'][:]]) #add to array
        flash_lon = np.concatenate([flash_lon,datafile.variables['lightning_flash_lon'][:]]) #add to array

    #Create CSV files of values from the populated flash_lat/lon arrays
    with open(csvfile, 'w', newline='') as myfile:
        writer = csv.writer(myfile)

```

```
        writer.writerows(zip(["Date"],["flash_lat"], ["flash_lon"])) #Define headers in row (zip creates columns)
         # Write data rows with common date, latitude, and longitude values
        common_date = begin_date_value.strftime("%m-%d")  # Format common date as MM_DD
        writer.writerows(zip([common_date]*len(flash_lat), flash_lat, flash_lon))

    #Create plot of lightning flash location heat map
    plt.figure(figsize=((20,20))) #Set plot dimensions
    map = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0))
    gl = map.gridlines(crs=ccrs.PlateCarree(central_longitude=0.0), draw_labels=True, linewidth=0.8, alpha=0.5, color='white', linestyle='--')
    lightning = map.hexbin(flash_lon, flash_lat, gridsize=300, bins='log',cmap='jet', mincnt=1 ,zorder=10) #Bin flash counts into hexbins using a gridsize of your choice

    #Draw geographic boundaries and meridians/parallels
    map.set_extent([-180, 180,-90, 90])
    map.coastlines(color='white')
    map.add_feature(cfeature.LAND, facecolor='gray')
    map.add_feature(cfeature.BORDERS, edgecolor='white')
    map.add_feature(cfeature.OCEAN, facecolor='black')
    gl.ylocator = mticker.FixedLocator([-90, -60, -30, 0 ,30, 60, 90])
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    gl.xlabels_top=False
    gl.ylabels_right=False

    #Create colorbar
    cbar = plt.colorbar(lightning, orientation='horizontal', pad=0.02, aspect=50)
    cbar.set_label('Flash Count', fontsize=12) #Remember to change label
```

```python

    #Create plot title based on file dates
    if begin_date != end_date:
        plot_title = 'ISS LIS Detected Lightning Flash Locations ' + begin_date + ' - ' + end_date
    else:
        plot_title = 'ISS LIS Detected Lightning Flash Locations ' + end_date

    plt.title(plot_title, fontsize = 18)

    #Save the plot as an image
    plt.savefig(os.path.join(dataDir, 'isslis_flashloc_'+ begin_int + '_' + end_int +'_plot.png'), bbox_inches='tight')
```

R code

Load the required libraries:

```r
library(ncdf4)
library(ggplot2)
library(dplyr)
```

Define the directory of the files:

```r
dataDir <- 'E:/ERASMUS/Internship/Data/02042023'
```

Define the 'main' function:

```r
main <- function(file_path) {

  # Identify all the ISS LIS NetCDF files in the directory and their pa
  raw_files <- list.files(path = dataDir, pattern = "ISS_LIS_.*\\.nc$",

  # Extract the dates for the files
  # Create empty lists to hold the orbit start and end times
  orbit_start <- vector()
  orbit_end <- vector()

  # Loop through the NetCDF files and for each file, extract the start
  # orbit, adding them to the respective empty list (orbit_start and or
```

```
for (i in raw_files) {
  datafile <- nc_open(i)

  start_value <-  ncvar_get(datafile, "orbit_summary_TAI93_start")
  end_value <- ncvar_get(datafile, "orbit_summary_TAI93_end")

  orbit_start <- c(orbit_start, start_value)
  orbit_end <- c(orbit_end, end_value)

  nc_close(datafile)
}

# From the start and end times, calculate the minimum and maximum dat
start_dates <- as.POSIXct(orbit_start, origin = "1993-01-01")
stop_dates <- as.POSIXct(orbit_end, origin = "1993-01-01")

begin_date_value <- min(start_dates)
end_date_value <- max(stop_dates)

# Create text and numerical dates to use in file names and plot title
begin_date <- format(begin_date_value, "%B␣%d,␣%Y")
end_date <- format(end_date_value, "%B␣%d,␣%Y")
begin_int <- format(begin_date_value, "%Y%m%d")
end_int <- format(end_date_value, "%Y%m%d")

# Create CSV file and destination
csvfile <- file.path(dataDir, paste0("isslis_flashloc_", begin_int, "

# Extract lightning flash locations
# Create empty data frames to populate lightning flash location coord
flash_data <- data.frame(flash_lat = numeric(), flash_lon = numeric()

# Loop through list of NetCDF files and for each file, extract the li
# and longitude, adding them to the respective data frame (flash_data
for (i in raw_files) {
```

```
    datafile <- nc_open(i)

    flash_lat <- ncvar_get(datafile, "lightning_flash_lat")
    flash_lon <- ncvar_get(datafile, "lightning_flash_lon")

    flash_data <- bind_rows(flash_data, data.frame(flash_lat, flash_lon

    nc_close(datafile)
  }

  # Create CSV file of values from the populated flash_data data frame
  write.csv(flash_data, csvfile, row.names = FALSE)

  # Create plot of lightning flash location heat map
  ggplot(flash_data, aes(x = flash_lon, y = flash_lat)) +
    geom_bin2d(bins = 300) +
    scale_fill_gradient(name = "Flash␣Count", guide = "colorbar") +
    coord_cartesian(xlim = c(-180, 180), ylim = c(-90, 90)) +
    labs(title = paste("ISS␣LIS␣Detected␣Lightning␣Flash␣Locations", be
    theme_bw()
}
```

Call the 'main' function with the specified 'dataDir':

```
main(dataDir)
```

.0.3 Read All the NetCDF Files Together and Create a DataFrame with Date and Time

In this step, we will modify the code to read all the NetCDF files together and create a single DataFrame that includes both the date and time information. This will enable us to have a comprehensive view of the data.

Appendix C: R Code for Sample Assessment Model

Random Forest Model in R

Load the required libraries:

```
library(randomForest)
library(random)
library(caret)
```

Set the seed for reproducibility:

```
set.seed(123)
```

Define the number of observations and generate random data:

```
num_observations <- 100

CAPE <- runif(num_observations, min = 100, max = 300)
Lightning <- sample(c("NO", "YES"), num_observations, replace = TRUE)
Temperature <- runif(num_observations, min = 25, max = 35)
Forest <- sample(c("NO", "YES"), num_observations, replace = TRUE)
AlertCount <- runif(num_observations, min = 5, max = 20)
Wildfire <- sample(c("NO", "YES"), num_observations, replace = TRUE)
```

Create the data frame:

```
data <- data.frame(
  CAPE = CAPE,
  Lightning = Lightning,
  Temperature = Temperature,
  Forest = Forest,
  AlertCount = AlertCount,
  Wildfire = Wildfire
)

print(head(data))
```

Convert categorical variables to factors:

```
data$Lightning <- as.factor(data$Lightning)
data$Forest <- as.factor(data$Forest)
data$Wildfire <- as.factor(data$Wildfire)
```

Split the data into training and testing sets:

```
set.seed(123)  # For reproducibility
train_indices <- sample(1:nrow(data), 0.7 * nrow(data))
train_data <- data[train_indices, ]
```

```
test_data <- data[-train_indices, ]
```

Train the Random Forest model:

```
model <- randomForest(Wildfire ~ ., data = train_data, ntree = 100)
```

Make predictions on the test data:

```
predictions <- predict(model, newdata = test_data)
```

Create a confusion matrix:

```
confusion <- confusionMatrix(predictions, test_data$Wildfire)
print(confusion)
```

References

[1] NASA. *Lightning imaging sensor (LIS) - sensor - observations*. URL: https://ghrc.nsstc.nasa.gov/lightning/overview_lis_instrument.html.

[2] A. Weigel. *Using ArcGIS to Convert LIS Very High Resolution Gridded Lightning Climatology NetCDF Data to GeoTIFF Format*. URL: https://ghrc.nsstc.nasa.gov/home/data-recipes/using-arcgis-convert-lis-very-high-resolution-gridded-lightning-climatology-netcdf-data.