Airspace in Conflict: An Analysis of Historical Flight Patterns and the Impact of War in Ukraine and Its Surrounding Regions

Student Name:	Mary Dannoun
Student ID:	2249082
Module Code:	CS57811
Module Title:	Distributed Data Analysis
Academic year:	2023/2024

1. Data Description and Research Question

1.1 Data Description

This study utilizes flight data sourced (Badami and Dannoun: Source Code, 2023; Appendix I) from two distinct databases of more than 200 million rows combined to analyze the impact of ongoing conflicts in Ukraine on regional flight activities. After querying selected countries, we were left with over 3.6 million rows of data. The data spans from January 1, 2016, to the present, providing a comprehensive view of flight patterns before, during, and after the onset of the conflicts.

Database 1: OpenSky Network (January 1, 2016 - July 23, 2023)

The data was extracted from the OpenSky Network (Schäfer, 2016), which employs Hadoop Impala technology with a cluster of 21,000 nodes. The `flights_data4` (Appendix G, Figure A and B) table was utilized, capturing critical information such as aircraft ID (icao24) (Appendix G, Figure C) , timestamps for first and last signals received (firstseen, lastseen), estimated departure and arrival airports (estdepartureairport, estarrivalairport), and the day of the flight. The data is based on Mode S/ADS-B communication, ensuring high reliability in tracking aircraft movements.

Database 2: Aviation Edge (Post-July 23, 2023)

To supplement recent data, the Aviation Edge database (Aviation Edge, 2023) was employed. This API-based source provides flight information based on specific parameters including airline, airport codes, flight type, and date range. Due to API constraints such as a maximum date range of ten days per query, data was meticulously collected by iterating over relevant airports for arrivals and departures.

Integration and Scope:

The data from both sources were harmonized to create a unified dataset. The comprehensive dataset covers global flights from 2016-01-01 to 2023-07-23 from Database 1 and focused flight data from selected countries affected by the conflicts post-July 23, 2023, from Database 2. This integrated approach provides a broad, yet detailed view of the aviation landscape influenced by the geopolitical shifts in Ukraine.

1.2 Research Question

The research question we aim to answer is the following:

"What are the historical patterns of flight departures and arrivals in Ukraine and its surrounding region, and how has the conflict within Ukraine affected these patterns?"

2. Data preparation and Cleaning

2.1 Data Preparation

For Database 1 (OpenSky Network):

The database, powered by Cloudera Impala (Hadoop File Formats Support, 2023), is queried through SSH in chunks, with the output redirected into a log file. Subsequently, GNU utilities are employed for data transformation; specifically, the `script` command captures the output, while `tail` and `head` trim unnecessary content. Following this, the `fart` utility replaces vertical bars with commas to create a CSV structure and removes any superfluous characters. Finally, the cleaned data is loaded into a Pandas DataFrame for deduplication, ensuring the dataset's temporal consistency, particularly within the non-null 'day' column.

For Database 2 (Aviation Edge):

The API delivers structured JSON with comprehensive flight details, where interaction entails crafting GET requests that retrieve data on departures and arrivals, tailored by parameters like airport codes and dates. Subsequently, time-related data is deduced from the JSON responses, with a clear preference hierarchy to ensure accuracy. Next, essential details such as ICAO codes are extracted and integrated into a Pandas DataFrame, harmonized to align with prior datasets. Consequently, the assimilated data is formatted as CSV, establishing a uniform foundation that facilitates the analysis of flight patterns across the designated time span, thereby setting the stage for thorough data processing and subsequent analytical endeavors.

2.2 Data Cleaning

To ensure dataset quality and reliability for analysis, data cleaning was methodically conducted. Initially, missing values were addressed by selectively preserving rows with either the departure or arrival airport information absent, recognizing the incomplete nature of ADS-B communication data. Conversely, rows lacking both were completely removed, as they contribute little to the study's regional flight pattern focus. Additionally, to maintain data integrity, any duplicates identified were eliminated, mitigating potential biases. These steps have refined the dataset, rendering it both reflective of actual flight activity and well-suited for analyzing the influence of regional conflicts on flight patterns.

2.3 Further Preparation for Exploratory Data Analysis

To enrich the flight data with geographical and temporal context, the "Airports" dataset (OurAiports, 2023) was utilized. Initially, flight data from both primary databases and the "Airports" data were loaded into Pandas DataFrames. Subsequently, these datasets were concatenated to form a comprehensive dataset encapsulating flight activities. Focusing on geographical context, the "Airports" dataset was filtered to include only relevant airports, ensuring they had valid identifiers. This refined the flight data to include flights connected to the chosen airports. For temporal context, a "time" column was created by transforming timestamps into datetime objects. Finally, the flight data was enriched by joining it with the "Airports" data, which infused additional attributes such as country names and coordinates into each flight record. This prepared dataset is now primed for exploratory analysis and further research inquiry.

3. Exploratory Data Analysis

3.1 Flight Count Visualization

• Analysis of the Overall Trend of Flight Count

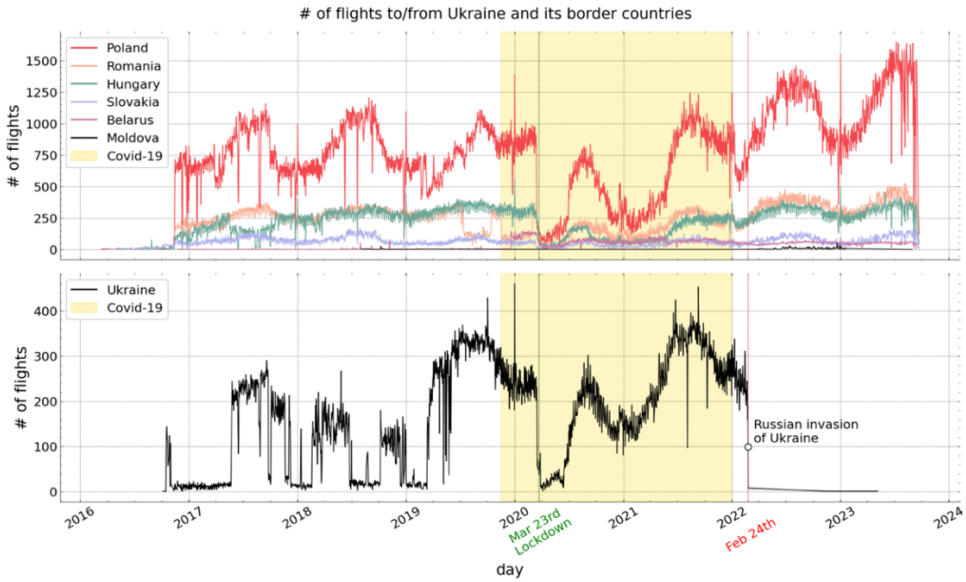


Figure 1: Flight Count Time Series

From 2016 to early 2024, the time series graph narrates the aviation trends in Ukraine and its neighbors. Poland's skies, bustling until February 2022, contrast with the less busy airspace of Belarus and Moldova. Ukraine's flight activity, variable and robust, plunges abruptly with the Russian invasion in February 2022, starkly illustrating the impact of geopolitical unrest on air travel. Additional irregularities in flight patterns, including those around the March 23rd Covid lockdown, suggest other underlying influences, setting the stage for further analysis.

• **Analysis of Flight Count Fluctuation**

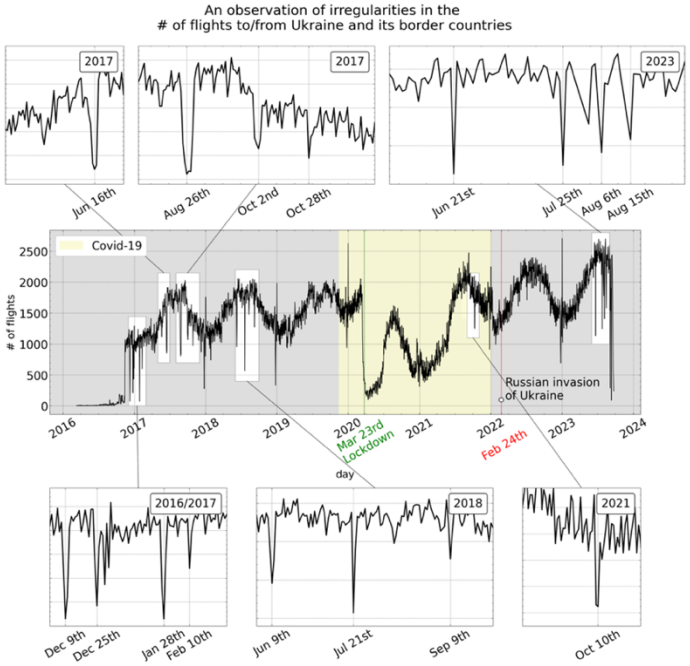


Figure 2: Zoomed View of Flight Count Fluctuation

The time series graph distinctly showcases seasonal flight patterns each year, with notable peaks during holiday seasons, particularly around Christmas and New Year. The graph also reflects the significant impact of the COVID-19 pandemic, starting in early 2020, where a steep decline in flights aligns with global lockdowns. Additionally, the Russian invasion of Ukraine in February 2022 is marked by a red line, indicating a notable decrease in flight numbers due to airspace closures and conflict. Furthermore, smaller inset graphs highlight specific dates across various years that show uncharacteristic spikes or drops in flights, suggesting irregularities that may require deeper investigation to uncover underlying causes.

• **Analysis of Flight Patterns Around Christmas**

The graph shows that flight activity dips on December 25th each year, likely due to the Christmas holiday. This decrease is followed by a varied recovery toward New Year's Eve, but instead of continuing to rise, flight numbers fall again on December 31st.

By January 1st, there's generally a sharp increase, indicating a resumption of travel after New Year celebrations. The pattern is consistent, though the scale of fluctuation changes notably during the 2019/2020 season, reflecting the impact of the COVID-19 pandemic.

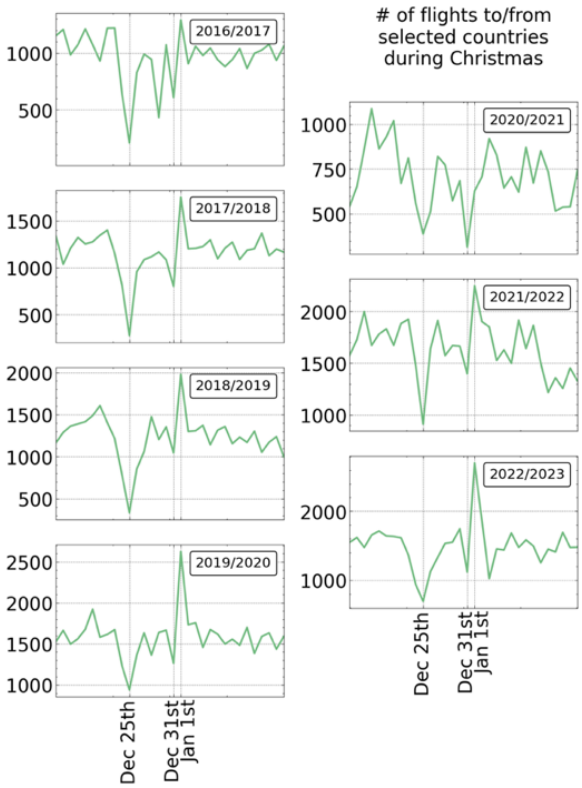
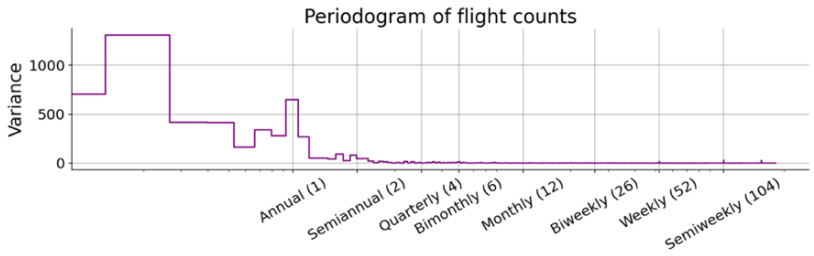


Figure 3: Zoomed View of Flight Count during Christmas

• Analysis of Seasonality of Flight Count Using Fourier Transformations



The periodogram reveals a clear annual cycle in flight volume, highlighting a robust yearly pattern, complemented by notable semiannual fluctuations. Yet, as the analysis shifts to finer scales—from monthly to weekly—the influence on flight numbers diminishes, underscoring those short-term variations have a lesser impact.

Figure 4: Decomposed Frequencies of Flight Count in the Fourier Space

• Analysis of Flight Density of Each Country with respect to Time

The heatmap illustrates monthly flight traffic between Ukraine and various European countries from November 2021 to October 2022, with higher flight counts depicted in darker shades.

Countries like Germany, Poland, and Turkey consistently exhibit intense traffic, possibly reflecting seasonal trends or the influence of events like the war in Ukraine as people fled the conflict.

Notably, coinciding with the onset of the Russian invasion, a stark cessation of flights from and to Ukraine is observed, underlining the conflict's immediate and profound effect on the country's air travel connectivity.

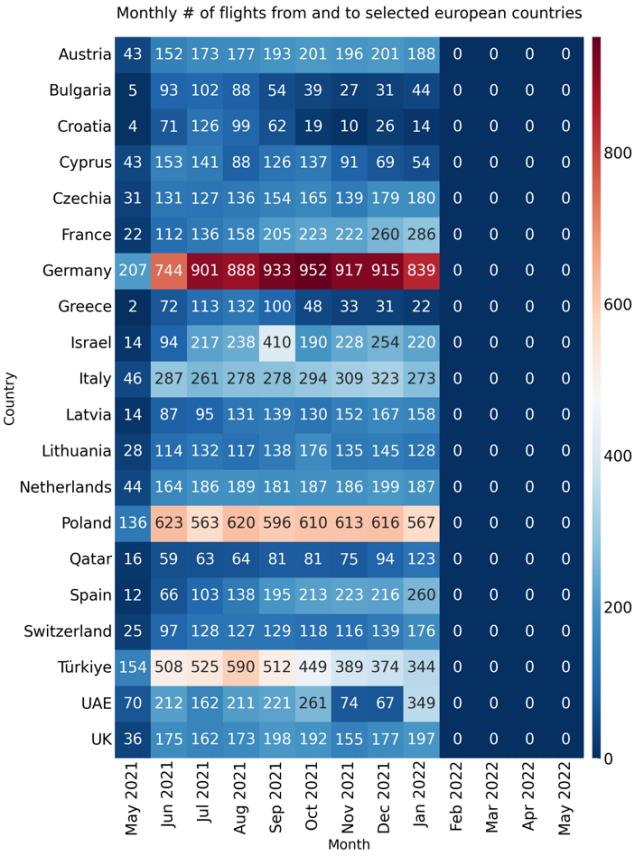


Figure 5: Flight Count Density Per Country Over Time

• Comparison of Flight Path Density Before and After the Russian Invasion of Ukraine

The flight route map vividly contrasts Ukraine's flight patterns just before and immediately following the war's onset. On February 23, 2022, the day before the conflict, a dense network of flight paths signifies a thriving, well-connected airspace. However, by February 24, 2022, these paths over Ukraine vanish, reflecting an abrupt halt in air traffic due to the war and resulting airspace closures. While neighboring countries maintain flight activities, their routes are significantly altered to circumvent Ukrainian airspace, highlighting the conflict's profound regional impact.

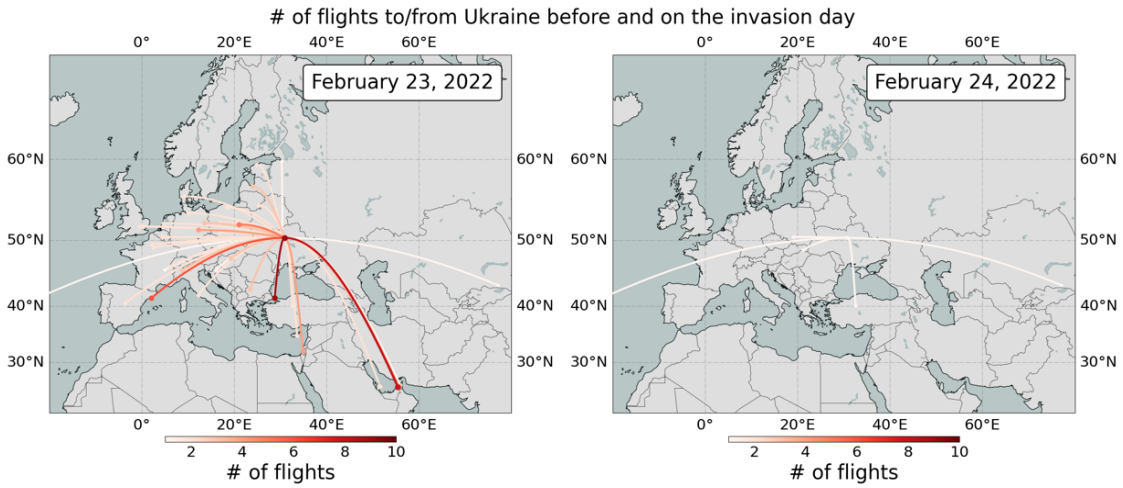


Figure 6: Before and After Comparison of Ukraine Flight Count on Invasion Day

3.2 Unsupervised Learning Method Implementation

• Airport Traffic Patterns Using K-Means Clustering:

To understand air traffic management, capacity planning, and regional economic impacts, we focused on analyzing air traffic distribution at airports. The methodology involves counting departures and arrivals at each airport, and applying K-Means clustering (CS5706, Lecture 5) to categorize airports with similar traffic levels (k = 5). The final step visualizes these clusters through a heatmap, marking each airport's location based on traffic intensity and cluster size, offering a clear representation of traffic distribution across Europe.

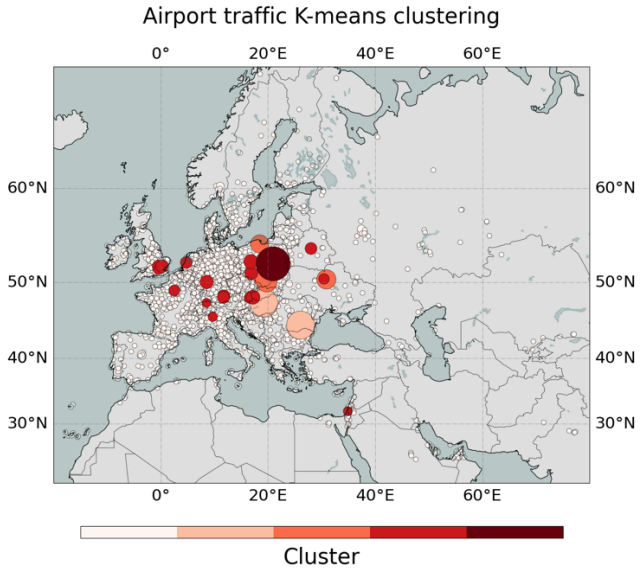


Figure 7: Clustering Results of Ukrainian Airports

• Aircraft Utilization Patterns Using Agglomerative Clustering:

In order to gain insights into operations research, logistics, and aviation management, we looked into aircraft utilization patterns to uncover operational efficiencies, aircraft usage, and scheduling patterns. The process involves calculating key metrics like total flight time and average flight duration for each aircraft, and performing Agglomerative Clustering (CS5706, Lecture 4) to group aircraft with similar utilization patterns. The analysis of cluster characteristics provides a deep understanding of various aircraft utilization patterns.

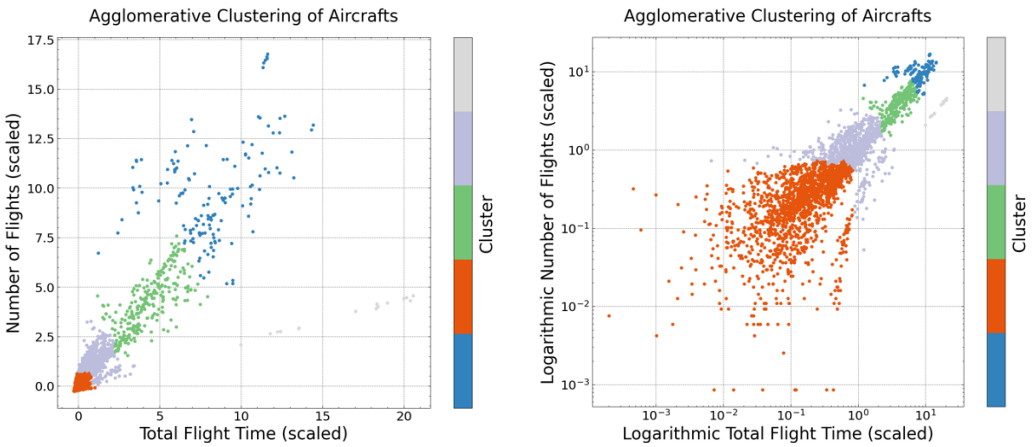


Figure 8: Region Aircraft Clustered by Flight Duration and Number of Flights (Left: Normal Scale, Right: Logarithmic Scale)

The two scatter plots visualize the Agglomerative clustering of aircraft by flight frequency and duration. The first uses a linear scale showing distinct groupings based on activity levels, while the second's logarithmic scale reveals a wider range of utilization patterns with color gradients indicating different clusters, which reflects the diversity in aircraft operations.

4. Machine Learning Prediction

In the Machine Learning section, the focus is on univariate time series forecasting to predict flight numbers per country over time in an attempt to study the effect of conflict in Ukraine on flight patterns. Advanced models like Extreme Gradient Boosted Decision Trees, LSTM, and GRU will be employed. The data will undergo preprocessing; features such as year, month, and day of the week will be extracted from timestamps, and sine and cosine functions will model seasonality, guided by Fourier transform analysis. External features such as 'is_holiday' from the 'Workalendar' dataset (Workalendar, 2023), and 'lockdown' from the 'Lockdowndates' dataset (Seanyboi, 2022). Non-ordinal categories will be one-hot encoded, and data normalization will be achieved using StandardScaler (Z-score).

In forecasting time series, decision trees are chosen for their strength in combining multiple models to boost prediction accuracy, while neural networks are used for their ability to identify complex patterns through space separation. Adhering to the course's teaching approach, we treated each time point as independent, omitting lag features that would normally account for time-based correlations in the data.

Algorithm 1: Decision Tree
Implemented Model: XGBRegressor

An XGBRegressor (CS5706, Lecture 7; Kumar et al, 2023), part of the XGBoost framework (XGBoost Documentation, 2022), is recognized for its ability to construct a robust predictive model by integrating outputs from simpler models like decision trees (Wohlwend, 2023). Launched in 2014, it has become popular for efficiently detecting patterns and relationships. When applied to time series forecasting, the XGBRegressor capitalizes on its pattern recognition prowess, utilizing a comprehensive array of temporal features to grasp historical trends. However, a notable limitation is its inability to extrapolate beyond the training data's range, a constraint inherent to tree-based models, which may affect its predictive performance in certain time series contexts (Wohlwend, 2023)..

```
model = XGBRegressor()
```

Code Block 1: Definition of XGBRegressor Model

Algorithm 2: Recurrent Neural Network (RNN)

Model 1 & 2: LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit)

Recurrent neural networks (RNNs) (CS5706, Lecture 6; Han et al, 2019) struggle with retaining information over long sequences, which is known as short-term memory. To address this, LSTM (Long Short-Term Memory) (Han et al, 2019) and GRU (Gated Recurrent Unit) (Han et al, 2019) models were developed, featuring gating mechanisms to

better manage information across time steps. There's no definitive choice between LSTM and GRU; it varies based on specific use cases. Using Keras (Kovelamudi et al, 2023), an LSTM model with a single layer of 700 units was implemented, incorporating a dropout of 10% to prevent overfitting.

```
model = Sequential([
    LSTM(units = 700),
    Dropout(0.1),
    Dense(units = 1)
], name = 'LSTM')

model.compile(optimizer = Adam(), loss = MeanSquaredError())
fit_kwargs = {'epochs': 30, 'batch_size': 32, 'verbose': 0}
```

Code Block 2: Definition of Single-Layer LSTM Model

```
model = Sequential([
    GRU(units = 700),
    Dropout(0.1),
    Dense(units = 1)
], name = 'GRU')

model.compile(optimizer = Adam(), loss = MeanSquaredError())
fit_kwargs = {'epochs': 30, 'batch_size': 32, 'verbose': 0}
```

Code Block 3: Definition of Single-Layer GRU Model

The model was optimized using the Adam optimizer (Keras Documentation, 2023), which adapts learning rates based on the exponential moving average of the gradients, and the performance was measured using mean squared error.

Model 3: A combination of LSTM and GRU

The LSTM and GRU models are combined to create a hybrid model that balances the potential underperformance of LSTM with the tendency of GRU to overpredict.

```
model = Sequential([
    LSTM(units = 700, return_sequences = True),
    Dropout(0.1),
    GRU(units = 700),
    Dropout(0.1),
    Dense(units = 1)
], name = 'LSTM_GRU')

model.compile(optimizer = Adam(), loss = MeanSquaredError())
fit_kwargs = {'epochs': 30, 'batch_size': 32, 'verbose': 0}
```

Code Block 4: Definition of Two-Layer LSTM>GRU Model

To streamline the time series forecasting process, the author of this report and their colleague developed a Python library with an API which was published on Github (Badami and Dannoun, 2023). This library manages the entire workflow from preprocessing to deployment, including feature engineering and model evaluation, but it requires users to provide their own model through the API.

5. High-Performance Computing Implementation

After scraping and cleaning our data in chunks using bash, our final dataset consists of around 200 million flight data for which we spawned a Hadoop cluster on a cloud server with four nodes with an overall storage of 80 GB, 16 CPUs and 32 GB of RAM (Appendix A and B).

Technique 1: Apache Hive

Apache Hive (LanguageManual ORC - Apache Hive, 2018), a data warehouse infrastructure built on top of Hadoop, is utilized for managing the extensive dataset in hand. Hive's big data processing capabilities enable efficient data querying, playing a crucial role in handling the large-scale data storage and management needs of the study (Appendix C, D and F).

Technique 2: Apache Impala:

Complementing Hive, Apache Impala provides a fast, interactive SQL-based query engine written in C for Hadoop (Apache Impala Documentation, 2023). This technology is essential for rapid data analysis, offering an interactive experience. Impala is used alongside Hive for data preparation tasks such as filtering selected countries and joining with the 'airports' table, with a focus on comparing the performance of HiveQL and Impala's SQL query engine (ProjectPro, 2023).

Hyperparameter Fine Tuning:

To address the computational challenges of grid search in hyperparameter tuning, a custom implementation of GridSearch is developed using Python's **multithreading functionality**. This approach significantly optimizes the efficiency of the machine learning process, ensuring that hyperparameter fine-tuning is both effective and time-efficient, even with the large-scale dataset involved in the study (Appendix E).

6. Performance Evaluation & Comparison of Methods

6.1 Machine Learning Evaluation & Comparison

Decision Tree Algorithm:

• XGBRegressor Results

The XGBRegressor model shows promising results in predicting flight counts. The model closely follows the actual data trends, with a scaled training score of 0.003651 indicating a strong fit. However, the scaled test score of 0.173791 suggests room for improvement in generalizing to new data. The visual suggests the model captures the overall pattern embodying the orange line, albeit with some ameliorated divergence.

	XGBRegressor
Scaled training score	0.003651
Scaled test score	0.173791

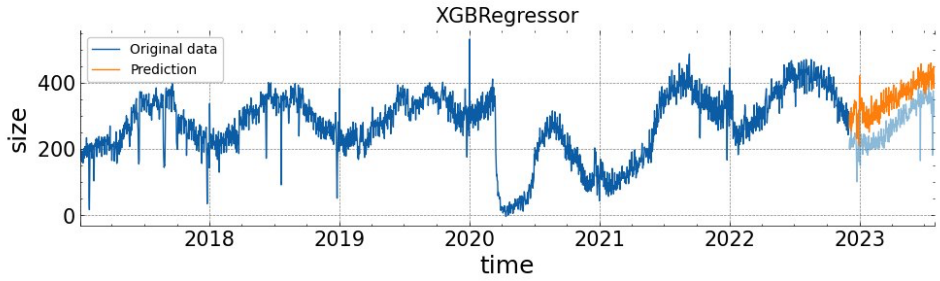


Figure 9: Evaluation Results of XGBRegressor Over the Validation Set

RNN Algorithm:

• LSTM, GRU and LSTM + GRU Results

A grid search on Neural Networks, using multithreading, evaluated 80 models across 5 data splits, taking 2337 seconds to find the best model. The optimal setup among all RNN models (LSTM, GRU, and LSTM + GRU) used two LSTM layers with 100 and 60 neurons, a batch size of 128, and 100 epochs, showing the highest average test score and strong robustness. The process benefited from TensorFlow's GPU acceleration on a MacBook Pro M1 silicon (Appendix E).

```
{
    layers: [LSTM, GRU]
    number_of_layers: [1, 2]
    layer_0_neurons: [100, 150]
    layer_1_neurons: [60, 100]
    batch_size: [32, 128]
    epochs: [100, 200]
}
```

Code Block 5: Grid Parameters

	LSTM
Scaled training score	0.003376
Scaled test score	0.131215

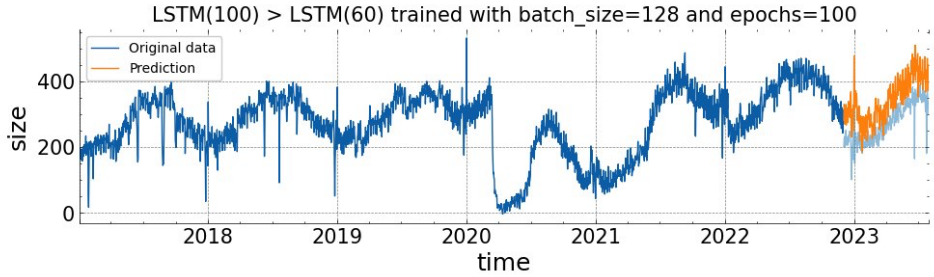


Figure 10: Evaluation Results of LSTM(100)>LSTM(60) Model

The LSTM100>LSTM60 model shows a close fit to the flight data, with a training score of 0.003376 and a test score of 0.131215. The predictions closely track the actual data's trends with better test results than the XGBResressor model.

• Performance Evaluation Sorted by Mean Test Score

The LSTM100>LSTM60 model, with a batch size of 128 and 100 epochs, demonstrates the best mean test score among the models, indicating better accuracy.

		Split	Mean			Standard deviation	
		variable	Training score	Test score	Execution time (s)	Training score	Test score
Model	Batch size	Epochs					
LSTM100>LSTM60	**128**	**100**	0.002296	0.271617	14.253503	0.000551	0.239281
LSTM150>LSTM100	**128**	**100**	0.002150	0.276615	17.031967	0.000540	0.251289
LSTM100>LSTM100	**128**	**100**	0.002572	0.280945	14.815283	0.000663	0.254216
GRU100	**128**	**100**	0.005093	0.282721	8.554641	0.001672	0.226439
LSTM150>GRU100	**128**	**100**	0.002420	0.296087	16.322591	0.000249	0.299474

Figure 11: Top 5 Models in the Grid Search Based on Mean Test Score

• Robustness Evaluation Sorted by STD Test Score

The LSTM100>LSTM60 model shows robust performance with the second-lowest standard deviation in test scores (0.239281), indicating its predictions are consistent across different validation sets.

		Split	Mean			Standard deviation	
		variable	Training score	Test score	Execution time (s)	Training score	Test score
Model	Batch size	Epochs					
GRU100	128	100	0.005093	0.282721	8.554641	0.001672	0.226439
LSTM100>LSTM60	128	100	0.002296	0.271617	14.253503	0.000551	0.239281
LSTM150>LSTM100	128	100	0.002150	0.276615	17.031967	0.000540	0.251289
LSTM100>LSTM100	128	100	0.002572	0.280945	14.815283	0.000663	0.254216
LSTM150	128	100	0.002574	0.299442	10.322127	0.000647	0.264279

Figure 12: Top 5 Models in the Grid Search Based on Mean STD

• Speed Evaluation Sorted by Mean Execution Time

The LSTM100>LSTM60 model has an execution time of 14.253503 seconds. Compared to other models listed, it is not the fastest, but it falls within a moderate range.

		Split	Mean			Standard deviation	
		variable	Training score	Test score	Execution time (s)	Training score	Test score
Model	Batch size	Epochs					
GRU100	128	100	0.005093	0.282721	8.554641	0.001672	0.226439
LSTM100	128	100	0.002942	0.309481	8.963943	0.000835	0.310214
LSTM150	128	100	0.002574	0.299442	10.322127	0.000647	0.264279
GRU150	128	100	0.004947	0.324070	12.903294	0.001558	0.300183
GRU100	128	200	0.002226	0.368083	13.241707	0.000557	0.370320
LSTM100	128	200	0.001177	0.417765	13.856914	0.000291	0.499143
GRU100>GRU60	128	100	0.003546	0.425911	13.934243	0.000681	0.571921
GRU100>LSTM60	128	100	0.003059	0.306081	14.128961	0.000797	0.306576
LSTM100>LSTM60	128	100	0.002296	0.271617	14.253503	0.000551	0.239281
LSTM100>GRU100	128	100	0.002773	0.319344	14.406128	0.000700	0.354902

Figure 13: Top 10 Models in the Grid Search Based on Execution Time

In summary, when evaluating RNN models that include LSTM, GRU, and their combinations, the LSTM models notably outperformed others in terms of prediction accuracy, consistency across validations, and execution speed. Specifically, the LSTM100>LSTM60 configuration emerged as the most effective, balancing performance and robustness quite well. Compared to the decision tree algorithm XGBRegressor, the LSTM models demonstrated superior predictive capabilities of flight numbers, making them the preferred choice for this time series forecasting task.

After selecting the LSTM(100) > LSTM(60) model as the best performer, we utilized it to forecast Ukraine's flight count under the ongoing war conditions in an attempt to answer the proposed research question. The resulting plot indicates a significant deviation from expected outcomes, with the model **inadequately** reflecting the cessation of flights caused by war. This limitation arises from the model's treatment of the "is_war" feature as a continuous variable rather than as a binary switch that would accurately represent the war's impact.

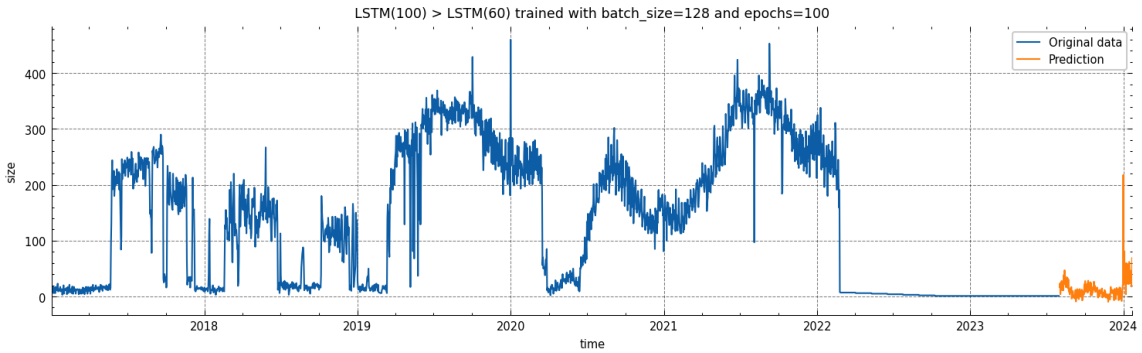


Figure 14: Future Forecast Results of the Selected Model for Ukraine's Flight Count in the State of War

To address this, methods like intervention analysis, which adjusts for sudden shifts in time series data, could be considered (Pennstate Eberly College of Science, 2023; Matarise, 2011). Despite this, the model's inability to accurately capture the binary nature of war's impact on flights has highlighted a critical area where traditional forecasting struggles—modeling binary events like war. This challenge underscores the need for more nuanced approaches or tailored models to account for such deterministic external factors.

In light of these findings, our model's forecast serves as a baseline for normal flight activity without the war indicator. This limitation means that while the model provides insights into regular flight patterns, it falls short in scenarios with abrupt, impactful events like war, suggesting a direction for future methodological improvements in the field of forecasting (Flovik, 2021; Krishnamurthi, Narayan and Raj, 1989).

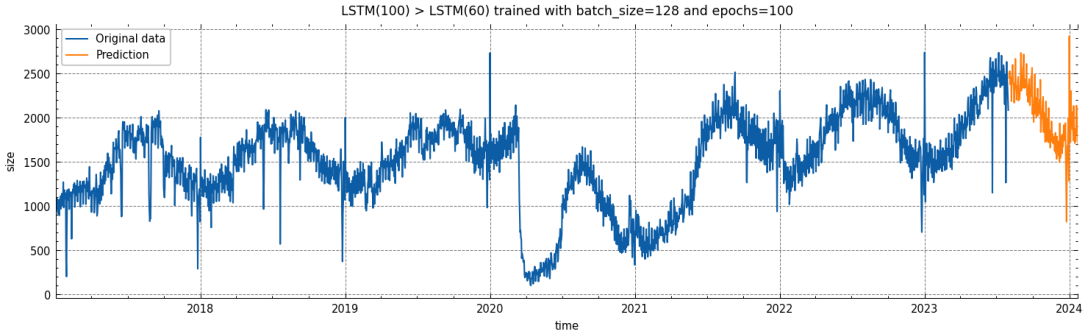


Figure 15: Future Forecast Results of the Selected Model for the Region's Flight Count in the State of War

The prediction suggests a gradual increase in flight counts heading into 2023, followed by a pronounced rise and subsequent sharp decline as it approaches 2024. This could indicate the model's attempt to capture cyclic trends or react to potential disruptions anticipated based on historical patterns such as Christmas.

6.2 High-Performance Computing Evaluation & Comparison

In our high-performance computing evaluation, we focused on optimizing data storage and query efficiency. We transformed our dataset into the ORC format, known for its superior performance in read, write, and processing speeds, particularly in large-scale data environments (LanguageManual ORC - Apache Hive, 2018). Utilizing this format allows us to capitalize on its columnar storage capabilities, which significantly reduces the data footprint and improves analytical processing times.

Our benchmarking exercise pitted Apache Hive against Impala, two prominent big data technologies, with a specific query (Appendix F) designed to test their performance. This query extracted latitude and longitude from a concatenated string in the airports' data, then created a view with separate fields for these coordinates and finally performed a LEFT JOIN operation with the flights' data to associate each flight with its respective departure and arrival points.

To ensure a fair comparison, we executed the query multiple times on both systems to mitigate any potential caching advantages. We meticulously recorded the 'real' time, which reflects the total elapsed time during the query execution and is most indicative of what a user would experience.

• Apache Hive Results:

real	**7m55.694s**
user	**0m57.110s**
sys	**0m8.767s**

Table 1: Query Execution Benchmark Results using Apache Hive

• Apache Impala Results:

real	**7m3.024s**
user	**1m2.308s**
sys	**0m4.116s**

Table 2:Query Execution Benchmark Results using Apache Impala

Our results showed that Impala outperformed Hive, completing the query in a shorter 'real' time. This aligns with expectations, as Impala is engineered for speed with its MPP architecture, which bypasses the MapReduce overhead that Hive relies on. Despite this, the performance gain was not as significant as one might expect, considering that our data processing was constrained by hardware limitations, specifically HDDs rather than faster SSDs or in-memory operations which Impala leverages for high speed.

Additionally, the system resource usage indicated a slight uptick in CPU time for Impala, attributed to the increased demands of disk I/O operations, which are more prevalent when data cannot be fully accommodated in memory. This was also reflected in the 'sys' time, suggesting that Impala's execution led to more time spent in kernel mode, handling these I/O processes. Overall, while Impala showed a clear advantage in processing speed, the comparison highlighted the impact of hardware constraints on performance outcomes in this type of query.

7. Discussion of Findings

This investigation into the flight patterns in Ukraine and its neighboring regions has yielded comprehensive insights, particularly in the context of the conflict within Ukraine. Historical data analysis has demonstrated that, prior to the conflict, Ukraine's airspace exhibited consistent and cyclical patterns of departures and arrivals, reflective of economic activity, seasonal tourism, and regional connectivity. The analysis of time series data via machine learning models, such as XGBRegressor and LSTM, corroborated these patterns, underscoring the regular flow of flight traffic characteristic of vibrant airspace.

However, the onset of conflict marked a paradigm shift. The war's immediate effect was palpable—a precipitous drop in flight counts, as captured by our LSTM(100)>LSTM(60) model emerging as superior to all other models, painting a stark contrast to the pre-conflict era. This sharp decline is indicative of the profound disruptions in civil aviation, with airspace closures and the rerouting of international flights. The conflict's impact transcends the operational aspect, hinting at broader socio-economic repercussions, including isolation and a potential decline in regional economic integration.

Our predictive models, trained on historical data, faced challenges when accounting for the binary nature of the 'is_war' feature. They initially failed to accurately simulate the abrupt cessation of flights caused by the conflict. This shortcoming stresses the limitations of conventional forecasting models in the face of binary, deterministic events like war. It also highlights the need for incorporating specialized techniques such as intervention analysis to better model the impact of such unforeseen events on time series data.

Despite these challenges, our findings contribute valuable perspectives on the resilience and vulnerability of air traffic networks in conflict zones. While the region's historical flight patterns displayed robustness against routine disruptions, the conflict has underscored the fragility of such networks under extreme conditions.

In conclusion, the historical patterns of flight activity in Ukraine and its vicinity were significantly altered due to the conflict. The research affirms that while machine learning can capture and forecast standard patterns in air traffic data, the modeling of sudden geopolitical events requires a nuanced approach, potentially blending traditional methods with advanced analytics to capture the full spectrum of impacts.

8. Data Management Plan and Author Contribution Statement

• Data Management Plan:

The Data Management Plan (Appendix H) outlines a comprehensive process beginning with data collection from OpenSky Network and Aviation Edge API. Data preparation involves transforming and compiling data into CSV format, integrating additional datasets like 'Airports' and 'Workalendar', and flagging relevant features such as lockdowns and holidays. A Hadoop Cluster is employed for data storage and organization, ensuring scalability and robustness. For data processing, timestamps are converted into features, seasonality is identified, categories are one-hot encoded, and data is scaled. The ML Prediction & Evaluation section details the use of various algorithms, including XGBRegressor, LSTM, GRU, and hybrid models, with performance evaluated based on metrics like Mean Squared Error. Finally, HPC Implementation & Evaluation assesses the effectiveness of Apache Hive and Impala, enhancing the computational efficiency of the project.

- **Authorship Contribution Statement :**

This study was conducted in collaboration with Hessan Badami (Student ID: 2350719). The work was divided as follows:

Both authors contributed equally to the majority of the data acquisition process and research. Specific contributions include the author of this report (Mary Dannoun, Student ID: 2249082) leading the development of the RNN LSTM algorithm in the machine learning section and the exploration of Apache Hive in the high-performance computing section. Hessan spearheaded the integration of the RNN GRU algorithm within the machine learning framework and focused on the utilization of Apache Impala for high-performance computing tasks. Each author's contribution was instrumental in the execution and analysis within their respective areas of the project.

9. References

Badami, M.H. and Dannoun, M. (2023). Github: Source Code. https://github.com/hesbadami/dda.

Schäfer, M. (2016) Historical database. https://opensky-network.org/data/impala.

Aviation Edge (2023) Aviation Edge Database Download - Aviation database and API. https://aviation-edge.com/aviation-database/.

Hadoop File Formats Support (2023). https://docs.cloudera.com/cdw-runtime/cloud/impala-reference/topics/impala-file-formats.html.

Dataset formats OurAirports (2023). https://ourairports.com/help/data-dictionary.html.

Workalendar (2023). GitHub - workalendar/workalendar: Worldwide holidays and workdays computational toolkit. https://github.com/workalendar/workalendar.

Wohlwend, B. (2023) 'Decision Tree, Random Forest, and XGBoost: An Exploration into the Heart of Machine Learning,' Medium, 12 August. https://medium.com/@brandon93.w/decision-tree-random-forest-and-xgboost-an-exploration-into-the-heart-of-machine-learning-90dc212f4948.

Seanyboi (2022) GitHub - seanyboi/lockdowndates: Retrieve the dates of the restrictions imposed by governments in countries around the world during the covid-19 pandemic. https://github.com/seanyboi/lockdowndates.

Badami, M.H. and Dannoun, M. (2023) GitHub - Forecaster: a time series forecasting package written in python. https://github.com/Hesbadami/Forecaster.

Matarise, F. (2011) 'Intervention analysis in time series,' in Springer eBooks, pp. 682–685. https://doi.org/10.1007/978-3-642-04898-2_308.

Pennstate Eberly College of Science (2023). 9.2 Intervention Analysis | STAT 510. https://online.stat.psu.edu/stat510/lesson/9/9.2#:~:text=Intervention%20analysis%20in%20time%20series,before%20and%20after%20the%20intervention.

Flovik, V., PhD (2021) How (not) to use Machine Learning for time series forecasting: The sequel. https://www.linkedin.com/pulse/how-use-machine-learning-time-series-forecasting-vegard-flovik-phd-1f/.

Krishnamurthi, L., Narayan, J. and Raj, S.P. (1989) 'Intervention analysis using control series and exogenous variables in a transfer function model: A case study,' International Journal of Forecasting, 5(1), pp. 21–27. https://doi.org/10.1016/0169-2070(89)90060-5.

CS5706 Week 4 Teaching Materials (2024). Lecture 4: Cluster Analysis I: Agglomerative Clustering. Available at https://brightspace.brunel.ac.uk.

CS5706 Week 5 Teaching Materials (2024). Lecture 5: Cluster Analysis II: K-Means Clustering. Available at https://brightspace.brunel.ac.uk.

CS5706 Week 7 Teaching Materials (2024). Lecture 7: Predictive Methods II: Tree-based Methods. Available at https://brightspace.brunel.ac.uk.

Kumar, S G.V and Jaisharma, K (2023). Improve the Accuracy for Flight Ticket Prediction using XGBRegressor Optimizer in Comparison with Extra TreeRegressor Performance. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10397633.

Python Package Introduction — xgboost 2.0.3 documentation (2022). https://xgboost.readthedocs.io/en/stable/python/python_intro.html.

CS5706 Week 6 Teaching Materials (2024). Lecture 6: Predictive Methods I: Neural Networks. Available at https://brightspace.brunel.ac.uk.

Han, P., Wang W., Shi, Q., and Yang, J. (2019). Real-time short- term trajectory prediction based on GRU neural network (2019). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9081618.

Kovelamudi, N. and Chollet, F. (2023). Keras documentation: Save, serialize, and export models. https://keras.io/guides/serialization_and_saving/.

Keras documentation: Adam (2023) . https://keras.io/api/optimizers/adam/.

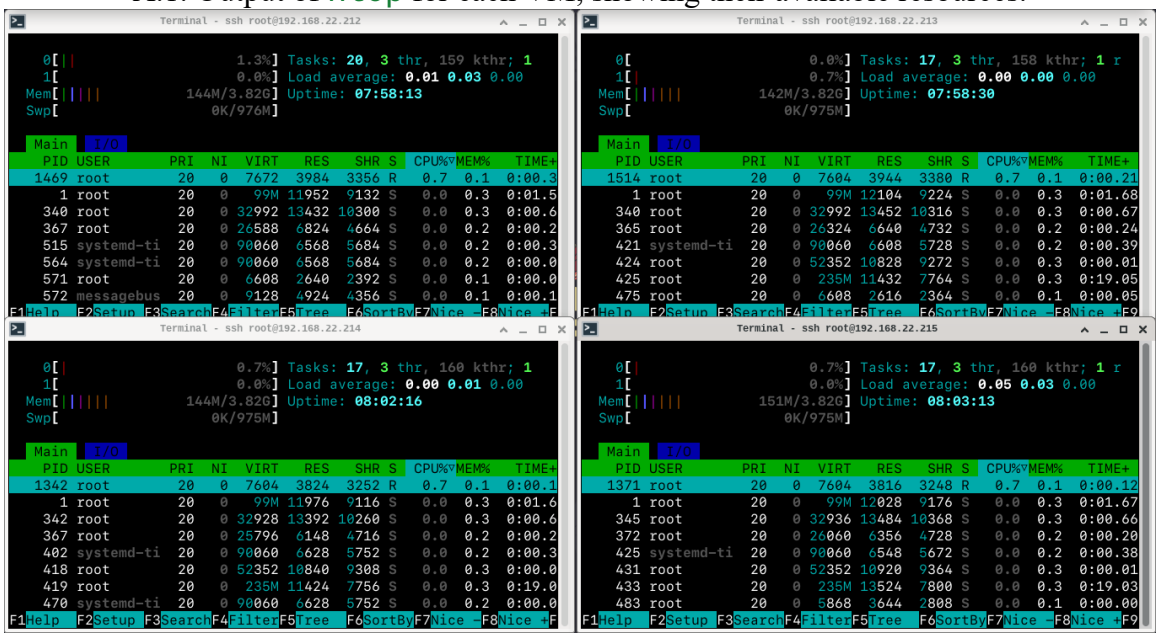
LanguageManual ORC - Apache Hive - Apache Software Foundation (2018). https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC.

Apache Impala Documentation (2023). How Impala Works with Hadoop File Formats. https://impala.apache.org/docs/build/html/topics/impala_file_formats.html.

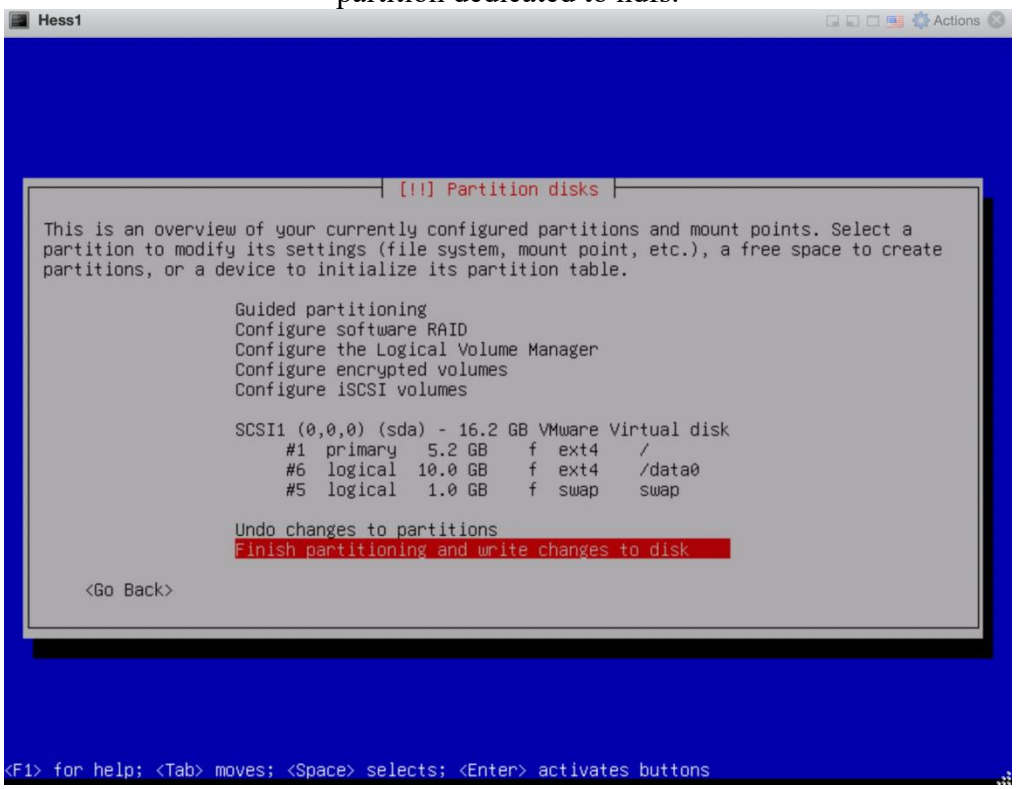
10. Appendix

APPENDIX A: DEBIAN INSTALLATION

A.1. Output of `htop` for each VM, showing their available resources.



A.2. Partition table of each VM. Each having a primary partition for the OS, and a 10.0GB partition dedicated to hdfs.



APPENDIX B: HADOOP SETUP CODE

Block B.1. Content of `core-site.xml` configuration.

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://namenode:9000</value>
    </property>
    <property>
        <name>dfs.permissions.enabled</name>
        <value>false</value>
    </property>
</configuration>
```

Block B.2. Content of `hdfs-site.xml` configuration.

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/mnt/hadoop_data/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/mnt/hadoop_data/hdfs/datanode</value>
    </property>
    <property>
        <name>dfs.permissions.enabled</name>
        <value>false</value>
    </property>
</configuration>
```

Block B.3. Content of `mapred-site.xml` configuration.

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
       <name>yarn.app.mapreduce.am.env</name>
       <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
    <property>
       <name>mapreduce.map.env</name>
       <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
    <property>
       <name>mapreduce.reduce.env</name>
       <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
```

```
    <property>
       <name>mapreduce.application.classpath</name>
       <value>${HADOOP_CLASSPATH}</value>
    </property>
</configuration>
```

Block B.4. Content of `yarn-site.xml` configuration.

```
<configuration>
    <property>
       <name>yarn.nodemanager.aux-services</name>
       <value>mapreduce_shuffle</value>
    </property>

    <property>
       <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
       <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

Block B.5. Content of `~/.ssh/config` configuration.

```
Host *
    User root

Host namenode
    Hostname 192.168.22.212
Host datanode0
    Hostname 192.168.22.213
Host datanode1
    Hostname 192.168.22.214
Host datanode2
    Hostname 192.168.22.215
```

Block B.6. Content of synchronization script.

```
#!/bin/sh
for i in {0..2}; do
    node="datanode$i"
    echo "Syncing sshconfig to $node."
    scp -p $sshconfig $node:/root/.ssh/config
    echo "Syncing hosts to $node."
    ssh $node "grep -q '192.168.22.212 namenode' /etc/hosts || echo '192.168.22.212
namenode' | tee -a /etc/hosts"
    ssh $node "grep -q '192.168.22.213 datanode0' /etc/hosts || echo '192.168.22.213
datanode0' | tee -a /etc/hosts"
    ssh $node "grep -q '192.168.22.214 datanode1' /etc/hosts || echo '192.168.22.214
datanode1' | tee -a /etc/hosts"
    ssh $node "grep -q '192.168.22.215 datanode2' /etc/hosts || echo '192.168.22.215
datanode2' | tee -a /etc/hosts"
    echo "Syncing hadooprc to $node."
    scp -p $hadooprc $node:/etc/profile.d/hadooprc.sh
    echo "Syncing java to $node"
    rsync -ravl $JAVA_HOME $node:/usr/lib/jvm/
    echo "Syncing hadoop configuration files to $node."
```

```
    rsync -ravl $HADOOP_HOME $node:/usr/local/
done
echo "Sync complete."
```

Block B.7. Content of `/etc/profile.d/hadooprc.sh` startup script.

```
#!/bin/sh

# Environment Variables for Hadoop Cluster
export hadooprc=/etc/profile.d/hadooprc.sh
export sshconfig=~/.ssh/config

# IPs
export namenode="192.168.22.212"
export datanode0="192.168.22.213"
export datanode1="192.168.22.214"
export datanode2="192.168.22.215"

# Java
export JAVA_HOME=/usr/lib/jvm/default-java
export PATH=$PATH:$JAVA_HOME/bin

# PDSH
export PDSH_RCMD_TYPE=ssh

# Hadoop Environment Variables
export HADOOP_HOME="/usr/local/hadoop"
export HADOOP_CONF_DIR="${HADOOP_HOME}/etc/hadoop"
export YARN_EXAMPLES="${HADOOP_HOME}/share/hadoop/mapreduce"
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

export HDFS_NAMENODE_USER="root"
export HDFS_DATANODE_USER="root"
export HDFS_SECONDARYNAMENODE_USER="root"
export YARN_RESOURCEMANAGER_USER="root"
export YARN_NODEMANAGER_USER="root"
```

APPENDIX C: Hive Setup

Step 1: Install Hive
1. Download Hive: Go to the Apache Hive official website and download the appropriate version for our Hadoop version.
2. Spawn a PostgreSQL database for hive metastore using docker.

```
docker run -d --name hive-postgres -e POSTGRES_PASSWORD=hivejoon -e
POSTGRES_USERNAME=hive -p 5432:5432 postgres
```

3. Extract and Set Up Hive:
Extract the downloaded Hive tar file to /usr/local/hive directory on our NameNode.
Set HIVE_HOME and add Hive's bin directory to PATH in our shell's configuration file. (Ref Appendix B)

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

Configure Hive by editing the hive-site.xml file with appropriate settings for our environment, including metastore and driver details.

4. Initialize Metastore: Run the schema initialization script provided by Hive to set up the metastore.

```
schematool -dbType postgres -initSchema
hive --service metastore
```

Content of hive-site.xml:

```
<configuration>
        <property>
            <name>javax.jdo.option.ConnectionURL</name>
            <value>jdbc:postgresql://localhost:5432/postgres</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionDriverName</name>
            <value>org.postgresql.Driver</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionUserName</name>
            <value>hive</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionPassword</name>
            <value>hivejoon</value>
        </property>
</configuration>
```

APPENDIX D: Data Conversion from CSV to ORC

To convert our data from CSV into ORC which is a much more efficient storage format, we first create a Hive table that matches the schema of our CSV file. We will then load our CSV data into this table.

```
CREATE TABLE flights_csv (
    icao24 STRING,
    firstseen TIMESTAMP,
    lastseen TIMESTAMP,
    estdepartureairport STRING,
    estarrivalairport STRING,
    day TIMESTAMP
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```
Time taken: 1.482 seconds

```
LOAD DATA INPATH '/user/root/flight_data' INTO TABLE flights_csv;
```
Time taken: 0.737 seconds

Next, we create a new Hive table that will store data in the ORC format:

```
CREATE TABLE flights_orc (
    icao24 STRING,
    firstseen TIMESTAMP,
    lastseen TIMESTAMP,
    estdepartureairport STRING,
    estarrivalairport STRING,
    day TIMESTAMP
)
STORED AS ORC;
```

We will now insert the data from the original CSV-formatted table into the new ORC-formatted table:

```
INSERT INTO TABLE flights_orc SELECT * FROM flights_csv;
```
Time taken: 499.79 seconds

A quick comparison to show how efficient ORC is compared to CSV:

```
hadoop fs -du -s -h /user/hive/warehouse/*
```

output:

```
8.9 G  17.8 G  /user/hive/warehouse/flights_csv_temp
3.1 G  6.2 G   /user/hive/warehouse/flights_orc
```

APPENDIX E: Grid Search Implementation With Hyperthreading for Hyperparameter Fine-tuning

Importing libraries:

```
import tensorflow as tf
import pandas as pd
from datetime import datetime
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, GRU, TimeDistributed,
Flatten, Conv1D, MaxPooling1D
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.losses import MeanSquaredError
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import ParameterGrid
import threading
```

Defining functions:

```
def parameter_grid(layers= [], layer_models = [], layers_param_grid = [],
train_param_grid = {}):

    if max(layers) > len(layers_param_grid):
        raise "Number of layers exceeds number of layer parameter grids passed"

    # Get all possible combinations of NN layers
    combinations = []
    for n_layers in layers:
        combinations += itertools.product(layer_models, repeat=n_layers)

    architectures = []
    for combination in combinations:
        architecture = {}
        for i_layer, arch_layer in enumerate(combination):
            layer_params = {'model': [arch_layer]} | layers_param_grid[i_layer]
            if i_layer != len(combination)-1:
                layer_params['return_sequences'] = [True]
            else:
                layer_params['return_sequences'] = [False]

            architecture[i_layer] = list(ParameterGrid(
                layer_params
            ))

        architectures.append(architecture)

    param_grid = list(ParameterGrid(
        train_param_grid | {'arch': list(ParameterGrid(architectures))}
    ))

    param_grid_increasing_NN =
    for p in param_grid:
        init =1 1e10
        for i in p["arch"]:
```

```
            if p["arch"][i]["units"] > init:
                del p[""]

    for p in param_grid:
        p['arch_name'] = ">".join([f"""{str(p["arch"][i]["model"]).split(".")[-
1].split("'")[0]}{p["arch"][i]["units"]}""" for i in p["arch"]])

    return param_grid

def split_handler(split, model, fit_kwargs, scorer, index, results):
    forecaster = Forecaster(
        split,
        x = 'time',
        y = 'size',
        group_features = ['country'],
        categorical_features = ['dayofmonth', 'quarter'],
        scoring_metric = scorer
    )

    a = time.time()
    result = forecaster.validate(
        model,
        seasonality = True,
        fit_kwargs = fit_kwargs
    )
    b = time.time()

    result += (b-a,)
    results[index] = result

def grid_search_nn(df, scorer, cv, param_grid, n_threads = 1):
    # Get date splits
    dates = df['time'].unique()
    split_dates = pd.date_range(df['time'].min(), df['time'].max(), periods=cv+1)
    splits = [
        df[(df['time'] >= split_dates[i]) & (df['time'] <= split_dates[i+1])] \
        for i in range(len(split_dates) - 1)
    ]

    counter = 0
    threads = []
    results = {}

    start = time.time()
    for param in param_grid:
        sequence = []
        for i in param['arch']:
            sequence.append(
                param['arch'][i]['model'](
                    units = param['arch'][i]['units'],
                    return_sequences = param['arch'][i]['return_sequences']
                )
            )
            sequence.append(Dropout(0.1))
```

```
        sequence.append(Dense(units = 1))
        name = param['arch_name']

        model = Sequential(sequence, name = name)
        model.compile(optimizer = Adam(), loss = MeanSquaredError())
        fit_kwargs = {'epochs': param['epochs'], 'batch_size': param['batch_size'],
'verbose': 0}

        for i_split, split in enumerate(splits):
            print(
                f"""{counter+1}/{len(splits)*len(param_grid)}: {name},
split_{i_split}, epochs={param['epochs']}, batch_size={param['batch_size']}""",
                end="\r"
            )

            index = (name, param['batch_size'], param['epochs'], f'split_{i_split}',
'time')

            while counter - len(results) >= n_threads:
                time.sleep(1)

            thread = threading.Thread(target=split_handler, args=(split, model,
fit_kwargs, scorer, index, results,))
            thread.start()
            threads.append(
                thread
            )
            counter += 1

    for thread in threads:
        thread.join()

    end = time.time()

    return results, end-start
```

Execution:

```
param_grid = parameter_grid(
    layers = [1, 2],
    layer_models = [LSTM, GRU],
    layers_param_grid = [{'units': [100, 150]}, {'units': [60, 100]}],
    train_param_grid = {'epochs': [100, 200], 'batch_size': [32, 128]}
)
results, exec_time = grid_search_nn(df, scorer=metrics.mean_squared_error, cv=5,
param_grid=param_grid, n_threads=5)
```

Cleaning results:

```
scores = [results[result][3] for result in results]
indexes = [result[:-2] for result in results]
splits = [result[-2] for result in results]
exec_times = [results[result][-1] for result in results]

index = pd.MultiIndex.from_tuples(indexes, names=['Model', 'Batch size', 'Epochs'])
```

```
grid = pd.DataFrame(
    ([score.values.flatten()[2:] for score in scores]),
    columns=['Training score', 'Test score'],
    index=index
)

grid['Split'] = splits
grid['Execution time (s)'] = exec_times

grid = pd.melt(grid.reset_index(), id_vars=['Model', 'Batch size', 'Epochs', 'Split'],
value_vars=['Training score', 'Test score', 'Execution time (s)'])
mean_df = grid.groupby(['Model', 'Batch size', 'Epochs', 'variable'],
as_index=False)['value'].mean()
mean_df['Split'] = 'Mean'
std_df = grid.groupby(['Model', 'Batch size', 'Epochs', 'variable'],
as_index=False)['value'].std()
std_df['Split'] = 'Standard deviation'

grid = grid.pivot_table(index=['Model', 'Batch size', 'Epochs'], columns=['Split',
'variable'], values='value').reindex(['Training score', 'Test score', 'Execution time
(s)'], level='variable', axis=1)
mean_df = mean_df.pivot_table(index=['Model', 'Batch size', 'Epochs'],
columns=['Split', 'variable'], values='value').reindex(['Training score', 'Test
score', 'Execution time (s)'], level='variable', axis=1)
std_df = std_df.pivot_table(index=['Model', 'Batch size', 'Epochs'], columns=['Split',
'variable'], values='value').reindex(['Training score', 'Test score'],
level='variable', axis=1)

grid = pd.concat([mean_df, std_df, grid], axis=1)
```

Final output:

		Split	Mean		Standard deviation		split_0			split_1			split_2			split_3			split_4		
		variable	Training score	Test score	Training score	Test score	Training score	Test score	Execution time (s)	Training score	Test score	Execution time (s)	Training score	Test score	Execution time (s)	Training score	Test score	Execution time (s)	Training score	Test score	Execution time (s)
Model	Batch size	Epochs																			
GRU100	32	100	0.002328	0.445048	0.000644	0.492349	0.002820	1.309636	17.942579	0.002497	0.119961	17.888630	0.002001	0.374001	17.995761	0.002944	0.239658	17.761976	0.001379	0.181982	17.817795
		200	0.001278	0.528587	0.000348	0.591908	0.001651	1.562406	31.833116	0.001413	0.105359	31.750239	0.001387	0.384657	31.634971	0.001222	0.404608	31.808236	0.000719	0.185908	31.683462
	128	100	0.005093	0.282721	0.001672	0.226439	0.006280	0.660518	8.549256	0.006823	0.109963	8.524637	0.005312	0.325451	8.563129	0.004481	0.159848	8.592578	0.002570	0.157825	8.543606
		200	0.002226	0.368083	0.000557	0.370320	0.002615	1.012702	13.143884	0.002700	0.113617	13.187230	0.002532	0.344573	13.288493	0.001852	0.169064	13.294304	0.001432	0.200456	13.294622
GRU100>GRU100	32	100	0.002138	0.425888	0.000401	0.490533	0.002796	1.287471	28.302410	0.001966	0.074520	28.389412	0.001777	0.315426	28.354922	0.002222	0.273992	26.323065	0.001931	0.178031	27.621510
		200	0.001231	0.508402	0.000238	0.631199	0.001235	1.615575	49.996034	0.001094	0.059860	49.285533	0.001512	0.401999	49.653215	0.000913	0.269915	49.603486	0.001400	0.194664	48.873229
	128	100	0.003688	0.435116	0.001257	0.521367	0.003270	1.354791	14.566338	0.004967	0.109339	14.562422	0.004394	0.346050	15.883518	0.004085	0.182018	16.830461	0.001726	0.183383	17.144001
		200	0.002094	0.431116	0.001294	0.483775	0.004402	1.280098	24.694806	0.001524	0.097211	25.071203	0.001575	0.360392	24.239927	0.001618	0.217997	23.577202	0.001352	0.199881	23.316865
GRU100>GRU60	32	100	0.001698	0.535193	0.000326	0.713195	0.001636	1.795126	26.429738	0.001794	0.077972	25.215940	0.001554	0.363251	25.225853	0.002192	0.289117	26.047115	0.001315	0.150502	26.233952
		200	0.000962	0.496401	0.000240	0.620342	0.000786	1.588192	43.267073	0.001082	0.069232	43.192171	0.001203	0.344670	44.480132	0.001103	0.311603	43.920987	0.000636	0.168308	44.440460
	128	100	0.003546	0.425911	0.000681	0.571921	0.003450	1.438696	11.025121	0.004529	0.077421	14.053664	0.003798	0.301430	14.020793	0.003264	0.156684	15.289087	0.002690	0.155322	15.282550
		200	0.001631	0.398619	0.000515	0.467632	0.001255	1.213987	36.748862	0.001703	0.061470	36.621220	0.002197	0.353261	36.597736	0.002030	0.197724	36.340637	0.000971	0.166651	36.586517
GRU100>LSTM100	32	100	0.001561	0.522183	0.000195	0.728992	0.001535	1.813079	32.679067	0.001665	0.079129	32.668599	0.001403	0.353214	32.556892	0.001837	0.227591	32.337314	0.001363	0.137900	32.339850
		200	0.000892	0.504582	0.000365	0.662118	0.000943	1.669425	49.871485	0.000687	0.059822	49.738329	0.000839	0.366817	49.486274	0.001478	0.285552	49.046256	0.000514	0.141292	49.616710
	128	100	0.002924	0.337724	0.000984	0.314444	0.003824	0.866174	14.840680	0.003097	0.093366	25.081742	0.003622	0.382309	25.228451	0.002733	0.170626	24.382462	0.001341	0.179146	24.518418
		200	0.001572	0.409641	0.000660	0.491390	0.001318	1.273333	33.572506	0.001987	0.074976	25.419256	0.001185	0.331830	25.232186	0.002502	0.188497	25.278921	0.000870	0.179570	22.817653
GRU100>LSTM60	32	100	0.001701	0.446457	0.000358	0.561459	0.001654	1.434965	26.268449	0.001303	0.072553	26.276090	0.001964	0.342281	26.085493	0.002156	0.229144	25.266922	0.001427	0.153344	25.153327
		200	0.001165	0.573534	0.000179	0.826513	0.000973	2.041369	43.170842	0.001159	0.077966	44.376058	0.001433	0.351641	44.349925	0.001222	0.226684	43.271002	0.001039	0.170012	43.463338
	128	100	0.003059	0.306081	0.000797	0.306576	0.003672	0.833290	13.972888	0.003125	0.071623	14.206609	0.003245	0.304934	14.319420	0.003563	0.144192	14.012776	0.001689	0.176365	14.133114
		200	0.001571	0.382711	0.000369	0.450186	0.002093	1.170268	21.049462	0.001286	0.083428	21.757583	0.001183	0.339143	21.792085	0.001765	0.166330	21.932304	0.001529	0.154385	21.827690
GRU150	32	100	0.001996	0.497210	0.000374	0.614127	0.002295	1.588327	20.461593	0.002358	0.148916	20.464881	0.001980	0.338763	20.481842	0.001931	0.226347	20.336719	0.001417	0.183695	20.477081
		200	0.001185	0.516969	0.000283	0.581177	0.001666	1.535003	36.064447	0.001160	0.107582	36.154802	0.001048	0.356480	36.145182	0.001121	0.395106	35.979551	0.000930	0.190672	36.134685
	128	100	0.004947	0.324070	0.001558	0.300183	0.006017	0.844659	12.855872	0.006486	0.121793	12.820956	0.005226	0.317086	12.850642	0.004488	0.168680	12.873747	0.002517	0.168135	13.115253
		200	0.002083	0.416358	0.000542	0.419405	0.002524	1.148565	15.590174	0.002503	0.178093	15.660150	0.002210	0.391120	15.598041	0.001976	0.171309	15.678278	0.001201	0.192704	15.666696
GRU150>GRU100	32	100	0.002188	0.498692	0.001007	0.674278	0.002853	1.689279	32.376015	0.001584	0.076740	32.685512	0.001516	0.369780	32.649110	0.003636	0.208987	32.449132	0.001350	0.148674	32.328679
		200	0.000911	0.501275	0.000119	0.622740	0.001074	1.594176	57.584900	0.000767	0.055646	57.893413	0.000917	0.343600	57.906880	0.000964	0.338597	57.580155	0.000833	0.174359	57.628472
	128	100	0.003240	0.368747	0.000960	0.408077	0.003781	1.085298	15.826738	0.004290	0.080189	16.106205	0.003292	0.300502	25.242751	0.003106	0.192658	16.116980	0.001733	0.185086	15.578202
		200	0.001620	0.452478	0.000485	0.533910	0.001786	1.386689	24.220836	0.001299	0.081305	23.170254	0.002381	0.391234	22.540227	0.001464	0.207639	22.202570	0.001167	0.195524	19.481873
GRU150>GRU60	32	100	0.001673	0.508973	0.000303	0.649474	0.001415	1.655471	30.412317	0.001854	0.092239	29.798579	0.001394	0.348680	29.811424	0.002103	0.299389	29.820932	0.001600	0.149085	29.934379
		200	0.001072	0.593557	0.000488	0.841642	0.000687	2.086963	52.570093	0.000906	0.065282	52.509087	0.001172	0.346978	52.287768	0.001874	0.285015	53.065097	0.000718	0.183547	52.786481
	128	100	0.002904	0.453367	0.000796	0.576468	0.002585	1.476760	15.175778	0.003050	0.099762	15.595020	0.003807	0.300043	15.629716	0.003352	0.192051	14.740930	0.001724	0.198221	15.136439
		200	0.001393	0.408345	0.000322	0.464234	0.001267	1.221369	23.383844	0.001338	0.076923	24.417068	0.001569	0.343677	24.406143	0.001823	0.207050	23.870833	0.000967	0.192704	23.703978
GRU150>LSTM100	32	100	0.001603	0.466299	0.000289	0.590796	0.001631	1.506105	32.734347	0.001257	0.069717	33.364950	0.001372	0.349618	33.343229	0.001812	0.255215	33.242958	0.001944	0.150841	33.242439
		200	0.000925	0.576183	0.000418	0.813820	0.001547	2.019864	59.978427	0.000768	0.076849	61.616800	0.000573	0.349338	61.421859	0.001150	0.275731	61.382902	0.000587	0.159135	61.669997
	128	100	0.002339	0.347130	0.000630	0.369746	0.002714	0.986052	16.398681	0.002573	0.087262	16.304996	0.002712	0.342575	16.250367	0.002468	0.138081	16.250565	0.001227	0.181682	16.499033
		200	0.001035	0.364966	0.000218	0.427383	0.001043	1.112675	25.678040	0.001012	0.065863	28.178249	0.001318	0.315402	28.168709	0.001093	0.168464	28.087565	0.000710	0.162425	28.063981
GRU150>LSTM60	32	100	0.001538	0.475820	0.000368	0.626862	0.001949	1.585006	31.442737	0.001485	0.073911	31.344653	0.001203	0.325158	31.344598	0.001885	0.232066	31.327330	0.001168	0.162958	31.349010
		200	0.001087	0.657555	0.000117	1.019134	0.001238	2.471365	52.571493	0.000921	0.075133	53.021509	0.001128	0.331224	53.157239	0.001107	0.273437	53.195451	0.001039	0.136617	53.220342
	128	100	0.002625	0.346594	0.000832	0.372250	0.002633	0.992164	15.725674	0.003700	0.082171	16.459139	0.002879	0.331918	16.470559	0.002531	0.152543	16.428683	0.001383	0.174173	16.428485
		200	0.001805	0.446522	0.000761	0.565899	0.002524	1.444312	23.454290	0.001240	0.072769	25.067999	0.000860	0.333094	25.122731	0.002572	0.226324	25.135177	0.001830	0.156113	24.923411
LSTM100	32	100	0.001701	0.509134	0.000551	0.681160	0.002405	1.714800	18.605750	0.002082	0.110239	18.613064	0.001632	0.372334	18.654721	0.001354	0.177488	18.601504	0.001030	0.170809	18.576161
		200	0.001029	0.482848	0.000220	0.604422	0.001264	1.550779	32.735802	0.001238	0.108727	32.717268	0.000952	0.360631	32.721681	0.000948	0.232869	32.714504	0.000743	0.161235	32.797479
	128	100	0.002942	0.309481	0.000835	0.310214	0.002904	0.845294	8.952358	0.003785	0.097518	8.974270	0.003599	0.309891	8.998363	0.002750	0.134406	8.960196	0.001674	0.160293	8.934530
		200	0.001177	0.417765	0.000291	0.499143	0.001235	1.294367	13.874962	0.001575	0.111926	13.876360	0.001140	0.356625	13.793860	0.001180	0.140498	13.846983	0.000758	0.185407	13.892405
LSTM100>GRU100	32	100	0.001830	0.423628	0.000406	0.514391	0.001822	1.328005	28.028571	0.002345	0.090115	27.983778	0.001292	0.350473	27.458720	0.002075	0.180138	27.367032	0.001617	0.169409	27.003989
		200	0.000880	0.511263	0.000314	0.646523	0.000928	1.648563	48.342419	0.001046	0.075847	48.623127	0.000659	0.396935	48.130431	0.001282	0.257779	47.829379	0.000488	0.177192	47.566589
	128	100	0.002773	0.319344	0.000700	0.354902	0.002246	0.945129	14.561668	0.003371	0.078844	14.708544	0.003432	0.237118	14.312595	0.002967	0.138726	14.226897	0.001848	0.196902	14.220935
		200	0.001499	0.333773	0.000452	0.359176	0.001154	0.956758	22.970229	0.001953	0.063175	22.906993	0.002021	0.305695	22.642105	0.001281	0.145933	22.294285	0.001086	0.197302	22.086332
LSTM100>GRU60	32	100	0.001731	0.436369	0.000365	0.528133	0.002021	1.362545	26.717938	0.001675	0.068337	26.722726	0.001755	0.361042	25.878032	0.002054	0.205382	26.575605	0.001148	0.184539	26.610204
		200	0.001025	0.446004	0.000283	0.537054	0.001333	1.385260	44.031852	0.000871	0.075741	43.898955	0.000997	0.387700	43.652092	0.001271	0.208155	43.691954	0.000651	0.173164	45.751490
	128	100	0.003515	0.333169	0.001705	0.360254	0.003087	0.965433	13.901506	0.003334	0.088702	13.943101	0.006452	0.277770	19.047223	0.002545	0.141498	19.031587	0.002157	0.192443	18.705069
		200	0.001437	0.385447	0.000617	0.413311	0.002468	1.105397	26.885393	0.001333	0.077031	27.120423	0.001162	0.340097	21.087164	0.001395	0.221743	20.978885	0.000826	0.182965	21.240056
LSTM100>LSTM100	32	100	0.001600	0.414100	0.000317	0.473323	0.002078	1.240853	31.508606	0.001605	0.079372	32.059074	0.001313	0.361596	31.992610	0.001692	0.219150	32.073233	0.001313	0.169532	32.025679
		200	0.000851	0.443182	0.000184	0.531938	0.000963	1.372392	49.515188	0.001027	0.058677	49.351570	0.000841	0.379426	49.403243	0.000876	0.223147	49.666124	0.000550	0.182268	49.418620
	128	100	0.002572	0.280945	0.000663	0.254216	0.002962	0.722570	14.366473	0.003305	0.092222	13.558677	0.002781	0.261393	15.314526	0.002174	0.153170	15.309305	0.001640	0.175371	15.527436
		200	0.001140	0.332925	0.000391	0.347138	0.000920	0.928875	22.805526	0.000877	0.064042	23.029936	0.001821	0.331287	23.970326	0.001115	0.141842	23.961802	0.000965	0.198576	23.785464
LSTM100>LSTM60	32	100	0.001802	0.414268	0.000584	0.507737	0.002155	1.307389	26.151449	0.001401	0.077240	26.178561	0.002023	0.334756	26.187693	0.002429	0.171006	26.179543	0.001003	0.180950	26.189275
		200	0.000804	0.469150	0.000150	0.607309	0.000887	1.540748	43.443549	0.000843	0.076898	43.521589	0.000881	0.354764	43.536504	0.000872	0.201515	43.436960	0.000537	0.171824	43.450649
	128	100	0.002296	0.271617	0.000551	0.239281	0.002501	0.677344	14.212787	0.002719	0.084235	14.272785	0.002306	0.293322	14.379891	0.002607	0.149246	14.386637	0.001350	0.153936	14.015416
		200	0.001228	0.339765	0.000148	0.352594	0.001369	0.955880	19.499313	0.001182	0.093647	21.816832	0.001358	0.302041	21.832557	0.001227	0.157916	21.812775	0.001006	0.189341	21.804487
LSTM150	32	100	0.001639	0.441194	0.000284	0.543442	0.001790	1.403623	21.392932	0.001328	0.107142	21.517107	0.001683	0.319147	21.448915	0.001386	0.201763	21.433134	0.002009	0.174294	21.533452
		200	0.000995	0.503400	0.000216	0.605580	0.001331	1.568905	42.570670	0.000864	0.107578	42.595655	0.000952	0.402196	42.483965	0.001061	0.258582	42.587707	0.000770	0.179739	42.942456
	128	100	0.002574	0.299442	0.000647	0.264279	0.002747	0.753865	10.213560	0.003263	0.106563	10.356351	0.002782	0.301066	10.369060	0.002566	0.145069	10.363748	0.001513	0.190646	10.307917
		200	0.001355	0.428717	0.000447	0.542245	0.002114	1.389087	16.441649	0.000926	0.101385	16.595471	0.001264	0.306533	16.584474	0.001283	0.148324	16.590285	0.001190	0.198257	16.460890
LSTM150>GRU100	32	100	0.001748	0.438716	0.000303	0.534212	0.001979	1.377063	40.828888	0.001281	0.077315	40.302039	0.001621	0.356138	39.581084	0.002009	0.218037	32.852293	0.001850	0.165027	32.889833
		200	0.000763	0.532812	0.000359	0.660282	0.000561	1.689836	58.148573	0.000626	0.059039	58.157450	0.000701	0.417616	57.896429	0.001395	0.304067	58.214585	0.000531	0.193502	57.863487
	128	100	0.002420	0.296087	0.000249	0.299474	0.002637	0.813782	16.602362	0.002558	0.069184	16.289892	0.002601	0.282834	16.010991	0.002204	0.140631	16.300605	0.002100	0.174006	16.409103
		200	0.001402	0.354267	0.000314	0.396185	0.001441	1.044338	26.898659	0.001406	0.059689	26.959139	0.001336	0.313291	26.900946	0.001855	0.163648	26.860715	0.000974	0.190370	26.814537
LSTM150>GRU60	32	100	0.002098	0.417556	0.000901	0.483529	0.002749	1.264303	29.920656	0.001634	0.073102	29.788989	0.001443	0.342281	29.491610	0.003350	0.240999	29.953946	0.001317	0.167093	29.934065
		200	0.000928	0.468321	0.000362	0.512754	0.000582	1.349898	52.995931	0.000950	0.063527	53.130511	0.001410	0.429671	52.594143	0.001131	0.330366	53.248337	0.000567	0.168146	53.173333
	128	100	0.003098	0.328575	0.001457	0.341531	0.002613	0.920289	16.509249	0.003294	0.077113	16.530568	0.003089	0.313887	15.406065	0.005267	0.164532	16.425735	0.001225	0.167055	16.340969
		200	0.001138	0.393558	0.000277	0.450195	0.001259	1.184195	24.721957	0.000971	0.075147	24.789179	0.001260	0.316452	24.650907	0.001449	0.206059	31.882144	0.000748	0.185935	32.056423
LSTM150>LSTM100	32	100	0.001380	0.399546	0.000311	0.477378	0.001213	1.231410	32.683689	0.001894	0.062464	32.839359	0.001206	0.361960	33.175709	0.001451	0.168806	33.158059	0.001136	0.173092	33.148359
		200	0.001068	0.466421	0.000511	0.563561	0.001944	1.444647	58.940870	0.000630	0.060981	58.785546	0.000996	0.436701	59.016765	0.000967	0.214291	59.134339	0.000803	0.175483	58.993893
	128	100	0.002150	0.276615	0.000540	0.251289	0.002210	0.707193	16.826800	0.002386	0.090546	16.779255	0.002508	0.284394	17.288812	0.002441	0.132039	17.052438	0.001204	0.168901	17.212528
		200	0.001073	0.372693	0.000347	0.402928	0.000762	1.062740	26.752774	0.001572	0.072048	26.825718	0.001096	0.386752	25.604161	0.001205	0.145841	26.861397	0.000731	0.196084	26.773708
LSTM150>LSTM60	32	100	0.001732	0.472971	0.000146	0.583257	0.001667	1.493873	29.859087	0.001837	0.069085	29.101182	0.001654	0.404881	30.482751	0.001573	0.191570	30.397942	0.001930	0.205447	30.421718
		200	0.000941	0.511565	0.000289	0.674152	0.000626	1.701719	52.996831	0.000858	0.075369	52.894647	0.000756	0.368453	53.479373	0.001124	0.251658	53.416921	0.001341	0.160627	53.350636
	128	100	0.002100	0.301267	0.000313	0.289664	0.001863	0.803052	15.697182	0.002227	0.088023	15.597249	0.002523	0.286109	20.699589	0.002159	0.145069	20.499166	0.001730	0.184081	20.578955
		200	0.001232	0.339954	0.000198	0.365039	0.001450	0.975658	29.256058	0.001437	0.076992	29.021192	0.001052	0.306344	24.511156	0.001169	0.146828	24.628599	0.001054	0.193946	23.896556

APPENDIX F: Query Scripts

Hive query script:

```
DROP VIEW IF EXISTS airports_coords_split;
CREATE VIEW airports_coords_split AS
SELECT
    iso_country, gps_code, iata_code,
    CAST(split(coordinates, ',')[0] AS FLOAT) AS longitude,
    CAST(split(coordinates, ',')[1] AS FLOAT) AS latitude
FROM airports_orc;

SELECT
    f.*, dep_airport.iso_country AS dep_country,
    dep_airport.latitude AS dep_lat, dep_airport.longitude AS dep_lon,
    arr_airport.iso_country AS arr_country,
    arr_airport.latitude AS arr_lat, arr_airport.longitude AS arr_lon
FROM flights_orc f
LEFT JOIN airports_coords_split dep_airport ON f.estdepartureairport =
dep_airport.gps_code
LEFT JOIN airports_coords_split arr_airport ON f.estarrivalairport =
arr_airport.gps_code;
```

Execution:

```
time hive -f query.hive
```

Result:

```
real  7m55.694s
user  0m57.110s
sys  0m8.767s
```

Impala query script:

```
DROP VIEW IF EXISTS airports_coords_split;
CREATE VIEW airports_coords_split AS
SELECT
    iso_country, gps_code, iata_code,
    CAST(SPLIT_PART(coordinates, ',', 1)) AS longitude,
    CAST(SPLIT_PART(coordinates, ',', 2)) AS latitude
FROM airports_orc;

INVALIDATE METADATA airports_coords_split;

SELECT
    f.*, dep_airport.iso_country AS dep_country,
    dep_airport.latitude AS dep_lat, dep_airport.longitude AS dep_lon,
    arr_airport.iso_country AS arr_country,
    arr_airport.latitude AS arr_lat, arr_airport.longitude AS arr_lon
FROM flights_orc f
LEFT JOIN airports_coords_split dep_airport ON f.estdepartureairport =
dep_airport.gps_code
```

```
LEFT JOIN airports_coords_split arr_airport ON f.estarrivalairport =
arr_airport.gps_code;
```

Execution:

```
time impala-shell -f query.impala
```

Result:

```
real  7m3.024s
user  1m2.308s
sys   0m4.116s
```

APPENDIX G: Schemas and Table Descriptions

```
[hadoop-29:21000] > SHOW TABLES;
+---------------------------+
| name                      |
+---------------------------+
| acas_data4                |
| allcall_replies_data4     |
| flarm_raw                 |
| flights                   |
| flights_data4             |
| identification_data4      |
| operational_status_data4  |
| position_data4            |
| rollcall_replies_data4    |
| sensor_visibility         |
| sensor_visibility_data3   |
| state_vectors             |
| state_vectors_data3       |
| state_vectors_data4       |
| velocity_data4            |
+---------------------------+
```

Figure A: OpenSky Table Schema

```
[hadoop-29:21000] > DESCRIBE flights_data4;
+-----------------------------------+-------------------+-----------------------------+
| name                              | type              | comment                     |
+-----------------------------------+-------------------+-----------------------------+
| icao24                            | string            | Inferred from Parquet file. |
| firstseen                         | int               | Inferred from Parquet file. |
| estdepartureairport               | string            | Inferred from Parquet file. |
| lastseen                          | int               | Inferred from Parquet file. |
| estarrivalairport                 | string            | Inferred from Parquet file. |
| callsign                          | string            | Inferred from Parquet file. |
| track                             | array<struct<     | Inferred from Parquet file. |
|                                   |   time:int,       |                             |
|                                   |   latitude:double,|                             |
|                                   |   longitude:double,|                            |
|                                   |   altitude:double,|                             |
|                                   |   heading:float,  |                             |
|                                   |   onground:boolean|                             |
|                                   | >>                |                             |
| serials                           | array<int>        | Inferred from Parquet file. |
| estdepartureairporthorizdistance  | int               | Inferred from Parquet file. |
| estdepartureairportvertdistance   | int               | Inferred from Parquet file. |
| estarrivalairporthorizdistance    | int               | Inferred from Parquet file. |
| estarrivalairportvertdistance     | int               | Inferred from Parquet file. |
| departureairportcandidatescount   | int               | Inferred from Parquet file. |
| arrivalairportcandidatescount     | int               | Inferred from Parquet file. |
| otherdepartureairportcandidates   | array<struct<     | Inferred from Parquet file. |
|                                   |   icao:string,    |                             |
|                                   |   horizdistance:int,|                           |
|                                   |   vertdistance:int|                             |
|                                   | >>                |                             |
| otherarrivalairportcandidates     | array<struct<     | Inferred from Parquet file. |
|                                   |   icao:string,    |                             |
|                                   |   horizdistance:int,|                           |
|                                   |   vertdistance:int|                             |
|                                   | >>                |                             |
| day                               | int               |                             |
+-----------------------------------+-------------------+-----------------------------+
```

Figure B: OpenSky flight_data4 Table Description

```
[hadoop-29:21000] > SELECT icao24, firstseen, estdepartureairport, lastseen, estarrivalairport, day FROM flights_data4 LIMIT 5;
+--------+------------+----------------------+------------+--------------------+------------+
| icao24 | firstseen  | estdepartureairport  | lastseen   | estarrivalairport  | day        |
+--------+------------+----------------------+------------+--------------------+------------+
| a05b7e | 1643221445 | KSFO                 | 1643239615 | NULL               | 1643155200 |
| a1180c | 1643225621 | NM89                 | 1643226293 | NULL               | 1643155200 |
| a26a6d | 1643240920 | 18JY                 | 1643241565 | 0WA2               | 1643155200 |
| a2a18a | 1643208611 | KFTW                 | 1643216331 | IA47               | 1643155200 |
| a4491c | 1643218439 | KLAX                 | 1643220562 | NULL               | 1643155200 |
+--------+------------+----------------------+------------+--------------------+------------+
```

Figure C: OpenSky flight_data4 Top 5 Rows

APPENDIX H: Data Management Plan

1. Overview

Researcher: Mary Dannoun
Project title: "Airspace in Conflict: An Analysis of Historical Flight Patterns and the Impact of War in Ukraine and Its Surrounding Regions"
Project duration: 3 months
Project context: The discipline of the research is Aerospace Studies, specifically focusing on Aviation Analytics. The subject of the research is an investigation into the historical patterns of flight departures and arrivals in Ukraine and its neighboring regions, with an emphasis on understanding how the conflict within Ukraine has influenced these patterns.

2. Defining Data/Research Sources

2.1 Where will your data/research sources come from? Data sources for this research include: - DB1: OpenSky Network: For historical global flight data. - DB2: Aviation Edge API: To supplement recent data on flights, particularly in and around Ukraine.
2.2 How often will you get new data? Data acquisition for this project was conducted in two primary phases: - Initial Data Collection: Historical flight data from January 1, 2016, to July 23, 2023, was retrieved from the OpenSky Network database. - Ongoing Data Collection: Post July 23, 2023, data regarding specific countries was collected through the Aviation Edge API. This data was captured continuously to maintain an up-to-date dataset that reflects current conditions, with a particular focus on the impact of the conflict in Ukraine. The data from the OpenSky Network was a one-time historical retrieval, while the Aviation Edge API provided ongoing data collection. The frequency of data updates from the Aviation Edge API depended on the terms of service and limitations of the API, typically allowing for data updates on a daily basis. Over time, the need for new data may decrease as the project progresses beyond the data analysis phase and into the model evaluation and reporting phases.
2.3 How much data/information will you generate? The initial dataset, before any cleaning, was 17.4 GB. After cleaning and processing, which included filtering and selecting relevant data, the size was reduced to 8.9 GB. This reduction signifies the removal of extraneous information and the focus on data pertinent to the study's objectives surrounding Ukraine's flight patterns and the impacts of conflict.
2.4 What file formats will you use? File Formats:

- ORC (Optimized Row Columnar) for processed data storage, ensuring efficiency in storage and query performance.
- CSV (Comma-Separated Values) for initial data collection and intermediate data processing steps.

Software Required:
- Apache Hadoop for managing and querying data stored in ORC format.
- Apache Hive and Apache Impala for data querying and analysis.
- General-purpose software like Microsoft Excel or programming languages like Python (with libraries like Pandas) for accessing and manipulating CSV files.

Data Types:
- Our data includes Numerical and Text data, stored in a structured table under Hive.

3. Organizing Data

How will you structure and name your folders and files?

Folder and Files in hdfs:

Flight data: /flight_data/* for all flight data, with replication factor of 2, named accordingly.

Results: /airports/* for the airports dataset, which will later be merged with flight_data in query.

4. Looking After Data

4.1 Where will you store your data?

The data will be stored on a cloud-based Hadoop cluster for scalability and accessibility. Additional copies will reside on network storage for ease of access and collaboration.

4.2 How will your data be backed up?

Hadoop replicates data over 4 servers with a replication factor of 2.
Snapshots of the VMs are stored on the main host utilizing ESXI VMware.
Update Frequency: Backups will be updated weekly to ensure data integrity and minimize loss by the cloud service provider.

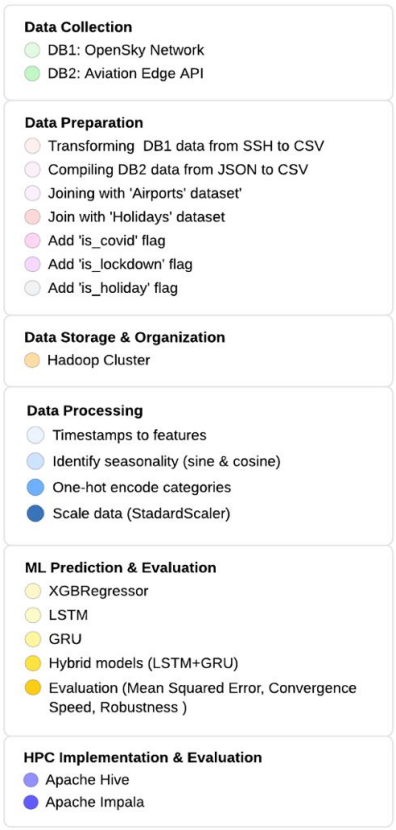


Figure 18: DMP Structure

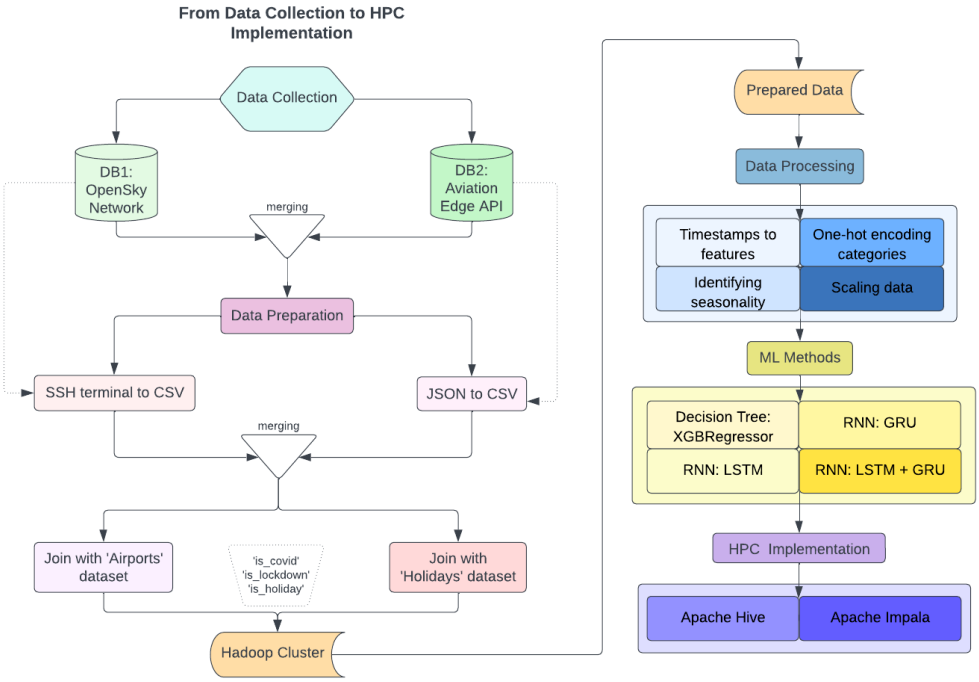


Figure 19: DMP Flowchart

APPENDIX I: Source Code

Badami, M.H. and Dannoun, M. (2023). Github: Source Code. https://github.com/hesbadami/dda.