

7153CEM BIG DATA ANALYTICS AND DATA VISUALISATION

**Understanding and Combating Global Terrorism: A Big Data
Approach**

AGBOOLA OLAYINKA KAZEEM

13412302

ABSTRACT

The goal of the project is to use big data analytics to better understand international terrorism and predict the success of an attack. The Global Terrorism Dataset is a thorough compilation of more than 180,000 terrorist incidents globally, was used in the study. Despite the dataset's richness, its high dimensionality was a considerable obstacle, leading to a shift in emphasis to a small number of important selected features. Data cleaning, preprocessing, and exploratory data analysis were applied to these attributes, laying the groundwork for later modelling. The study found that machine learning models, such as logistic regression, support vector machine, naïve Bayes, and random forest, can be used to predict the success of terrorist attacks. The study also found that feature selection is important when working with high-dimensional data

Keywords: Global terrorism dataset, logistic regression, support vector machine, naïve Bayes, random forest, feature selection

CONTENTS

Introduction.....	4
Background/Related Work.....	4
Dataset Description.....	5
Methodology.....	6
Logistic Regression.....	6
Support Vector Machine (SVM).....	6
Random Forest.....	6
Naïve Bayes.....	6
Software Installation.....	6
Experimental Section.....	7
Tableau Visualization.....	9
Result Discussion.....	10
Conclusion.....	11
Future Works.....	12
Social and Ethical Impact	12
References	12
Appendix.....	13

1. INTRODUCTION

Terrorism, a violent act typically perpetrated for political, intellectual, or religious purposes, continues to pose a severe danger to world peace and security. Daily routines are disrupted, worry and uncertainty are increased, and it frequently ends with the death of innocent lives. As a result, it is necessary for academics, decision-makers, and security agencies everywhere to comprehend the dynamics of international terrorism, particularly the patterns, reasons, and goals that underlie such deeds.

Data-driven insights are becoming more and more crucial in the digital era for comprehending complex and varied phenomena like terrorism. The success of this project, "Understanding and Combating Global Terrorism: A Big Data Approach," rests on this data-driven methodology.

To better understand terrorism episodes, patterns, and outcomes, a comprehensive global terrorism dataset will be analyzed using big data analytics.

The work uses machine learning techniques including Logistic Regression, Random Forest, and Support Vector Machines to model and predict the success of terrorist strikes. Area Under the ROC Curve (AUC-ROC) is used to evaluate how well these models perform at making reliable and accurate predictions.

Due to how delicate the topic is, the project is conscious of ethical concerns. It complies with data protection regulations to guarantee responsible data usage, observance of private rights, and preservation of the dignity of persons impacted by the incidents. We must take all necessary precautions to prevent any misuse of the information obtained from our research and to take into account how it can harm individuals and society.

2. Background/ Related Work

Data-Driven Approaches to Terrorism Analysis:

The surge of big data and computational techniques has introduced new paradigms in the domain of counter-terrorism. Researchers are increasingly shifting from traditional qualitative methodologies to data-driven approaches to extract patterns, trends, and actionable insights from vast amounts of unstructured data (Chen et al., 2008).

Machine Learning in Counter-Terrorism:

Various researchers have exploited machine learning algorithms to analyze the GTD. For instance, studies have employed clustering techniques to discern patterns and groups in terrorism data (Wang et al., 2012). Logistic regression has been instrumental in understanding the factors influencing the success or failure of terrorist attacks (Enders & Sandler, 2000). Decision tree algorithms and random forests have shown promise in predicting locations susceptible to future attacks (Li & Schuurman, 2018).

Support Vector Machines and Terrorism Data:

There have been limited studies specifically using SVM on terrorism data. However, recent initiatives show its potential in classifying large-scale versus small-scale attacks (Sharma & Panigrahi, 2013).

Naïve Bayes and Textual Analysis:

A notable trend is the use of Naïve Bayes in analyzing textual data related to terrorism—be it propaganda material, online forums, or news articles. Its efficiency in classifying and predicting sentiments in textual data has been documented (Sebastiani, 2002).

Limitations and Challenges:

The major limitation in the domain of data-driven terrorism analysis is the high dimensionality and, often, the incompleteness of the datasets. This complicates the task of feature selection and warrants advanced methods to streamline data for efficient machine learning application (Agarwal & Sureka, 2015).

Moreover, the real-world implication of misclassification—especially when predicting the success of a terrorist attack—is of grave concern. It accentuates the need for models with heightened accuracy and robustness (Agarwal & Sureka, 2015).

3. DATASET DESCRIPTION

Information on terrorist attacks worldwide is compiled in the Global Terrorism Database (GTD), a sizable open-source database. The University of Maryland's National Consortium for the Study of Terrorism and Responses to Terrorism (START) is in charge of it.

Since the early 1970s, the GTD has compiled information on more than 180,000 terrorist incidents. It is one of the best datasets on terrorism and covering more than 45 years. The dataset contains incidences from all across the world and takes both domestic and foreign terrorism into consideration. The database's events each have more than 135 characteristics attached to them, providing comprehensive information on every facet of the attack. Some of the columns are

Date and location of the incident.

Attack type (for example: bombing, armed assault, hijacking).

Target type (for example: government, military, civilians).

Names of the perpetrators

Number of casualties (fatalities and injuries).

Whether the attack was successful.

The motives and any claims of responsibility.

Weapon types

Some of the data types present in the dataset are

Integer	EventId, targettype, natlty1
String	Country_txt, Region_tx
Float	Latitude,Longitude
Dates	Year,Month,Day

4. METHODOLOGY

The source of my dataset is Kaggle (<https://www.kaggle.com/datasets/START-UMD/gtd>).

The following are the methods I used for this project

4.1 Logistic Regression:

Logistic Regression is a statistical method used for analyzing datasets where one or more independent variables determine an outcome. The outcome is measured with a dichotomous variable, meaning there are only two possible outcomes. It is employed to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables (Hosmer, Lemeshow, & Sturdivant, 2013).

4.2 Support Vector Machine (SVM):

Support Vector Machine (SVM) is a supervised machine learning algorithm suitable for both classification or regression challenges. It functions by classifying data by pinpointing the hyperplane that optimally divides a dataset into classes (Cortes & Vapnik, 1995).

4.3 Random Forest:

Random Forest is an ensemble learning method usable for both regression and classification tasks. It builds a 'forest' of decision trees, with each tree trained on a random subset of the data and features, and combines their outputs to enhance accuracy and control overfitting (Breiman, 2001).

4.4 Naïve Bayes:

Naïve Bayes is a probabilistic classification method based on Bayes' theorem and assumes independence between predictors. Especially effective for high-dimensional datasets, it is frequently employed for text classification tasks (Rish, 2001).

4.5 Software Installation

As required by the coursework, the installation of the Java, Hadoop and Spark was done on Ubuntu after installing a virtual machine on my computer. The java software was installed using the 'sudo apt install' command, I also got the required versions of Hadoop and spark. The process followed were outlined in the lab session documents, the evidence of the installations can be seen in figures 1,2a,2b and 3

5. EXPERIMENTAL SECTION

For the implementation, I initiated the setup by installing a version of PySpark on Google Colab and the findspark library. The purpose of findspark was to facilitate the initialization of the Spark environment within Colab (Figure 4)

The next stage is to import all the requisite libraries, including those for machine learning from PySpark (as shown in Figure 5)

With the environment set up and the libraries loaded, the next step was dataset import. The dataset, `global_terrorism.csv`, was loaded into a Spark DataFrame (`df`), and also show the headers and first few rows (Figure 6).

I also checked the size and dimensionality of the dataset, this can be seen in figure 7

The data preprocessing stage starts by dropping multiple columns from the DataFrame `df`. The columns were removed because they were irrelevant or redundant to the project. The code also drops cells that are empty before proceeding to show the first 10 rows of the cleaned DataFrame (Figure 8)

An important step in handling high-dimensional data is the application of dimensionality reduction techniques. For this, Principal Component Analysis (PCA) was used. The figure 9 shows the application of Principal Component Analysis (PCA) using PySpark's machine learning library to the global terrorism dataset. It first creates a list of one-hot encoded feature columns. Using the `VectorAssembler`, these features are combined into a single vector column. PCA is then applied to reduce this high-dimensional data into 13 principal components. The transformed data, now represented by these principal components, is displayed, aiding in dimensionality reduction, potential noise reduction, and possibly improving machine learning model performance

Following PCA, the figure 10 shows a feature selection process as part of the data preprocessing where only a subset of columns from the original dataset is chosen based on their relevance or potential significance to the project. By selecting these columns, I aim to improve the efficiency and interpretability of the machine learning model for the project. Reducing the dimensionality can lead to faster model training, better generalization to new data, and clearer insights into the relationship between variables and the target outcome, 'success'.

The next stage is to preprocess the categorical columns in the dataset for machine learning algorithms. It first identifies the categorical columns that require transformation. Using `StringIndexer`, each unique string value of these columns is mapped to a unique numerical index, making them useable or compatible for the machine learning models. Subsequently, `OneHotEncoder` converts these numeric indices into binary vectors, ensuring the model doesn't misunderstand the numeric indices as having ordinal significance. A pipeline is then constructed to systematically apply the indexing and encoding processes. Then pipeline transforms the original dataset, producing a new DataFrame '`df_encoded`' with the processed columns added (Figure 11).

The next stage is the data splitting stage, I will be splitting the data in to training and testing sets. About 70% of the data will be allocated to training set while the remaining 30% goes to the testing set. The `seed=42` is a way to make sure that the splitting process will be split the same way anytime the data is run again with the same seed cunt. This brings in some level of consistency across different runs (Figure12).

The next stage is to start the Model Training and Performance evaluation using ROC AUC.

Figure 13 shows the start of a logistic regression model targeting the 'success' column using specified features. After training on `train_data`, the model predicts outcomes on `test_data`. To assess performance, an evaluator measures the Area Under the ROC curve (AUC) — a higher AUC indicates better performance. The output shows the performance of the `ROC_AUC`

The Support Vector Machine (SVM) model, specifying 'success' as the target variable and the set features for prediction. It then trains this SVM model on the `train_data`. Afterwards, it uses the trained model to predict outcomes on `test_data`. The performance of this model is quantified using the Area Under the ROC Curve (AUC), with a higher AUC signaling a more accurate model. Then the code displays both the predicted and actual labels for the initial ten entries in the `test_data` data frame (Figure 14)

Next is the initializing of a Random Forest classifier, still picking 'success' as the target variable, the designed features for prediction, and setting the number of trees in the forest to 100 because of the size of the dataset. It then trains this classifier on the train_data. After training, it employs the trained model to predict outcomes on test_data. The model's performance is measured using the Area Under the ROC Curve (AUC), where a higher value signifies better model accuracy (Figure 15).

The final model is the Naive Bayes model, it first assembles one-hot encoded features into a single vector column for both training and testing data. After defining and training the Naive Bayes model on the training data, predictions are made on the test set. The model's performance is then assessed using the Area Under the ROC Curve (ROC AUC), and the ROC AUC score is printed (Figure 16).

6. TABLEAU VISUALIZATION

This visualization (Figure 17) shows the location of the top 20 terrorist groups, this was done by placing the country on the face of the worksheet, then the Gname which contains the names of all terrorist groups in the filter section, this enabled me to select the top 20 terrorist groups and Gname was also placed on the color tab so each of the groups will be represented by a distinct color as shown on the legend placed on the right side of the image.

Also to see for each of the top 20 terrorist groups the kind of weapons associated to them over the years (Figure 18), I have used the same filtered Gname and also placed the Gname on the rows section, the weapon type (weaptype1 txt) was placed in the color coding part giving rise to the color code on the legend to show for each of the groups, the kind of weapons that is expected to be seen anytime they initiate an attack.

To see the distribution of attacks from the dataset, I put the attack type in the color tab and the sum of targets as labels, with the aid of a pie chart (Figure 19), it can be observed that bombings/explosions make up the majority of the kinds of attacks experienced globally which is then closely followed by armed assaults.

To see the prevalence of the top 20 terrorist groups based on the types of attacks they perpetrate I used a stacked bar chart in descending order to present this in figure 20.

Figures 21, 22, 23 and 24 show for all the countries represented in the dataset, the kinds of weapon types that have been used at least once in each of the countries.

7. RESULT DISCUSSION

7.1 Logistic Regression:

A ROC AUC of 0.82 for the logistic regression model demonstrates its robustness in distinguishing successful from unsuccessful terrorist incidents. The success of this model could be attributed to the linear relationships it identified between the features and the target variable. In the complex world of global terrorism, where factors are interwoven, logistic regression was able to decipher substantial linear associations to make predictions. However, it must be noted as a linear model, it might have missed some non-linear intricacies.

7.2 Support Vector Machine:

With an AUC of 0.70, The SVM model shows moderate performance with an AUC of 0.70. It's notably lower than logistic regression, indicating that SVM might not be capturing the complexities or relationships within the dataset as effectively as logistic regression. This might suggest that the SVM, particularly the linear version applied here, struggled to capture the complexities inherent in the dataset. Terrorism data is multifaceted, with non-linear and hierarchical patterns that the SVM might have found challenging to unravel.

7.3 Random Forest:

Random Forest also presents a decent AUC value of 0.75, although not as high as logistic regression. Its ensemble nature, leveraging multiple decision trees, offers a more diversified approach to the problem, capturing non-linearity better than logistic regression but in this case not quite outperforming it. The model's inherent ability to capture variable importance can further provide insights into factors most influential in determining the success of terrorist incidents.

7.4 Naïve Bayes:

The Naïve Bayes returned the ROC AUC score of 0.32 which is notably low. This suggests that the model has poor predictive power in distinguishing between the classes of the target variable (in this case, 'success' of a terrorist attack). In practical terms, relying on this model to predict the outcome or success of a terrorist event would likely yield many incorrect predictions. The score is below 0.5, indicating the model is performing worse than random guessing for this particular dataset.

8. CONCLUSION

Based on the results, the logistic regression model is the most suitable choice for predicting outcomes on the global terrorism dataset given the current feature set and preprocessing. It's essential to note that while AUC provides a good measure of a model's ability to differentiate between classes across various threshold values, other metrics and practical considerations, such as interpretability, computational efficiency, and model efficiency, should also be taken into account when choosing a model for deployment.

Predicting the success of terrorist activity, high accuracy is crucial. Misclassification can have severe real-world implications. For instance, if an impending large-scale attack is deemed unsuccessful by the model but turns out otherwise, the consequences are dire. Therefore, while the AUC values suggest a good predictive power, it's essential to approach deployment with caution, considering the severe repercussions of false negatives.

The global terrorism dataset, with its intricacies and complexity, was subjected to three machine learning classifiers. The Logistics Regression classifier emerged as the most effective in predicting the success of a terrorist act, with a respectable AUC of 0.82. Meanwhile, Random Forest also delivered a commendable performance. However, the linear nature of the Support Vector Machine might have hindered its efficacy on this dataset.

The Naïve Bayes value of 0.32 indicates a subpar performance. This suggests that the dataset's features may not be entirely independent, challenging the core assumption of this method. While Naive Bayes is effective in areas like text classification, its application for this particular dataset seems less fitting. For future endeavors, there's potential in refining the model, leveraging domain expertise, or seeking alternative modeling strategies, as evident in it being the lowest AUC of the three models selected.

While the models' predictive performances provide actionable insights, it's equally important to appreciate the broader implications. Terrorism is a deeply sensitive topic with significant geopolitical, social, and ethical ramifications. Thus, while machine learning can offer valuable predictions, it should be complemented with expert analyses, contextual understanding, and ethical considerations.

9. FUTURE WORKS

For future works on the global terrorism dataset project, there's potential to add more recent data to discern emerging terrorism trends. Trying different advanced machine learning models, such as neural networks or gradient boosting, might enhance prediction accuracy. There's also an opportunity to refine the dataset through deeper feature engineering. Collaborating with geopolitical and social experts can also ensure predictions are contextually grounded. Moving forward, there's a vision for a real-time predictive system that can swiftly assess a terrorist act's likelihood of success. Emphasizing model interpretability will ensure clear decision-making, and given the fluid nature of terrorism, models should undergo regular updates to remain relevant.

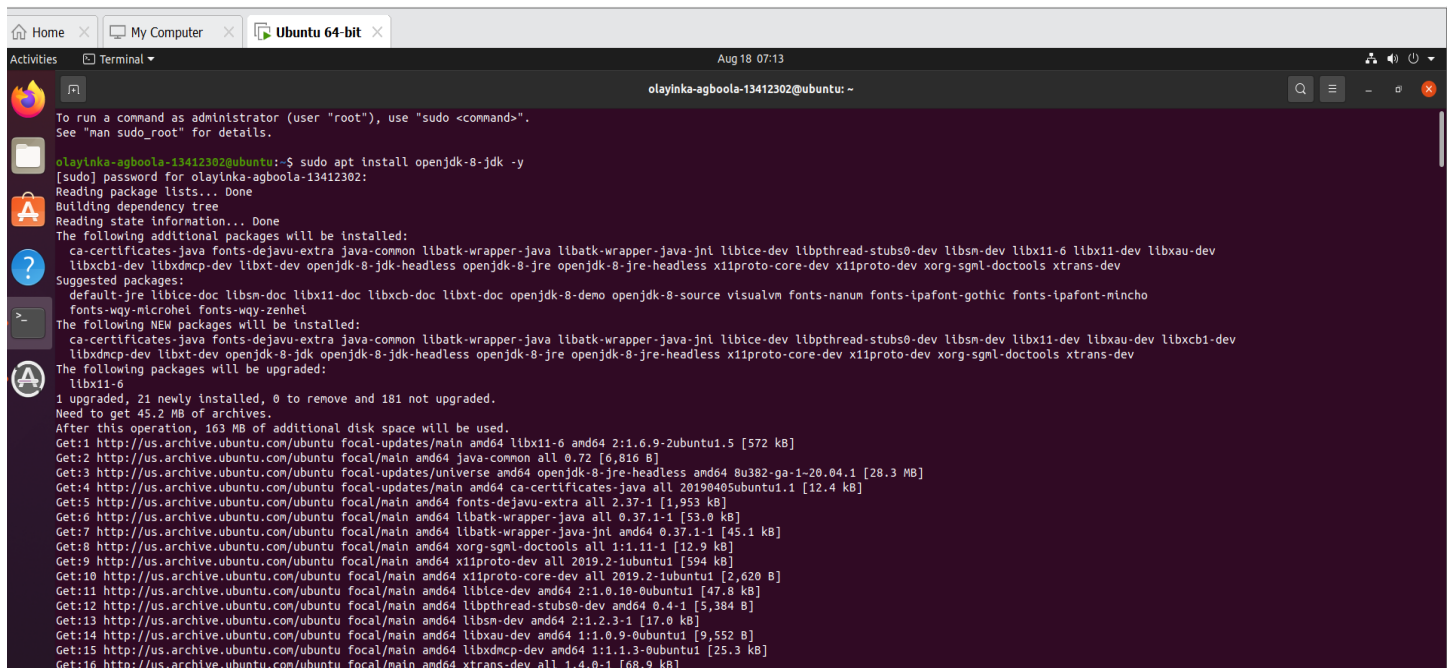
9. SOCIAL AND ETHICAL IMPACT

Utilizing machine learning to predict terrorist attacks requires navigating a series of ethical challenges. Privacy concerns arise when models access personal data, potentially enabling individual tracking and monitoring. Maintaining the confidentiality of this data is vital, and individuals must retain control over their personal information. These models, if not trained with diverse data, risk inherent biases, which can lead to undue discrimination, potentially stigmatizing specific demographics like white males as terrorism risks. Accountability is necessary, with clear procedures for recognizing and rectifying biases and ensuring non-discriminatory usage of these models. Transparency is equally critical, ensuring the workings of these models are comprehensible, building trust in their functionality. While machine learning can help pinpoint potential terror threats, leading to preventative measures, it's paramount to balance this with individuals' privacy and rights. The technology's capability for monitoring potentially leads to mass surveillance, risking the erosion of civil liberties. Striking a balance between security and privacy is imperative, as is preventing any undue discrimination through the models. Before implementing such systems, weighing all these ethical concerns and establishing checks against bias, discrimination, and unwarranted surveillance is essential.

References

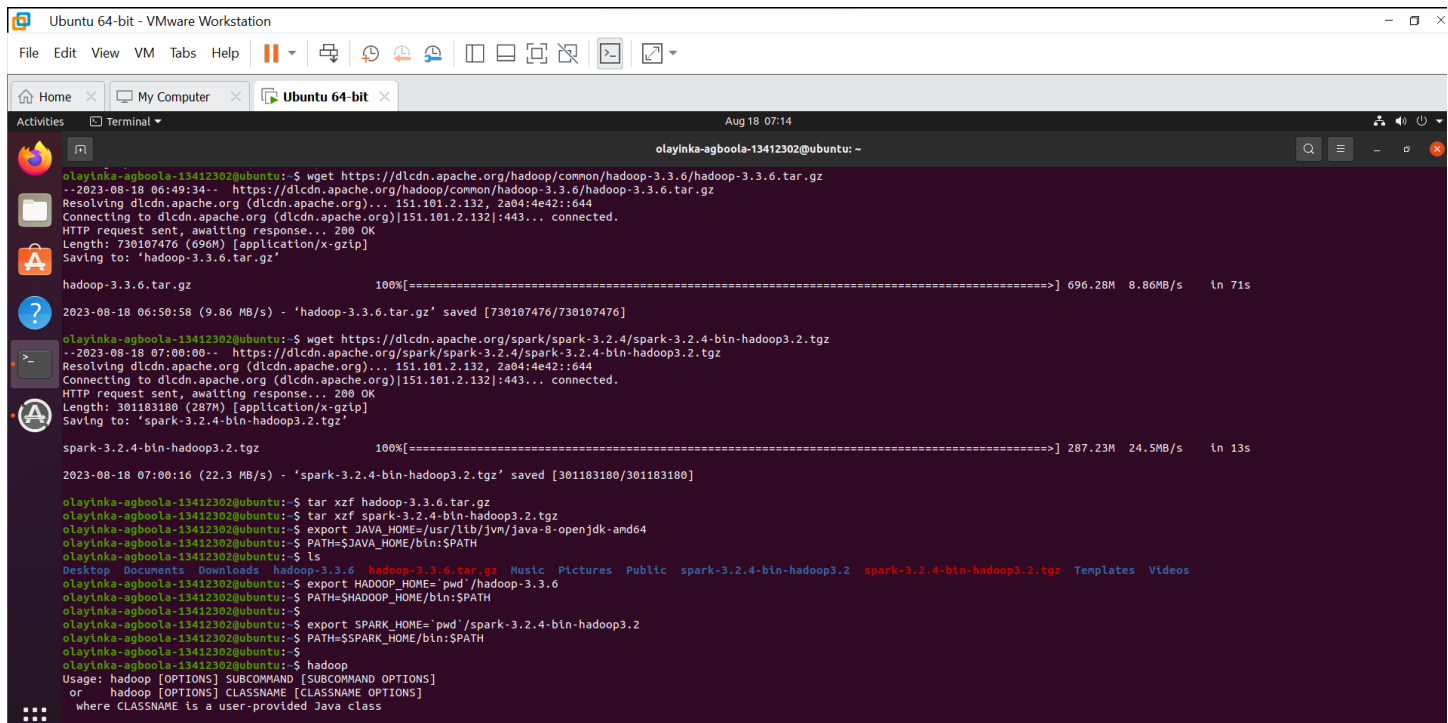
- Kaggle : <https://www.kaggle.com/datasets/START-UMD/gtd>
- Chen, H., Reid, E., Sinai, J., Silke, A., & Ganor, B. (2008). *Terrorism Informatics*. Springer.
- LaFree, G., & Dugan, L. (2007). Introducing the global terrorism database. *Terrorism and Political Violence*, 19(2), 181-204.
- Wang, X., Gerber, M. S., & Brown, D. E. (2012). Automatic crime prediction using events extracted from Twitter posts. In *Social Computing, Behavioral-Cultural Modeling and Prediction* (pp. 231-238). Springer.
- Enders, W., & Sandler, T. (2000). Is transnational terrorism becoming more threatening? A time-series investigation. *Journal of Conflict Resolution*, 44(3), 307-332.
- Li, Y., & Schuurman, N. (2018). A review of the applications of agent-based models in terrorism and counter-terrorism. *Terrorism and Political Violence*, 30(4), 574-590.
- Sharma, A., & Panigrahi, P. K. (2013). A detailed analysis on various types of attacks on IDS in MANET. *International Journal of Computer Applications*, 68(14).
- Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1), 1-47.
- Agarwal, P., & Sureka, A. (2015). Applying social media intelligence for predicting and identifying on-line radicalization and civil unrest oriented threats. *ArXiv Preprint ArXiv:1511.06858*.
- Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (Vol. 398). John Wiley & Sons.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273-297.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.
- Rish, I. (2001). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46). IBM.

Appendix



```
olayinka-agboola-13412302@ubuntu: ~$ sudo apt install openjdk-8-jdk -y
[sudo] password for olayinka-agboola-13412302:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev libx11-6 libx11-dev libxau-dev
  libxcb1-dev libxdmcp-dev libxt-dev openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless x11proto-core-dev x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  default-jre libice-doc libsm-doc libx11-doc libxcb-doc libxt-doc openjdk-8-demo openjdk-8-source visualvm fonts-nanum fonts-ipafont-gothic fonts-ipafont-mincho
  fonts-wqy-microhe font-wqy-zenhei
The following NEW packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev
  libxdmcp-dev libxt-dev openjdk-8-jdk openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless x11proto-core-dev x11proto-dev xorg-sgml-doctools xtrans-dev
The following packages will be upgraded:
  libx11-6
1 upgraded, 21 newly installed, 0 to remove and 181 not upgraded.
Need to get 45.2 MB of archives.
After this operation, 163 MB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 libx11-6 amd64 2:1.6.9-2ubuntu1.5 [572 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu focal/main amd64 java-common all 0.72 [6,816 B]
Get:3 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 openjdk-8-jre-headless amd64 8u382-ga-1-20.04.1 [28.3 MB]
Get:4 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 ca-certificates-java all 20190405ubuntu1.1 [12.4 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu focal/main amd64 fonts-dejavu-extra all 2.37-1 [1,953 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libatk-wrapper-java all 0.37.1-1 [53.0 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libatk-wrapper-java-jni amd64 0.37.1-1 [45.1 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu focal/main amd64 xorg-sgml-doctools all 1:1.11-1 [12.9 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu focal/main amd64 x11proto-dev all 2019.2-1ubuntu1 [594 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu focal/main amd64 x11proto-core-dev all 2019.2-1ubuntu1 [2,620 B]
Get:11 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libice-dev amd64 2:1.0.10-0ubuntu1 [47.8 kB]
Get:12 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libpthread-stubs0-dev amd64 0.4-1 [5,384 B]
Get:13 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libsm-dev amd64 2:1.2.3-1 [17.0 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libxau-dev amd64 1:1.0.9-0ubuntu1 [9,552 B]
Get:15 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libxdmcp-dev amd64 1:1.1.3-0ubuntu1 [25.3 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu focal/main amd64 xtrans-dev all 1.4.0-1 [68.9 kB]
```

Figure 1: Installation of Java



```
olayinka-agboola-13412302@ubuntu: ~$ wget https://d1cdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
--2023-08-18 06:49:34-- https://d1cdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving d1cdn.apache.org (d1cdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to d1cdn.apache.org (d1cdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: 'hadoop-3.3.6.tar.gz'

hadoop-3.3.6.tar.gz 100%[=====] 696.28M 8.86MB/s in 71s

2023-08-18 06:50:58 (9.86 MB/s) - 'hadoop-3.3.6.tar.gz' saved [730107476/730107476]

olayinka-agboola-13412302@ubuntu: ~$ wget https://d1cdn.apache.org/spark/spark-3.2.4/spark-3.2.4-bin-hadoop3.2.tgz
--2023-08-18 07:00:00-- https://d1cdn.apache.org/spark/spark-3.2.4/spark-3.2.4-bin-hadoop3.2.tgz
Resolving d1cdn.apache.org (d1cdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to d1cdn.apache.org (d1cdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 301183180 (287M) [application/x-gzip]
Saving to: 'spark-3.2.4-bin-hadoop3.2.tgz'

spark-3.2.4-bin-hadoop3.2.tgz 100%[=====] 287.23M 24.5MB/s in 13s

2023-08-18 07:00:16 (22.3 MB/s) - 'spark-3.2.4-bin-hadoop3.2.tgz' saved [301183180/301183180]

olayinka-agboola-13412302@ubuntu: ~$ tar xzf hadoop-3.3.6.tar.gz
olayinka-agboola-13412302@ubuntu: ~$ tar xzf spark-3.2.4-bin-hadoop3.2.tgz
olayinka-agboola-13412302@ubuntu: ~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
olayinka-agboola-13412302@ubuntu: ~$ PATH=$JAVA_HOME/bin:$PATH
olayinka-agboola-13412302@ubuntu: ~$
olayinka-agboola-13412302@ubuntu: ~$ export HADOOP_HOME='pwd' /hadoop-3.3.6
olayinka-agboola-13412302@ubuntu: ~$ PATH=$HADOOP_HOME/bin:$PATH
olayinka-agboola-13412302@ubuntu: ~$
olayinka-agboola-13412302@ubuntu: ~$ export SPARK_HOME='pwd' /spark-3.2.4-bin-hadoop3.2
olayinka-agboola-13412302@ubuntu: ~$ PATH=$SPARK_HOME/bin:$PATH
olayinka-agboola-13412302@ubuntu: ~$
olayinka-agboola-13412302@ubuntu: ~$ hadoop
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
or hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
where CLASSNAME is a user-provided Java class
```

Figure 2a: Installation of Hadoop and Spark


```
!pip install pyspark==3.0.0
!pip install findspark

WARNING: Skipping pyspark as it is not installed.
Collecting pyspark==3.0.0
  Downloading pyspark-3.0.0.tar.gz (204.7 MB)
    Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9 (from pyspark==3.0.0)
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
    Building wheels for collected packages: pyspark
    Building wheel for pyspark (setup.py) ... done
    Created wheel for pyspark: filename=pyspark-3.0.0-py2.py3-none-any.whl size=205044164 sha256=2842f911
    Stored in directory: /root/.cache/pip/wheels/b1/bb/8b/ca24d3f756f2ed967225b0871898869db676eb5846df5ac
Successfully built pyspark
Installing collected packages: py4j, pyspark
  Attempting uninstall: py4j
    Found existing installation: py4j 0.10.9.7
    Uninstalling py4j-0.10.9.7:
      Successfully uninstalled py4j-0.10.9.7
Successfully installed py4j-0.10.9 pyspark-3.0.0
```

Figure 4: Installation of Pyspark and Findspark on Colab

```
[ ] import findspark
findspark.init()

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("global_terrorism").getOrCreate()

import pyspark.sql.functions as F
from pyspark.mllib.stat import Statistics
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

Figure 5: Importing of libraries to be used

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("global_terrorism").getOrCreate()

csv_path = '/content/sample_data/global_terrorism.csv'

df = spark.read.csv(csv_path, header=True, inferSchema=True)
df.head()
df.show(10)
```

eventid	year	month	day	approxdate	extended	resolution	country	country_txt	region	region_txt	provstate	city	latitude	longit

Figure 6: Importing the dataset, checking the column names


```
+ Code + Text

#to check number of rows
df.count()
print ('The number of rows:', df.count())

#To check number of columns
len(df.columns)
print ('The number of columns:', len(df.columns))

The number of rows: 181691
The number of columns: 135
```

Figure 7: Checking the number of rows and columns

```
Code + Text

#Data Preprocessing

import pyspark.sql.functions as F
from pyspark.mllib.stat import Statistics

# Drop the columns
df = df.drop('approxdate', 'resolution', 'attacktype3', 'target2', 'corp2')
df = df.drop('ransomamt', 'ransompaid', 'ransompaidus', 'ransomamtus')
df = df.drop('ransomnote', 'nreleased', 'hostkidoutcome', 'hostkidoutcome_txt')
df = df.drop('attacktype3_txt', 'targtype', 'targsubtype', 'natlty2', 'natlty3')
df = df.drop('gname2', 'gsubname3', 'claimmode', 'claimmode2_txt', 'nhours')
df = df.drop('natlty3_txt', 'gsubname', 'claimmode_txt', 'weaptype2', 'ndays')

# Drop rows with empty cells
df = df.na.drop()

# Show the first 10 rows of the DataFrame after dropping missing data
df.show(10)

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| eventid | iyear | imonth | iday | country | country_txt | region | region_txt | p |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1.97E11 | 1970 | 7 | 2 | 58 | Dominican Republic | 2 | Central America &... |
```

Figure 8: Data Preprocessing

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

RAM ☐ Disk ☐

```
+ Code + Text

from pyspark.ml.feature import PCA, VectorAssembler

# List of encoded features
feature_columns = [column+"_ohe" for column in categorical_columns]

# Assembling encoded features into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features_vector")
df_assembled = assembler.transform(df_encoded)

# Applying PCA
num_principal_components = 13
pca = PCA(k=num_principal_components, inputCol="features_vector", outputCol="pca_features")
pca_model = pca.fit(df_assembled)
df_pca = pca_model.transform(df_assembled)

# Show transformed data
df_pca.select("pca_features").show()

+-----+
|      pca_features |
+-----+
| [-1.2112337161875... |
| [-1.5188752712147... |
```

Figure 9: Principal Component Analysis

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] # Select the columns
selected_columns = ["iyear", "imonth", "country_txt", "region_txt", "city", "attacktype1_txt",
                    "targettype1_txt", "targetsubtype1_txt", "target1", "natlty1_txt", "gname", "motive", "success"]

df = df.select(selected_columns)
df.show()
```

iyear	imonth	country_txt	region_txt	city	attacktype1_txt	targettype1_txt	targetsubtype1_txt	target1	natlty1_txt
1970	1	United States	North America	Cairo	Armed Assault	Police	Police Building (...)	Cairo Police Head...	United States
1970	1	United States	North America	Madison	Facility/Infrastr...	Military	Military Recruit...	R.O.T.C. offices ...	United States
1970	1	United States	North America	Madison	Facility/Infrastr...	Government (General)	Government Buildi...	Selective Service...	United States
1970	1	United States	North America	Rio Piedras	Facility/Infrastr...	Business	Retail/Grocery/Ba...	Baker's Store	United States
1970	1	United States	North America	New York City	Bombing/Explosion	Educational Insti...	School/University...	James Madison Hig...	United States
1970	1	United States	North America	Jersey City	Facility/Infrastr...	Violent Political...	Party Office/Faci...	Headquarters	United States
1970	1	United States	North America	New York City	Armed Assault	Police	Police Security F...	New York Police O...	United States
1970	1	United States	North America	West Point	Bombing/Explosion	Government (General)	Judge/Attorney/Court	Clay County Court...	United States
1970	2	United States	North America	Portland	Facility/Infrastr...	Government (General)	Government Buildi...	Selective Service...	United States
1970	2	United States	North America	Cairo	Armed Assault	Police	Police Building (...)	Illinois State Po...	United States
1970	2	United States	North America	Denver	Bombing/Explosion	Transportation	Bus (excluding to...	Denver City Schoo...	United States
1970	2	United States	North America	Seattle	Bombing/Explosion	Government (General)	Politician or Pol...	Fred Dore	United States
1970	2	United States	North America	Dorado	Facility/Infrastr...	Government (General)	Government Buildi...	Selective Service...	United States
1970	2	United States	North America	Carolina	Facility/Infrastr...	Business	Hotel/Resort	San Juan Hotel	Puerto Rico
1970	2	United States	North America	New York City	Bombing/Explosion	Business	Multinational Cor...	Brooklyn General ...	United States
1970	2	United States	North America	New York City	Bombing/Explosion	Business	Multinational Cor...	General Electric ...	United States

Figure 10: Feature Selection

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder

# Columns to be indexed and encoded
categorical_columns = ["country_txt", "region_txt", "city", "attacktype1_txt", "targettype1_txt",
                      "targetsubtype1_txt", "target1", "natlty1_txt", "gname", "motive"]

indexers = [StringIndexer(inputCol=column, outputCol=column+"_index", handleInvalid="keep") for column in categorical_columns]
encoders = [OneHotEncoder(inputCol=column+"_index", outputCol=column+"_ohe") for column in categorical_columns]

pipeline = Pipeline(stages=indexers + encoders)

df_encoded = pipeline.fit(df).transform(df)
```

Figure 11: Indexing and Encoding of Categorical Columns

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help Saving...

Code + Text

```
# Splitting of the dataset into training and testing set
train_data, test_data = df_encoded.randomSplit([0.7, 0.3], seed=42)
```

Figure 12: Data Split into Training and Testing Sets

+ Code + Text

✓ RAM
Disk 

```
[35] #To define the Logistic Regression model
lr = LogisticRegression(labelCol="success", featuresCol="features")

# Training the LR model
lr_model = lr.fit(train_data)

#To make predictions
predictions_lr = lr_model.transform(test_data)

# Selection of (prediction, true label) and compute test error
evaluator_lr = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="success", metricName="areaUnderROC")

# Evaluation of the model and to print the AUC
auc_lr = evaluator_lr.evaluate(predictions_lr)
print(f"AUC of Logistic Regression model is: {auc_lr}")

# Show the first 10 rows of the predictions DataFrame
predictions_lr.show(10)
```

AUC of Logistic Regression model is: 0.816400580551524

Figure 13: Logistics Regression with Output

+ Code + Text

✓ RAM
Disk 

```
[37] from pyspark.ml.classification import LinearSVC

#To define the Support Vector Machine model
svm = LinearSVC(labelCol="success", featuresCol="features")

#Training of the model
svm_model = svm.fit(train_data)

#To make predictions
predictions_svm = svm_model.transform(test_data)

#Selection of (prediction, true label) and compute test error
evaluator_svm = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="success", metricName="areaUnderROC")

#Evaluation of the model and printing the AUC
auc_svm = evaluator_svm.evaluate(predictions_svm)
print(f"AUC of SVM model is: {auc_svm}")

#Showing the first 10 rows of the predictions DataFrame
predictions_svm.show(10)
```

AUC of SVM model is: 0.7070633768746977

Figure 14: Support Vector Machine with Output

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment Share ⚙️

Code + Text

✓ RAM Disk ^

```
[38] from pyspark.ml.classification import RandomForestClassifier

# Define the Random Forest model
rf = RandomForestClassifier(labelCol="success", featuresCol="features", numTrees=100)

# Train the model
rf_model = rf.fit(train_data)

# Make predictions
predictions_rf = rf_model.transform(test_data)

# Select (prediction, true label) and compute test error
evaluator_rf = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="success", metricName="areaUnderROC")

# Evaluate the model and print the AUC
auc_rf = evaluator_rf.evaluate(predictions_rf)
print(f"AUC of Random Forest model is: {auc_rf}")

# Show the first 10 rows of the predictions DataFrame
predictions_rf.show(10)
```

AUC of Random Forest model is: 0.7518142235123367

Figure 15: Random Forest Model with Output

Agboola Olayinka BigDataCW.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment Share ⚙️

+ Code + Text

✓ RAM Disk ^

```
148 #Using the VectorAssembler to combine the feature columns into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features_vector")
train_data_assembled = assembler.transform(train_data)
test_data_assembled = assembler.transform(test_data)

#Setting up the Naive Bayes model
nb = NaiveBayes(featuresCol="features_vector", labelCol="success", predictionCol="prediction")

#Training the Naive Bayes model
nb_model = nb.fit(train_data_assembled)

#To Predict on the test set
predictions = nb_model.transform(test_data_assembled)

#Performance Evaluation of ROC AUC
evaluator = BinaryClassificationEvaluator(labelCol="success", metricName="areaUnderROC")
roc_auc = evaluator.evaluate(predictions)

#To show the output
print(f"ROC AUC: {roc_auc}")
```

ROC AUC: 0.32873730043541366

Figure 16: Naive Bayes Model with Output

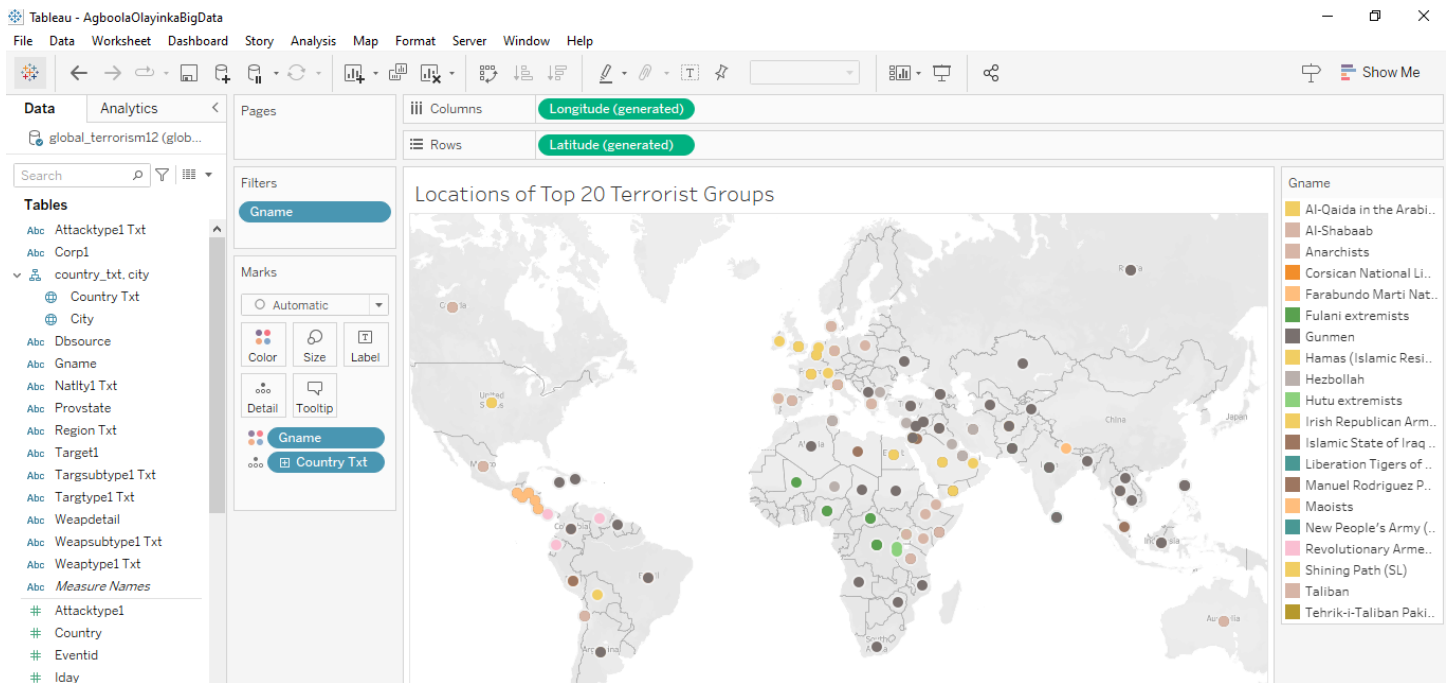


Figure 17: Location of the top 20 Terrorist groups

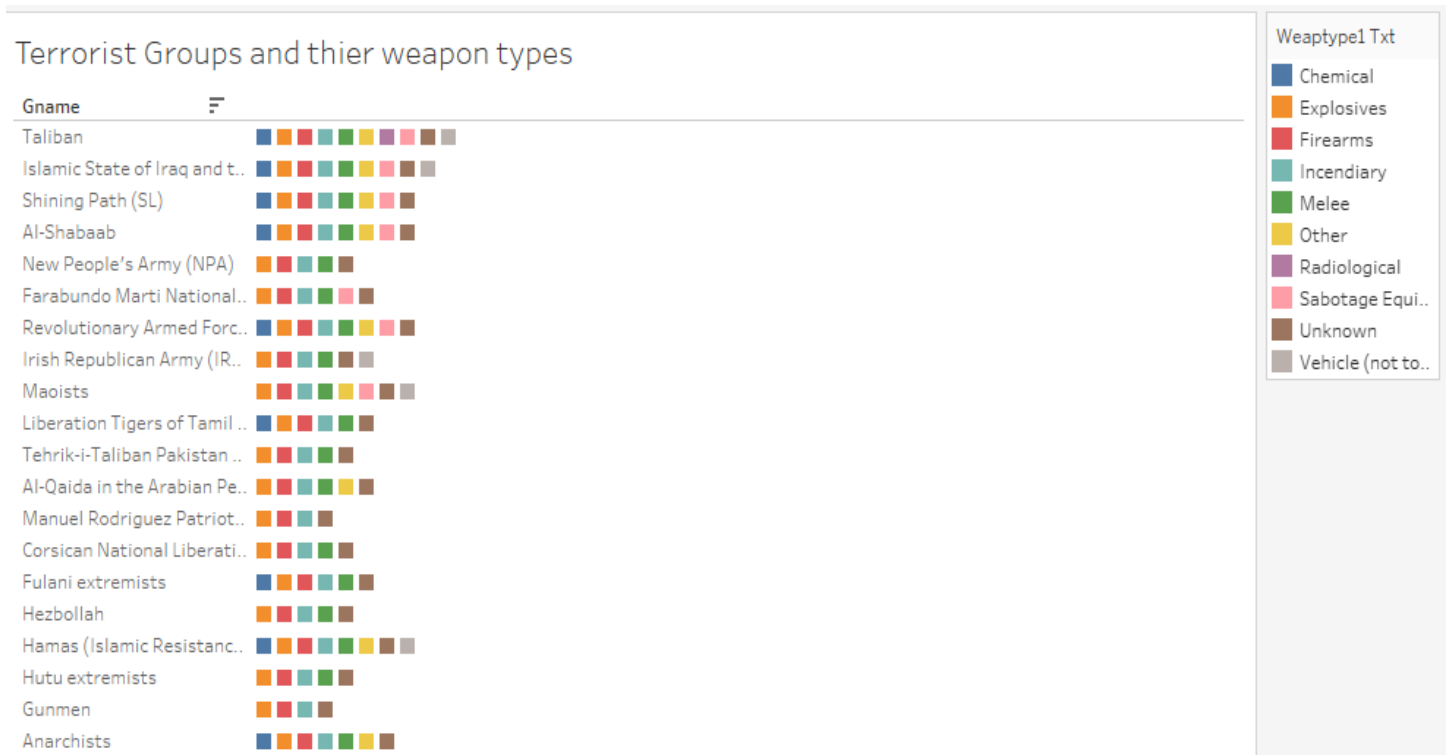
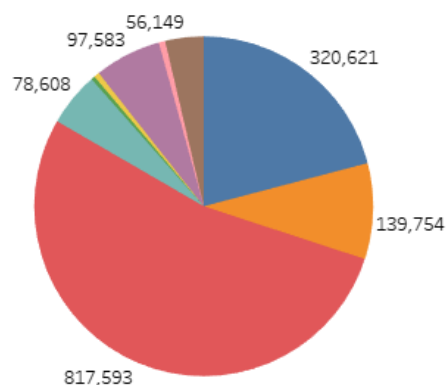


Figure 18: Weapon types of top 20 terrorist groups

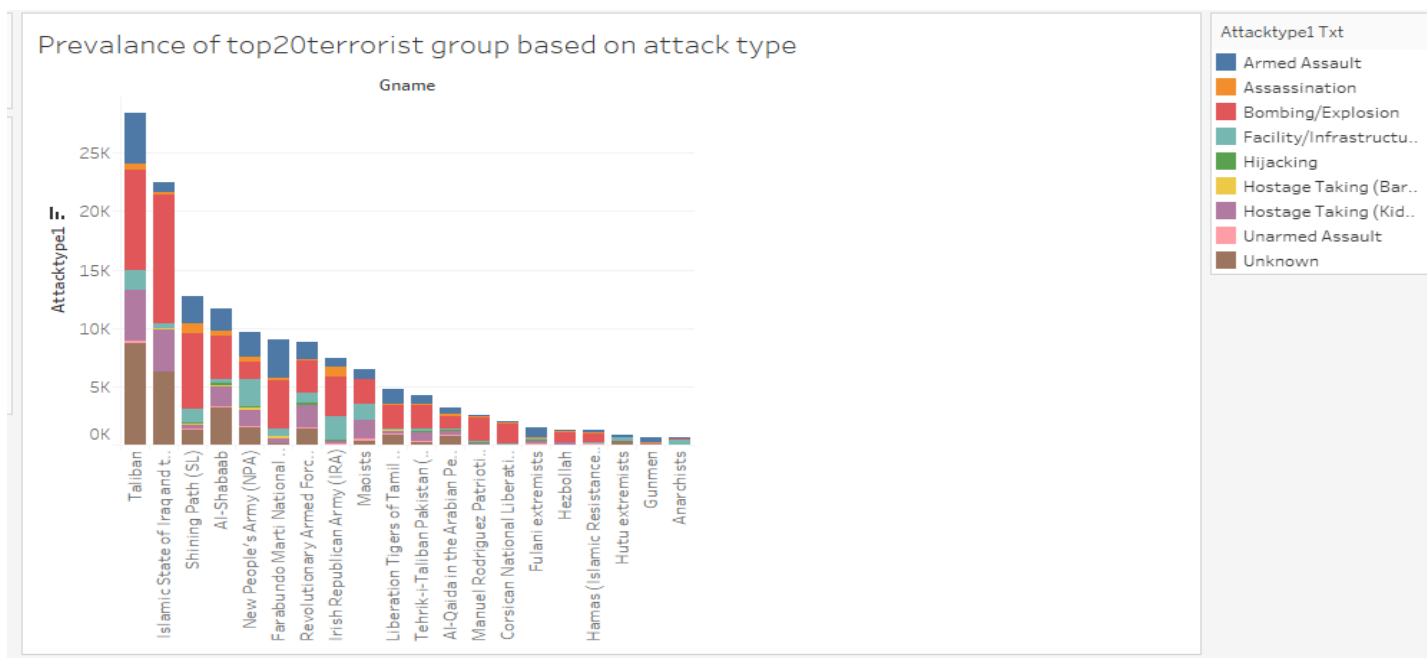
Distribution of attack types



Attacktype1 Txt
Armed Assault
Assassination
Bombing/Explosion
Facility/Infrastructu..
Hijacking
Hostage Taking (Bar..
Hostage Taking (Kid..
Unarmed Assault
Unknown

SUM(Targetype1)
1,533,421

Figure 19: Distribution of Attack Types



Attacktype1 Txt
Armed Assault
Assassination
Bombing/Explosion
Facility/Infrastructu..
Hijacking
Hostage Taking (Bar..
Hostage Taking (Kid..
Unarmed Assault
Unknown

Figure 20: Prevalence of terrorist group based on attack type

Types of Weapon used in different countries

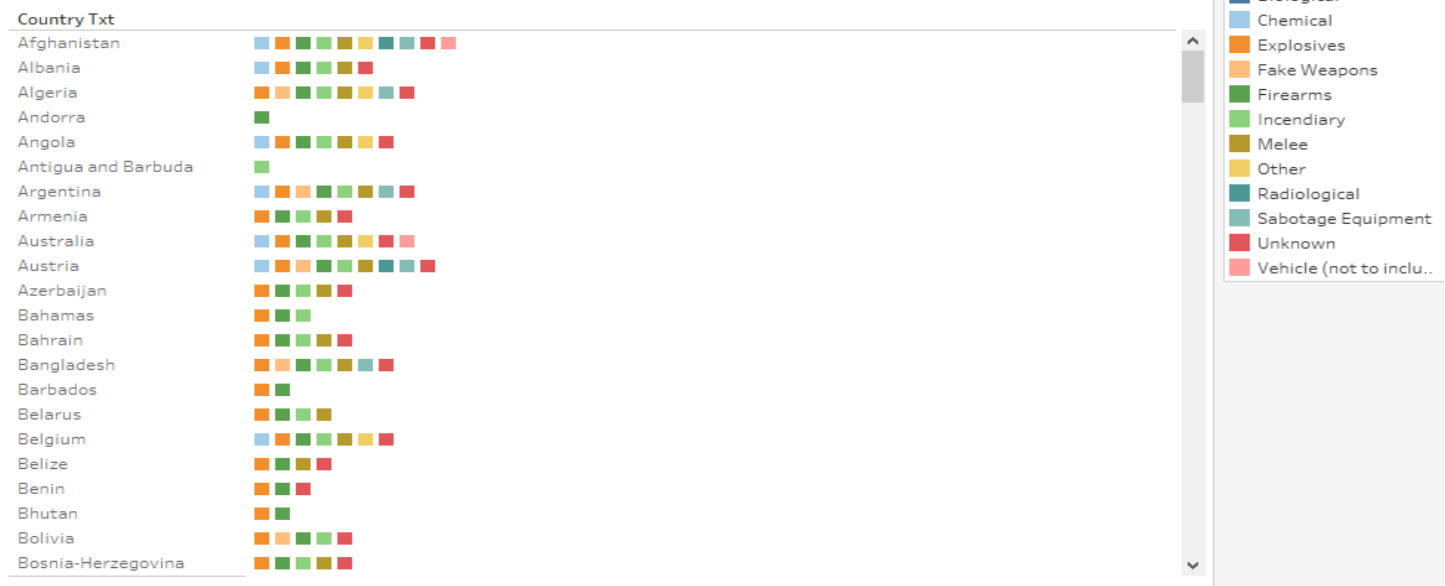


Figure 21: Countries and Weapon types Experienced

Types of Weapon used in different countries

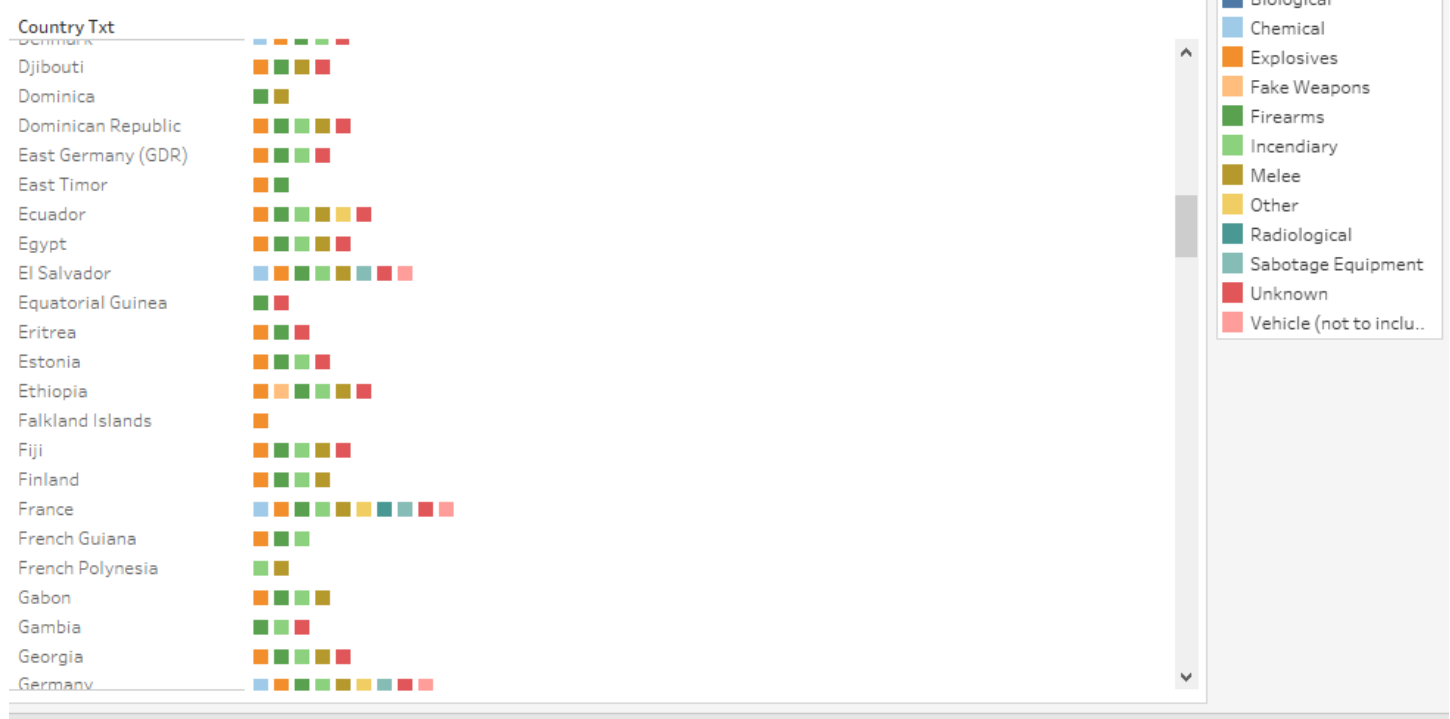


Figure 22: Countries and Weapon types Experienced

Types of Weapon used in different countries

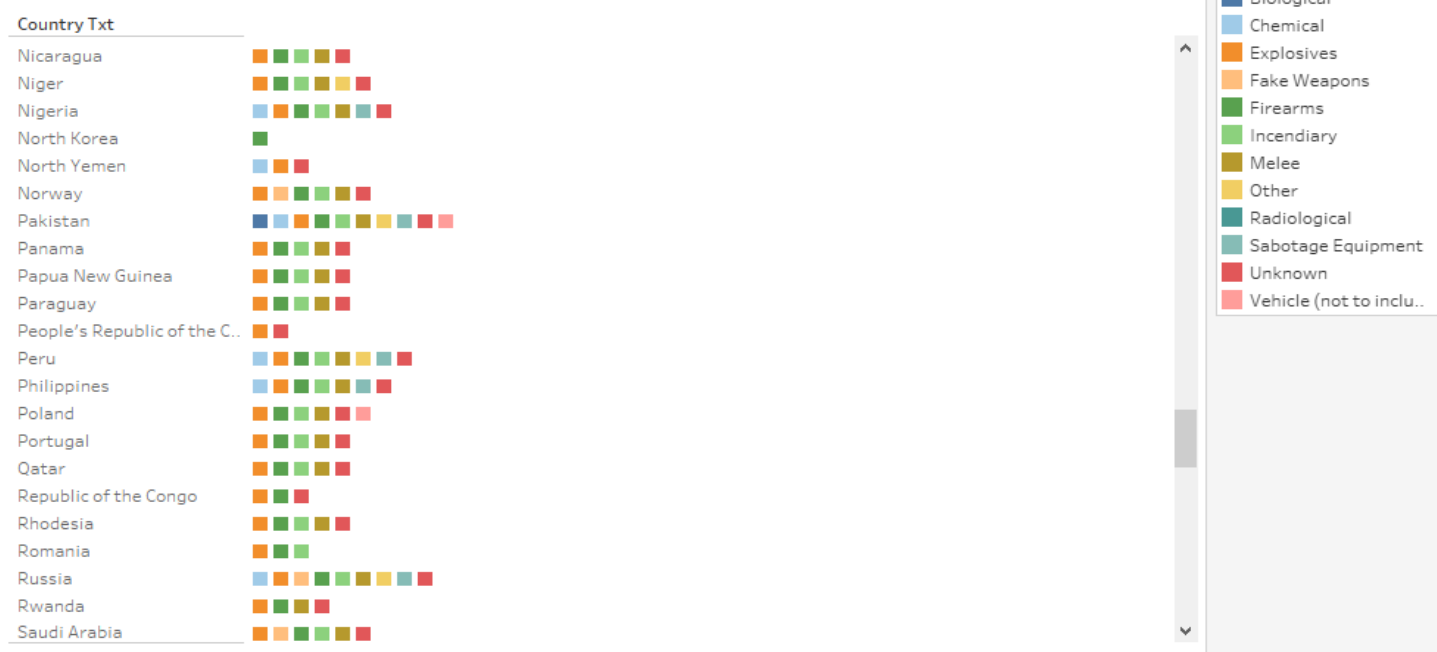


Figure 23: Countries and Weapon types Experienced

Types of Weapon used in different countries



Figure 24: Countries and Weapon types Experienced

FULL PROJECT CODES

```
!pip install pyspark==3.0.0
!pip install findspark
import findspark
findspark.init()

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("global_terrorism").getOrCreate()
csv_path = '/content/sample_data/global_terrorism.csv'

df = spark.read.csv(csv_path, header=True, inferSchema=True)
df.head()
df.show(10)
#to check number of rows
df.count()
#To check number of columns
len(df.columns)
#Data Preprocessing

import pyspark.sql.functions as F
from pyspark.mllib.stat import Statistics

# Drop the columns
df = df.drop('approxdate', 'resolution', 'attacktype3', 'target2', 'corp2')
df = df.drop('ransomamt', 'ransompaid', 'ransompaidus', 'ransomamtus')
df = df.drop('ransomnote', 'nreleased', 'hostkidoutcome', 'hostkidoutcome_txt')
df = df.drop('attacktype3_txt', 'targtype', 'targsubtype', 'natlty2', 'natlty3')
df = df.drop('gname2', 'gsubname3', 'claimmode', 'claimmode2_txt', 'nhours')
df = df.drop('natlty3_txt', 'gsubname', 'claimmode_txt', 'weaptype2', 'ndays')
df = df.drop('nhostkidus', 'weaptype2_txt', 'claim2', 'gsubname2', 'gname3')
# Show the first 10 rows of the DataFrame after dropping missing data
df.show(10)

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier,
LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.feature import PCA, VectorAssembler

# List of encoded features
feature_columns = [column+"_ohe" for column in categorical_columns]

# Assembling encoded features into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features_vector")
df_assembled = assembler.transform(df_encoded)

# Applying PCA
num_principal_components = 13
pca = PCA(k=num_principal_components, inputCol="features_vector",
outputCol="pca_features")
pca_model = pca.fit(df_assembled)
df_pca = pca_model.transform(df_assembled)

# Show transformed data
```

```

df_pca.select("pca_features").show()
# Select the columns
selected_columns = ["iyear", "imonth", "country_txt", "region_txt", "city",
"attacktype1_txt",
                    "targettype1_txt", "targsubtype1_txt", "target1", "natlty1_txt",
"gname", "motive", "success"]

df = df.select(selected_columns)
df.show()
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder

# Columns to be indexed and encoded
categorical_columns = ["country_txt", "region_txt", "city", "attacktype1_txt",
"targettype1_txt",
                    "targsubtype1_txt", "target1", "natlty1_txt", "gname", "motive"]

indexers = [StringIndexer(inputCol=column, outputCol=column+"_index",
handleInvalid="keep") for column in categorical_columns]
encoders = [OneHotEncoder(inputCol=column+"_index", outputCol=column+"_ohe") for column
in categorical_columns]

pipeline = Pipeline(stages=indexers + encoders)

df_encoded = pipeline.fit(df).transform(df)
from pyspark.ml.feature import VectorAssembler

input_cols = ["iyear", "imonth"] + [column+"_ohe" for column in categorical_columns]
assembler = VectorAssembler(inputCols=input_cols, outputCol="features")

df_encoded = assembler.transform(df_encoded)
# Splitting of the dataset into training and testing set
train_data, test_data = df_encoded.randomSplit([0.7, 0.3], seed=42)
#To define the Logistic Regression model
lr = LogisticRegression(labelCol="success", featuresCol="features")

# Training the LR model
lr_model = lr.fit(train_data)

#To make predictions
predictions_lr = lr_model.transform(test_data)

# Selection of (prediction, true label) and compute test error
evaluator_lr = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",
labelCol="success", metricName="areaUnderROC")

# Evaluation of the model and to print the AUC
auc_lr = evaluator_lr.evaluate(predictions_lr)
print(f"AUC of Logistic Regression model is: {auc_lr}")

# Show the first 10 rows of the predictions DataFrame
predictions_lr.show(10)
from pyspark.ml.classification import LinearSVC

#To define the Support Vector Machine model
svm = LinearSVC(labelCol="success", featuresCol="features")

```

```

#Training of the model
svm_model = svm.fit(train_data)

#To make predictions
predictions_svm = svm_model.transform(test_data)

#Selection of (prediction, true label) and compute test error
evaluator_svm = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",
labelCol="success", metricName="areaUnderROC")

#Evaluation of the model and printing the AUC
auc_svm = evaluator_svm.evaluate(predictions_svm)
print(f"AUC of SVM model is: {auc_svm}")

#Showing the first 10 rows of the predictions DataFrame
predictions_svm.show(10)
from pyspark.ml.classification import RandomForestClassifier

# Define the Random Forest model
rf = RandomForestClassifier(labelCol="success", featuresCol="features", numTrees=100)

# Train the model
rf_model = rf.fit(train_data)

# Make predictions
predictions_rf = rf_model.transform(test_data)

# Select (prediction, true label) and compute test error
evaluator_rf = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",
labelCol="success", metricName="areaUnderROC")

# Evaluate the model and print the AUC
auc_rf = evaluator_rf.evaluate(predictions_rf)
print(f"AUC of Random Forest model is: {auc_rf}")

# Show the first 10 rows of the predictions DataFrame
predictions_rf.show(10)
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import BinaryClassificationEvaluator

feature_columns = ["country_txt_oh", "region_txt_oh", "city_oh",
"attacktype1_txt_oh",
                    "targettype1_txt_oh", "targetsubtype1_txt_oh", "target1_oh",
                    "natlty1_txt_oh", "gname_oh", "motive_oh"]

#Using the VectorAssembler to combine the feature columns into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features_vector")
train_data_assembled = assembler.transform(train_data)
test_data_assembled = assembler.transform(test_data)

#Setting up the Naive Bayes model
nb = NaiveBayes(featuresCol="features_vector", labelCol="success",
predictionCol="prediction")

```

```
#Training the Naive Bayes model
nb_model = nb.fit(train_data_assembled)

#To Predict on the test set
predictions = nb_model.transform(test_data_assembled)

#Performance Evaluation of ROC AUC
evaluator = BinaryClassificationEvaluator(labelCol="success", metricName="areaUnderROC")
roc_auc = evaluator.evaluate(predictions)
```