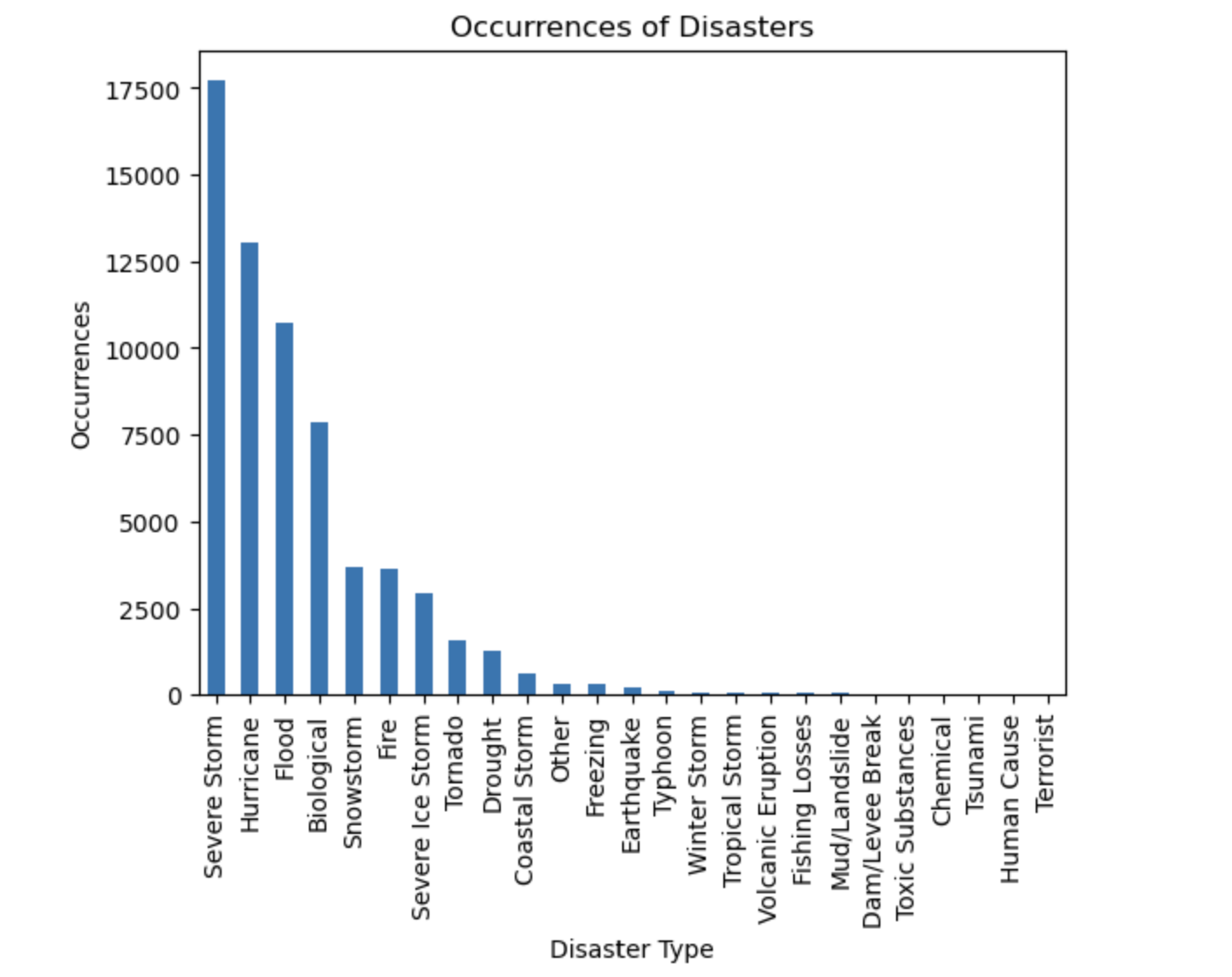
**OpenFEMA's Disaster Declarations Summary**

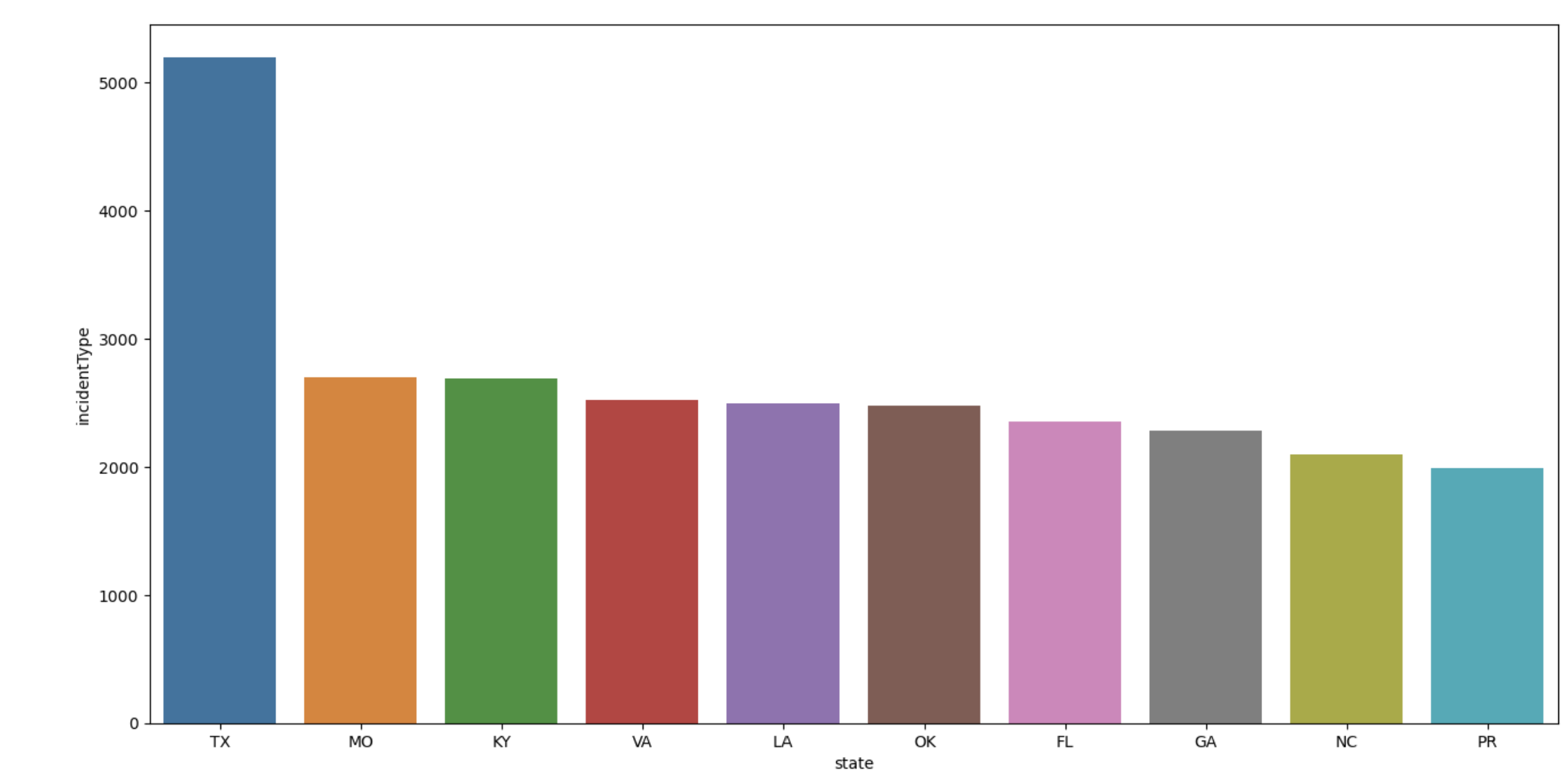
Some Insights from the data visualized (File Name - “Disaster\_Analysis\_initial”)



The chart plotted with occurrences provides a visual representation of the different types of disasters reported in the dataset and their respective frequencies. The x-axis represents the various disaster types, while the y-axis represents the number of occurrences.

Among the types with the highest count, the most frequently reported disaster is Severe Storm, with a total of 17,705 instances. Following closely behind are Hurricane with 13,046 occurrences and Flood with 10,722 occurrences.

On the other hand, the least frequently reported events include Terrorist with 5 occurrences, Human Cause with 7 occurrences, and Tsunami with only 9 occurrences. While these disaster types may be less common, they still provide valuable insights into specific events that have occurred



The chart showcases the top states with the highest number of reported incidents, providing insights into the states that are most susceptible to disasters.

Among these states, Texas (TX) stands out with a significant number of reported incidents, leading the list with 5,195 occurrences. Following closely behind is Missouri (MO) with 2,702 incidents, followed by Kentucky (KY) with 2,693 incidents. Virginia (VA) and Louisiana (LA) also demonstrate a high risk profile, each reporting 2,524 and 2,493 incidents, respectively.

The chart further reveals that Oklahoma (OK), Florida (FL), Georgia (GA), North Carolina (NC), and Puerto Rico (PR) are also states prone to disasters, each reporting a substantial number of incidents, ranging from 2,480 to 1,987 occurrences.

**PreProcessing Data**

Checking for Missing Values:

Examined the dataset for missing values to ensure data integrity and completeness.

Below are the columns which have a significant number of missing values that might impact the analysis.

Null Value Counts (Column-wise):

femaDeclarationString 0

disasterNumber 0

state 0

declarationType 0

declarationDate 0

fyDeclared 0

incidentType 0

declarationTitle 0

ihProgramDeclared 0

iaProgramDeclared 0

paProgramDeclared 0

hmProgramDeclared 0

incidentBeginDate 0

incidentEndDate 605

disasterCloseoutDate 15114

tribalRequest 0

fipsStateCode 0

fipsCountyCode 0

placeCode 0

designatedArea 0

declarationRequestNumber 0

lastIAFilingDate 46561

lastRefresh 0

hash 0

id 0

Upon analyzing the dataset, it was found that some columns have null values. The column incidentEndDate has 605 null values, indicating that for some disaster declarations, the end date of the incident is not available. The column disasterCloseoutDate has 15,114 null values, indicating missing information regarding the closeout date for those disasters. Additionally, the column lastIAFilingDate has 46,561 null values, indicating missing values for the date of the last Individual Assistance filing.

It is important to consider these null values when performing analysis or calculations involving these columns. Depending on the analysis objectives, these null values can be handled by imputing values, excluding rows with missing data.

* For column lastIAFilingDate, as it contains a large number of missing values and they cannot be accurately imputed, one option is to exclude the column from your analysis
* Imputing missing values in the 'incidentEndDate' column with the mean may not be appropriate if the 'incidentEndDate' values are time-dependent and have a sequential nature. In cases where the incident end dates are expected to follow a chronological order, imputing with the mean could introduce incorrect information like previous date which can’t be earlier than Incident start date.

**New Derived features –**

1. Duration\_of\_Incident: Calculated duration of the incident by subtracting the incidentBeginDate from the incidentEndDate. This can provide information about the length of time a disaster event lasted.

2. Declared Programs Count: Feature which equals to sumof number of programs (ihProgramDeclared, iaProgramDeclared, paProgramDeclared, hmProgramDeclared) declared for each disaster declaration. This can indicate the severity or magnitude of the disaster.

3. Time Since Last Disaster: it represents the duration in days since the previous disaster declaration for each state. It captures the time elapsed between consecutive disaster events and provides information about the frequency or recurrence of disasters in a particular region. Further we can aggregate them to find Maximum Time Since Last Disaster and Minimum Time Since Last Disaster.

**Train Test Split**

Target variable –

1. State-Level Impact: If dataset could have state-level information, we can create a target variable based on the overall impact or cost of disasters in each state. This can involve aggregating the damage cost or number of casualties across all disaster events in each state .
2. Severity Level: If we could have information about the severity or impact of each disaster event, such as a severity scale indicating the level of damage or casualties, then we can use that information as a target variable to predict the severity level of future disaster events.
3. Predicting the incident type: we have the "incidentType" column in our dataset, so that can be taken as the target variable and we can build a model to predict the type of incident based on other available features.

**Standardization** –

Standardization, which is also known as z-score normalization, is a common preprocessing technique used in machine learning.

I have used standardization to make ensure that the features showing disasters summary have similar scales, preventing any single feature from dominating the learning process due to its larger magnitude.

By standardizing the data, We can make the features comparable and create a more balanced representation of the numerical variables and it also improve the model's convergence, accuracy, and interpretability.

It also helps eliminate the potential bias introduced by differences in the magnitude of the numerical values.

Standardization is typically applied to numerical features , so here are suitable columns of our dataset-

disasterNumber

fyDeclared

ihProgramDeclared

iaProgramDeclared

paProgramDeclared

hmProgramDeclared

tribalRequest

fipsStateCode

fipsCountyCode

placeCode

declarationRequestNumber

**TASK 3:**

1.

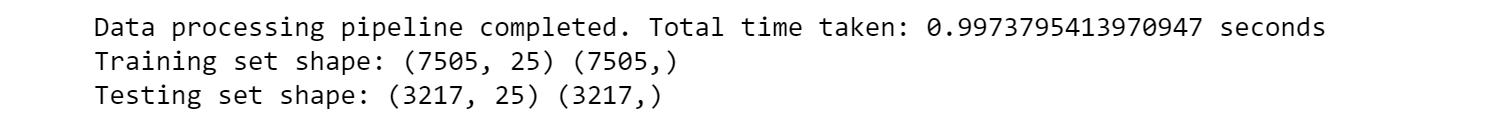
New Pipeline workbook file name – “**disaster\_analysis\_pipeline”**

A data processing pipeline is designed to handle disaster declaration data efficiently. It consists of several steps, - including data loading, preprocessing, feature engineering, standardization, and train-test split.

By following this pipeline, we ensure that the data is prepared effectively for machine learning tasks.

A separated Notebook file (**disaster\_analysis\_pipeline**) has been created which consists of pipeline and code breaked into modules for efficiently processing.

The pipeline took 0.9973 seconds to complete, allowing us to quickly analyze and model the data.



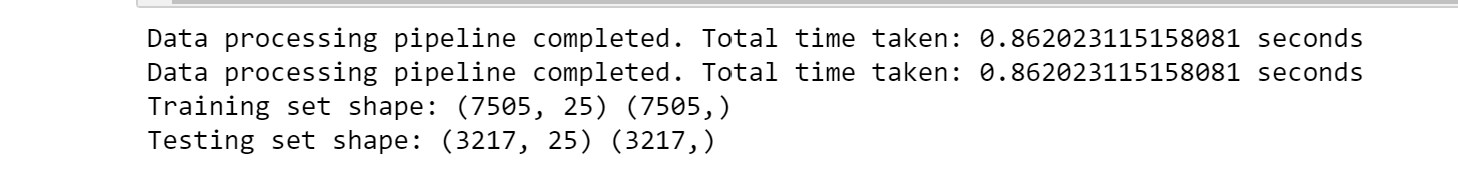
2.

New optimized workbook file name – **“disaster\_analysis\_pipeline\_optimized”**

I optimized the code by incorporating multiprocessing, a technique that utilizes parallel processing to improve efficiency and reduce execution time.

By dividing the pipeline into separate functions and leveraging the multiprocessing.Pool class, I parallelized the preprocessing, feature derivation, and standardization steps. This allowed for concurrent execution of tasks and significantly reduced the overall execution time.

With the optimized code, the data processing pipeline now completes in approximately 0.862 seconds, demonstrating the effectiveness of multiprocessing in enhancing efficiency.



3.

Our team's data scientists can leverage this pipeline to streamline the import and preparation of data for machine learning tasks. The pipeline offers a systematic process for data retrieval, filtering, transformation, feature engineering, and standardization and splitting into train and test datasets.

By adhering to this pipeline, data scientists can effectively process the data and ensure its readiness for machine learning algorithms. The seamless integration of the pipeline into their existing workflow saves valuable time and effort in data preparation, allowing them to concentrate on model development and analysis.

After obtaining the train-test split from the data processing pipeline, our data scientists can utilize the resulting datasets to create models such as linear regression and random forest.

They can fit the model using the training data and evaluate its performance using various metrics such as mean squared error or R-squared. They can then make predictions on the test data using the trained model and assess its accuracy by comparing the predicted values with the actual values.

By analyzing the accuracy of the models and comparing their predictions to the ground truth, our data scientists can gain insights into the effectiveness of different algorithms in modeling the data and making predictions. This allows them to select the most suitable model for the given task and refine it further if necessary to improve its accuracy.

4.

To utilize the data processing pipeline in a cloud platform (here Azure), the data scientists can follow these steps:

1. Set up an Azure environment: Create an Azure account and configure the desired environment, such as Azure Machine Learning Studio, Azure databricks and Azure Notebooks.
2. Create a project: Within the Azure environment, initiate a new project for data analysis and machine learning tasks.
3. Import libraries: Ensure the necessary libraries, including pandas, requests, scikit-learn, and matplotlib, are installed & imported into the project.
4. Define the pipeline code: Create a new Python notebook in Azure and copy the data processing pipeline code which I have created in “disaster\_analysis\_pipeline\_optimized” file
5. Specify API URL and target column: Customize the api\_url variable to the relevant API URL and assign the target column name to target\_column if needs to change, otherwise same can be taken if it aligns to the requirements.
6. Execute the pipeline: Run the code in the Azure environment to execute the pipeline, fetching data, performing preprocessing, deriving features, applying standardization, and splitting the dataset.
7. Access processed data: Once the pipeline completes, access the processed datasets such as X\_train, X\_test, y\_train, and y\_test within the Azure environment. These datasets can be used for further analysis and modeling.
8. Analyze and model: Utilize the processed datasets in Azure for exploratory data analysis, feature engineering, model selection, and training. Leverage Azure's machine learning services, such as Azure Machine Learning Studio or Azure Databricks, for efficient modeling and deployment.

By following the outlined steps and leveraging Azure's cloud platform, data scientists can take advantage of the scalability, flexibility, and integrated services offered. The data processing pipeline seamlessly integrates into Azure environments, enabling team to make the most of platform resources for effective data import, preparation, and streamlined workflows in machine learning.