**METEOR IMPACT ANALYSIS PROGRAM**

**INTRODUCTION TO DATA SCIENCE – PROJECT REPORT**

**Group Members:**

**Ibrahim Mahmood – 2021748**

**Amiya Aden Balouch – 2021090**

**Date of Submission: May 19, 2023**

**Course Instructor: Ma’am Abinta Mir**

**Table of Contents**

**1. Overview**

**2. Methodology**

**2.1 Data Collection and Preprocessing**

**2.2 Feature Selection and Extraction**

**2.2.1 Random Forest**

**2.2.2 Recursive Feature Selection (RFE)**

**2.3 Model Training and Prediction**

**2.3.1 KNN Model**

**2.3.2 Extreme Gradient Boosting (XGB)**

**2.4 Runtime Impact Analysis**

**2.5 Plotting Using Matplotlib and Seaborn**

**2.6 AI Integration**

**3. Key Findings**

**4. Tools and Technologies Used**

**4.1 Python**

**4.2 PySimpleGUI**

**4.3 scikit-learn**

**4.4 Matplotlib**

**4.5 Chat GPT API**

**4.6 Text-to-Speech API**

**4.7 Beautiful Soup for Web Scraping**

**5. Recommendations**

**6. Conclusion**

**1**

**1. Overview**

The meteor impact analysis project aimed to develop a predictive analysis program using techniques such as the K-Nearest Neighbors (KNN) model, Random Forest Regressor and XGB model. The project was conducted as part of the course "Introduction to Data Science". The main objectives were to:

* Perform data collection by looking for large datasets of past data about meteor impacts.
* Clean and preprocess this data.
* Train ML models for predicting certain features of the data.
* Generate comprehensive plots, graphs and a report as the analysis.
* Implement AI components for text generation and text-to-speech functionality.
* Create a user-friendly front end using PySimpleGUI.

**2. Methodology**

**2.1 Data Collection and Preprocessing**

Two datasets were utilized: data set 1, a publicly available dataset on meteor impacts, and data set 2, a detailed dataset on different meteor impacts. Web scraping was performed to acquire data set 1, which was available on Github, and we saved as a CSV file. Data set 2 was downloaded directly from NASA's website as a CSV. The data from both sets underwent cleaning and preprocessing to address missing values and generate local CSV files. This included dropping empty rows, removing irrelevant data and formulating features where needed.

**2.2 Feature Selection and Extraction**

Feature selection is an essential part of model training. Only the features that have significant impact on the prediction to be made must be utilized. We used 2 models to perform this task:

* + 1. **Random Forest:** We used regression with random forest model to predict the most crucial features when it came to calculating Impact Energy. To make use of this model we used the sickit-learn library. This model, like most ML models, takes from the dataset, the training data and the predicted class:

X\_train = df\_train[features]

Y\_train = df\_train["Impact Energy"]

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

**2**

After this we used the .fit() function to train the model:

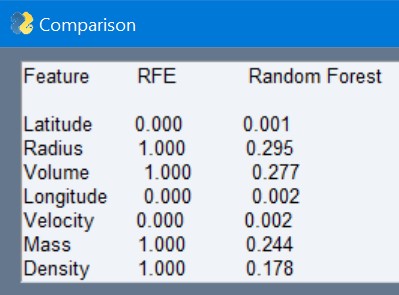
rf.fit(X\_train, Y\_train)

And then we ranked the importance of the features to help with feature extraction:

importances = rf.feature\_importances

* + 1. **Recursive Feature Selection (RFE**): We used this model for cross-checking purposes and to provide further insight about the data. It verified that our model had been trained correctly and selected the right features, namely Mass, Radius, Volume and Density.

We provide an efficient way for the user to view comparison between both:



The functionality and working of the RFE model is quite similar to random forest model. It uses fit() in a similar manner. This is the model being created in our code:

rfe = RFE(estimator=rf, n\_features\_to\_select=4, step=1)

**2.3 Model Training and Prediction**

To predict impact and radiation energies, a KNN model was trained using data set 2. However, due to the lack of common features between the two datasets, statistical and mathematical formulas were employed. Mass calculations were performed by utilizing other features such as diameter, density, and volume. Density values were averaged based on the type of rocks present in data set 1. Once enough mutual features were identified, the KNN model was trained on data set 2 and used to generate predicted values for the energy features in data set 1. Radiated energy was calculated using XGN model.

**3**

* + 1. **KNN Model:** The KNN model consists of 2 main functions: train\_model() and predict\_impact\_energy(). The train\_model() function loads a DataFrame (df\_train) from a CSV file, and then selects the features "Mass", "Radius", "Volume", and "Density" for training. The feature data is extracted from df\_train and stored in X\_train.

We also made use of normalization techniques. To normalize the features, a **MinMaxScaler** object called scaler is created, and the feature data is transformed using scaler.fit\_transform(), resulting in X\_train\_normalized. The target variable "Impact Energy" is extracted from df\_train and stored in Y\_train. A KNN model is created, trained on the normalized feature data (X\_train\_normalized) and target variable (Y\_train) using knn.fit(). Finally, the function returns the trained knn model and the scaler object.

The predict\_impact\_energy() function loads another DataFrame (df\_test) from the CSV file. It removes the "Velocity" column from df\_test. The train\_model() function is then called, returning the trained knn\_model and scaler objects. The features "Mass", "Radius", "Volume", and "Density" are defined again. The feature data is extracted from df\_test and stored in X\_test. To make predictions, the feature data is normalized using scaler.transform() and stored in X\_test\_normalized. The impact energy is predicted for the test dataset using knn\_model.predict(X\_test\_normalized), and the predicted values are stored in predicted\_impact\_energy. These predictions are added as a new column named "Impact Energy" in df\_test. The modified df\_test DataFrame is then saved as a csv file. Finally, the function returns a list containing the first 10 predicted impact energy values from predicted\_impact\_energy.

* + 1. **Extreme Gradient Boosting:** This was a model we had not studied in our course, and we stumbled upon it quite randomly. However, we really liked its regularization techniques, and the fact that it provides support for feature importance. So, we utilized it to predict the Radiated Energy feature.

In the train\_model() function, the code first loads a DataFrame (df\_train) from dataset 2. It selects the features "Mass", "Radius", "Volume", and "Density" for training, stored in the features list. The feature data is extracted from df\_train and assigned to X\_train. In this code, we made use of another normalization method, rather than MinMaxScaler.

**4**

To scale the features, a **RobustScaler** object named scaler is created. The feature data in X\_train is then normalized using scaler.fit\_transform(), resulting in X\_train\_normalized.

The target variable "Radiated Energy" is extracted from df\_train and stored in Y\_train.

A gradient boosting regression model (xgboost\_model) from the XGBoost library is initialized, and the model is trained on the normalized feature data (X\_train\_normalized) and target variable (Y\_train) using xgboost\_model.fit(). Finally, the function returns the trained xgboost\_model and the scaler object.

The predict\_radiated\_energy() function loads another DataFrame (df\_test) from dataset 1. The train\_model() function is called, returning the trained xgboost\_model and scaler objects. The features "Mass", "Radius", "Volume", and "Density" are defined again. The feature data is extracted from df\_test and stored in X\_test. To prepare the test data for prediction, the feature data is normalized using scaler.transform() and stored in X\_test\_normalized. The radiated energy is predicted for the test dataset using xgboost\_model.predict(X\_test\_normalized) and divided by 100 million for scaling purposes. The predicted values are added as a new column named "Radiated Energy" in df\_test. The modified df\_test DataFrame is then saved as a CSV with the name ‘predicted dataset’. Finally, the function returns a list containing the first 10 predicted energy values from the "Radiated Energy" column in df\_test.

**2.3 Runtime Impact Analysis**

The KNN model was further utilized to generate numerical data about possible meteor impacts and their effects based on varying density, velocity of impact, and mass of the meteor. Users could inquire about the effects of a meteor impact in any region, and the KNN model would predict statistical data. We generated an entire ideal dataset at runtime for the user’s location, using the previously trained KNN and XGB models. Other than this, we also generated various plots at runtime.

**5**

**2.4 Plotting Using Matplotlib and Seaborn:**

In our code, we utilized various types of plots and graphs to visualize and analyze the data. To examine the relationship between each feature and the radiated energy, we created **scatter plots** by plotting the feature values on the x-axis and the radiated energy on the y-axis. These scatter plots allowed us to visually understand how changes in the feature values affect the radiated energy, helping us identify any patterns or trends.

To compare the impact energy and radiated energy, we generated a **bar chart**. This chart consisted of two bars for each index, representing the impact energy and radiated energy values. By directly comparing the two energy values, we gained a clear understanding of their magnitudes and relative differences.

To analyze the relationship between each feature and the impact energy, we utilized **line graphs**. By plotting each feature against the impact energy, these graphs provided a visual representation of the trend and variation in impact energy with respect to each feature. This enabled us to identify any patterns or trends in the impact energy as the feature values changed.

For a comprehensive view of the relationships and correlations between different pairs of features, we employed a **pair plot**. By using seaborn's pairplot function, we generated a grid of scatter plots that displayed the relationships between pairs of features. This allowed us to identify any dependencies or patterns within the dataset, contributing to a deeper understanding of the data.

Finally, to visualize the correlation between features, we created a heatmap. By computing the correlation matrix and representing the correlation values with colors, the heatmap provided an intuitive display of the strength and direction of correlations. This enabled us to identify strong positive or negative correlations between features and gain insights into important features within the dataset.

In summary, we utilized these plots and graphs to gain valuable insights and enhance our understanding of the data. They facilitated the identification of patterns, trends, and correlations, enabling us to perform a more comprehensive analysis and interpretation of the dataset.

**6**

**2.5 AI Integration**

To enhance the project's scope, AI tools were integrated. Chat GPT's API was used for text generation, where user input was sent to Chat GPT, and insightful text about the given location was generated at runtime. Additionally, text-to-speech functionality was implemented using the same API.

**3. Key Findings**

* Successful development of a meteor impact analysis program using a trained KNN model.
* The KNN model enabled prediction of impact and radiation energies based on available data.
* Integration of AI components expanded the project's capabilities, providing additional insights through text generation and text-to-speech functionality.
* The user-friendly front end developed with PySimpleGUI allowed easy interaction with the application, enabling location selection on an interactive world map.

**4. Tools and Technologies Used**

The meteor impact analysis project utilized various tools and technologies to successfully develop and implement the program. These tools played a crucial role in data processing, modeling, AI integration, and user interface design. The following are the key tools and technologies employed:

**4.1 Python**

Python, a versatile and powerful programming language, served as the primary language for the project. Python's rich ecosystem of libraries and frameworks provided the necessary tools for data manipulation, modeling, visualization, and user interface development. The project leveraged Python's simplicity and readability, making it easier to implement complex algorithms and integrate different components seamlessly.

**7**

**4.2 PySimpleGUI**

PySimpleGUI, a Python library, was used to create an intuitive and user-friendly graphical user interface (GUI). PySimpleGUI provided a straightforward and efficient way to design and develop the program's front end. It allowed for the creation of interactive elements, such as buttons, input fields, and world maps, enabling users to easily navigate and interact with the program's features. The simplicity and flexibility of PySimpleGUI made it an ideal choice for developing a visually appealing and interactive user interface.

**4.3 scikit-learn**

scikit-learn, a popular machine learning library in Python, was employed for training and implementing the K-Nearest Neighbors (KNN) model. scikit-learn provides a wide range of machine learning algorithms and utilities, making it easy to preprocess data, split datasets, train models, and evaluate their performance. The KNN model in scikit-learn was utilized to predict impact and radiation energies based on the available meteor impact data. Its ease of use and extensive documentation facilitated the implementation and fine-tuning of the model.

**4.4 Matplotlib**

Matplotlib, a powerful plotting library in Python, was utilized for generating descriptive plots and graphs. These visualizations enhanced the understanding of the impact's effects by presenting the data in a clear and concise manner. Matplotlib's versatility and extensive customization options allowed for the creation of various types of plots, including scatter plots, bar charts, and geographical heatmaps. The integration of Matplotlib with the program facilitated the visual representation of data, enabling users to interpret and analyze meteor impact information effectively.

**4.5 Chat GPT API**

The Chat GPT API was integrated into the program to leverage the capabilities of the GPT-3 language model for text generation. Chat GPT's API enabled the program to generate insightful and context-specific text about meteor impacts, geological characteristics, and mitigation strategies. By sending user input to the Chat GPT API, the program received dynamic and informative responses, enhancing the information provided to the users. The Chat GPT API expanded the program's capabilities and made it more interactive and informative.

**8**

**4.6 Text-to-Speech API**

To expand the project's capabilities, we integrated AI tools, specifically utilizing Chat GPT's API for text generation. This integration allowed us to generate insightful text about a given location in real-time. When a user input was received, it was sent to Chat GPT, which then generated relevant text about the location. This feature provided dynamic and informative content to enhance the project's functionality.

Moreover, we implemented text-to-speech functionality using the same Chat GPT API. By leveraging this functionality, the generated text was converted into speech. This enabled users to listen to the generated content instead of reading it, offering an alternative and convenient way to access the information.

In one specific scenario, when the "Write Report" event was triggered, a long statistical report was requested. The report focused on the hypothetical scenario of a large meteor falling on a specific country, taking into account real-world data rather than using dummy information. The report aimed to provide a realistic analysis of the situation by considering factors such as the country's population density, infrastructure, geography, and emergency response capabilities. Various aspects were addressed in the report, including the extent of damage, incurred costs, injuries, fatalities, and more. Percentages and statistics were utilized to present a comprehensive overview and better explain the situation.

To generate the report dynamically, we utilized the askGPT function from another file. This function prompted Chat GPT to generate the text based on the given parameters, such as the country name and the scenario. The generated report was then displayed in the application's user interface, specifically in the "Dynamic Text" section.

Additionally, we provided the "Read Report" functionality to accommodate users who preferred listening to the report instead of reading it. We used the pyttsx3 library to initialize a text-to-speech engine. The generated text was passed to the engine, which then synthesized it into speech. By adjusting the voice settings, such as the language and the rate of speech, we ensured a suitable auditory experience for the users. The synthesized speech was played back to the user, allowing them to consume the report's content through audio.

**9**

In summary, the integration of AI tools, including Chat GPT's text generation API and text-to-speech functionality, added significant value to the project. It enabled real-time generation of insightful text about a given location and provided users with the option to either read or listen to the generated content. The "Write Report" event allowed for the creation of statistical reports with realistic data, while the "Read Report" event facilitated the auditory consumption of the generated text. These features expanded the project's capabilities and enhanced the user experience by offering dynamic, informative, and accessible information.

**4.7 Beautiful Soup for Web Scraping**

Web scraping was employed to collect data for meteor impacts. The Python libraries Beautiful Soup and Requests facilitated the web scraping process, allowing the program to extract data from online sources. These libraries provided tools for parsing HTML, retrieving data from web pages, and extracting specific information. Web scraping enabled the acquisition of relevant meteor impact data, which served as the foundation for the program's analysis.

**5. Restrictions**

* GUI could have better exception handling.
* Data (and its quantity) used may cause models to be imperfectly trained.
* The program is restricted to analysis of areas specific to country only.
* Text-to-speech integration causes software to halt temporarily while the program is generating speech. Hence, no other functions can be performed during this time.

**6. Recommendations and Conclusions**

Based on the project's findings, the following recommendations and conclusions are made:

**6.1 Recommendations:**

* Continuously update and refine the KNN model by incorporating additional relevant data to improve prediction accuracy.
* Explore the possibility of integrating other machine learning algorithms to compare their performance with the KNN model.
* Enhance the text generation component by incorporating more advanced natural language processing techniques.

**10**

**6.2 Conclusion:**

* The developed meteor impact analysis program successfully predicted impact and radiation energies using the KNN model.
* The integration of AI components added valuable insights and improved the user experience.
* The user-friendly front end facilitated easy interaction and visualization of meteor impact effects.

In conclusion, the meteor impact analysis project demonstrated the successful application of Data Science. The developed program provided insights into meteor impacts and their effects, showcasing the potential for future research and applications in this domain.

**11**