Meteor Impact Analysis

Ibrahim Mahmood - 2021748
Amiya Aden - 2021090

***Abstract*—This paper describes a meteor impact research project that made use of K-Nearest Neighbours (KNN), Random Forest Regressor, and XGB machine learning models. The goal of the research was to gather, clean, and preprocess data from two datasets in order to forecast impact and radiation energies. Recursive Feature Selection (RFE) and Random Forest models were used to pick and extract features. The software has an intuitive user interface with AI-based text production and text-to-speech features. The implementation utilized Python, PySimpleGUI, scikit-learn, Matplotlib, and Chat GPT API. The results showed that model training was successful, the KNN model is predictive, and AI integration adds value.**

1. METHODOLOGIES

1.1 Data Collection and Pre-processing

Data set 1, a publicly accessible dataset on meteor impacts, and data set 2, a thorough dataset on various meteor impacts, were both used. Data set 1 was obtained by web scraping and stored as a CSV file; it was made accessible on Github. Data set 2 was simply obtained as a CSV file from the NASA website. Both sets of data underwent cleaning and preprocessing in order to fill in any missing values and produce local CSV files. This includes deleting empty rows, eliminating unnecessary information, and adding functionality as needed.

1.2 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.

1.2.1 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)
```

1.2.2 Random Forest: 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 table is given ahead.

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:

Comparison		
Features	***RF***	***RFE***
Latitude	0	0.001
Radius	1	0.295
Volume	1	0.277
Longitude	0	0.002
Velocity	0	0.002
Density	1	0.178
Mass	1	0.244

TABLE I. COMPARISON: RF AND RFE

1.3 Model Training and Prediction

Using data from set 2, a KNN model was trained to forecast the energies of impact and radiation. However, statistical and mathematical procedures were used because the two datasets did not have any common characteristics. Other parameters including diameter, density, and volume were used to produce mass calculations. In data set 1, density values were averaged depending on the kinds of rocks that were present. After finding sufficient mutual features, the KNN model was trained on data set 2 and utilised to provide projected values for the energy characteristics in data set 1. XGN model was used to calculate the amount of radiation. All the headings in the main body of your paper are numbered (automatically).

1.3.1 KNN Model: The train_model() and predict_impact_energy() methods are the two key parts of the KNN model. The train_model() method reads a DataFrame (df_train) from a CSV file and chooses the features "Mass", "Radius", "Volume", and "Density" for training. From df_train, the feature data is taken, and it is then saved in X_train.

Additionally, we used normalisation strategies. The feature data is modified using scaler.fit_transform() to produce X_train_normalized by creating a MinMaxScaler object called scaler and calling scaler. From df_train, the goal variable "Impact Energy" is taken out, and it is then saved in Y_train. Using knn.fit(), a KNN model is built and trained on the target variable (Y_train) and normalised feature data (X_train_normalized). The function then returns the scaler object and the trained knn model.

Another DataFrame (df_test) is loaded by the predict_impact_energy() function from the CSV file. The "Velocity" column in df_test is deleted. The trained knn_model and scaler objects are then returned by using the train_model() method. Redefining the terms "Mass," "Radius," "Volume," and "Density" From df_test, the feature data is taken, and it is then saved in X_test. The feature data is transformed using scaler.transform() and then saved

in X_test_normalized in order to generate predictions. Using knn_model.predict(X_test_normalized), the

impact energy is projected for the test dataset, and the anticipated values are saved in predicted_impact_energy. In df_test, a new column called "Impact Energy" is created to reflect these forecasts. A csv file is then created from the changed df_test DataFrame. In the function's final output, the first 10 anticipated impact energy values from predicted_impact_energy are returned as a list.

1.3.2 XGB Model: 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 order to anticipate the Radiated Energy characteristic, we used it. A RobustScaler object called scaler is constructed in order to scale the features.

After using scaler.fit_transform() to normalise the feature data in X_train, X_train_normalized is produced. From df_train, the goal variable "Radiated Energy" is taken out and saved in Y_train. Initialising a gradient boosting regression model (xgboost_model) using the XGBoost package, the model is trained using xgboost_model.fit() on the target variable (Y_train) and the normalised feature data (X_train_normalized). The function then returns the scaler object and the trained xgboost_model.

Another DataFrame (df_test) from dataset 1 is loaded using the predict_radiated_energy() method. The trained xgboost_model and scaler objects are returned by the call to the train_model() method. Redefining the terms "Mass," "Radius," "Volume," and "Density" From df_test, the feature data is taken, and it is then saved in X_test. The feature data is normalised using scaler.transform() and saved in X_test_normalized to make the test data ready for prediction. In order to scale, the radiated energy is projected for the test dataset.

Using xgboost_model.predict(X_test_normalized). In df_test, a new column called "Radiated Energy" is introduced to reflect the expected values. A CSV file named "predicted dataset" is then created from the updated df_test DataFrame. The function then produces a list of the top 10 projected energy values from the "Radiated Energy" column in the df_test data set.

1.4 Runtime Impact Analysis

The KNN model was further used to provide numerical data regarding potential meteor strikes and their consequences depending on changing density, impact velocity, and meteor mass. Users might ask the KNN model to forecast statistical data on the outcomes of a meteor strike in any location. Using the previously trained KNN and XGB models, we created a whole ideal dataset at runtime for the user's location.

1.5 Plotting Using Seaborn and MatPlotLib

1) In our code, we utilized different sorts of plots and charts to imagine and analyze the information. To look at the relationship between each include and the emanated vitality, we made diffuse plots by plotting the include values on the x-axis and the transmitted vitality on the y-axis. These diffuse plots permitted us to visually understand how changes within the highlight values influence the emanated vitality, making a difference us distinguish any designs or patterns.

2) To compare the affect vitality and emanated vitality, we created a bar chart. This chart comprised of two bars for each file, speaking to the affect vitality and transmitted vitality values. By straightforwardly comparing the two vitality values, we picked up a clear understanding of their sizes and relative contrasts.

3) To analyze the relationship between each highlight and the affect vitality, we utilized line charts. By plotting each highlight against the affect vitality, these charts given a visual representation of the slant and variety in affect vitality with regard to each include. This enabled us to distinguish any designs or patterns within the affect vitality as the include values changed.

4) For a comprehensive see of the connections and relationships between diverse sets of highlights, we utilized a combine plot. By utilizing seaborn's pairplot work, we produced a network of scramble plots that shown the connections between sets of highlights. This permitted us to distinguish any conditions or designs inside the dataset, contributing to a more profound understanding of the information.

5) At last, to imagine the relationship between highlights, we made a heatmap. By computing the relationship framework and speaking to the relationship values with colors, the heatmap given an natural show of the quality and course of relationships. This empowered us to distinguish solid positive or negative relationships between highlights and pick up bits of knowledge into critical highlights inside the dataset.

6) In outline, we utilized these plots and charts to pick up profitable experiences and upgrade our understanding of the information. They encouraged the recognizable proof of designs, patterns, and relationships, empowering us to perform a more comprehensive investigation and translation of the dataset.

1.6 AI Integration

AI tools were added to the project to broaden its reach. The text was generated using Chat GPT's API, which received user input and produced interesting text about the location at runtime. Using the same API, text-to-speech capabilities were also developed.

2. KEY FINDINGS

• The KNN model allowed for the prediction of impact and radiation energy based on the data that was available, resulting in the successful construction of a meteor impact study programme.
• The user-friendly front end created with PySimpleGUI allowed easy interaction with the application, enabling location selection on an interactive world map.
• The integration of AI components expanded the project's capabilities, providing additional insights through text generation and text-to-speech functionality.

3. TOOLS AND TECHNOLOGIES

3.1 Python

The project's main programming language was Python, which is both flexible and strong. The required tools for data processing, modelling, visualisation, and user interface development were made available by the robust ecosystem of libraries and frameworks in Python. The project took use of Python's readability and simplicity, which made it simpler to construct complicated algorithms and smoothly combine various components.

3.2 PySimpleGUI

An easy-to-use graphical user interface (GUI) was made using the Python module PySimpleGUI. The front end of the programme may be designed and developed with ease using PySimpleGUI. It enabled the development of interactive components that made it simple for users to browse and interact with the program's features, such as buttons, input fields, and global maps. PySimpleGUI was a great option for creating a user interface that is both aesthetically pleasing and engaging because of its simplicity and versatility.

3.3 Scikit-Learn

The K-Nearest Neighbours (KNN) model was trained and put into use using scikit-learn, a well-known machine learning package written in Python. A variety of machine learning methods and tools are available through scikit-learn, making it simple to preprocess data, divide datasets, train models, and assess their effectiveness. Based on the meteor impact data that was available, the KNN model in scikit-learn was used to forecast the impact and radiation energies. The model's implementation and improvement were assisted by its user-friendliness and thorough documentation.

3.4 MatPlotLib

We produced informative charts and graphs using Matplotlib, a robust Python plotting package. By clearly and succinctly displaying the data, these visualisations improved comprehension of the impact's impacts. The ability to create a variety of plots, including scatter plots, bar charts, and geographic heatmaps, was made possible by Matplotlib's adaptability and broad customization possibilities. The program's inclusion with Matplotlib allowed the visual depiction of data, allowing users to properly evaluate and analyse data on meteor impact.

3.1 Python

The project's main programming language was Python, which is both flexible and strong. The required tools for data processing, modelling, visualisation, and user interface development were made available by the robust ecosystem of libraries and frameworks in Python. The project took use of Python's readability and simplicity, which made it simpler to construct complicated algorithms and smoothly combine various components.

3.5 Chat GPT API

An easy-to-use graphical user interface (GUI) was made using the Python module PySimpleGUI. The front end of the programme may be designed and developed with ease using PySimpleGUI. It enabled the development of interactive components that made it simple for users to browse and interact with the program's features, such as buttons, input fields, and global maps. PySimpleGUI was a great option for creating a user interface that is both aesthetically pleasing and engaging because of its simplicity and versatility.

3.6 Text-to-Speech API

Our approach converts text to speech using the pyttsx3 package. The voice synthesis process is started by the tts function using text as the input. Using setProperty, we changed the speech rate to 180 words per minute and the voice property to "english+f3". We make sure the text is prepared for synthesis by enqueuing it using engine.say(text). Engine.runAndWait() then begins the synthesis and waits for it to finish. This programme offers a flexible and user-friendly method for turning text into speech that may be used for a variety of purposes, including voice assistants, the creation of audiobooks, and accessibility improvements.

3.7 Web Scraping Using Beautiful Soup

Web scraping was used to get information about meteor strikes. The web scraping procedure was made easier by the Python packages Beautiful Soup and Requests, enabling the programme to extract data from internet sources. These libraries included resources for extracting particular information, accessing data from web sites, and parsing HTML. Web scraping made it possible to gather pertinent meteor impact information, which laid the groundwork for the program's study.

4. 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.

5. RECOMMENDATIONS AND CONCLUSION

5.1 Recommendations:

• To increase prediction accuracy, keep updating and improving the KNN model with new pertinent data.
• Investigate the integration of other machine learning algorithms to see how well they perform in comparison to the KNN model.
• Improve the text generating function by adding more sophisticated natural language processing methods.

5.1 Conclusion:

• The KNN model was successfully used in the built meteor impact analysis programme to estimate impact and radiation energy. • The incorporation of AI components enhanced the user experience and offered useful insights.
• The user-friendly front end made it simple to interact with and view the repercussions of meteor impacts.

In conclusion, the study on meteor impact analysis showed how effectively artificial intelligence may be used. The created programme shed light on meteor strikes and their results, demonstrating the possibility for further study and use in this area.

REFERENCES

[1] Cody Winchester (Github): *Earth Impact Data*, https://github.com/cjwinchester/earth-impact-data/blob/main/earth-impact-craters.csv (accessed *March 2023).*

[2] Jet Propulsion Laboratory*: Fireballls and Bolide Data*, https://cneos.jpl.nasa.gov/fireballs/ (accessed *March 2023).*

[3] Open AI*: API Reference*, https://platform.openai.com/docs/api-reference/authentication (accessed April 2023).

[4] Kaggle*: Study Aestroid Impact Possibility*, https://www.kaggle.com/code/mathchi/study-asteroid-impact-possibility (accessed April 2023).

[5] GeeksforGeeks*: Themes in PySimpleGUI*, https://www.geeksforgeeks.org/themes-in-pysimplegui/ (accessed May 2023).

[6] Stack Overflow*: Adding Elements horizontally instead of vertically*, https://stackoverflow.com/questions/71721158/how-do-i-add-elements-horizontally-instead-of-vertically-in-pysimplegui (accessed May 2023).

[7] Tutorials Point*: Adding Elements horizontally instead of vertically*, https://www.tutorialspoint.com/pysimplegui/pysimplegui_popup_windows.htm (accessed May 2023).

[8] Scikit Learn*: Random Forest Regressor*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (accessed May 2023).

[9] Free Code Camp*: Web Scraping*, https://www.freecodecamp.org/news/web-scraping-python-tutorial-how-to-scrape-data-from-a-website/ (accessed March 2023).