A REPORT ON

ANALYSIS OF TERRORIST ATTACKS IN INDIA FROM 1975 TO 2016

BY RHIDDHIJIT ADHIKARI

CONTENTS

INTRODUCTION

Terrorism has emerged as a significant threat to India's national security over the years. The diverse socio-political landscape, geographical proximity to conflict-prone regions, and internal vulnerabilities have made India susceptible to various forms of terrorism. Acts of terror have not only caused immense loss of innocent lives but have also hampered economic growth, instilled fear in society, and challenged the fabric of India's multicultural identity.

In recent times, India has witnessed several devastating terrorist attacks perpetrated by both domestic and international extremist groups. These attacks have targeted prominent cities, religious sites, transportation networks, and government establishments, aiming to create chaos and undermine the country's stability.

One of the most horrific incidents was the 2008 Mumbai attacks, where ten gunmen belonging to the Pakistan-based militant organization Lashkar-e-Taiba (LeT) unleashed a series of coordinated shootings and bombings across the city. This incident claimed the lives of 166 people and left hundreds injured, sending shockwaves throughout the nation and highlighting the urgent need for robust counter-terrorism measures.

Apart from external threats, India has also dealt with homegrown terrorist organizations like the Indian Mujahideen (IM) and the Students Islamic Movement of India (SIMI). These groups have been involved in carrying out deadly bombings and orchestrating other acts of violence, often motivated by communal tensions or radical ideologies. Their presence underscores the complex challenge of tackling both international and domestic terrorism within the country.

The Indian government, recognizing the gravity of the situation, has taken numerous steps to combat terrorism. It has strengthened intelligence agencies, improved international cooperation, enhanced border security, and enacted stringent laws to punish terrorists and their financiers. Additionally, initiatives have been undertaken to address underlying issues such as poverty, unemployment, and social alienation that contribute to the vulnerability of individuals to extremist ideologies.

While progress has been made in curbing terrorism, the threat persists, necessitating a continued collective effort from various stakeholders, including the government, security forces, civil society, and the international community. Combating terrorism requires a multi-pronged approach that combines effective intelligence gathering, proactive law enforcement, community engagement, and efforts to address root causes.

In this project, we delve deeper into the various dimensions of terrorism in India, exploring its causes, impacts, and counter-terrorism strategies. By understanding the complexities surrounding this issue, we can work towards building a safer, more resilient nation that upholds the values of peace, harmony, and justice.

PROBLEM STATEMENT

The main objective of this project is to test the performance of machine learning and deep learning algorithms for predictive analysis. The dataset used for this purpose is the Global Terorrism Database obtained from Kaggle.

Machine learning algorithms play a crucial role in predictive analysis. These algorithms, such as linear regression, decision trees, support vector machines, and neural networks, learn patterns from historical data to make predictions on unseen data. They are trained using labeled examples, where the algorithm attempts to find relationships between input features and the target variable

.

During the data preprocessing stage, various techniques are applied to clean and transform the data, ensuring its quality and relevance. Feature selection helps identify the most influential variables that contribute to accurate predictions while reducing complexity and noise.

Model training involves feeding the prepared data into the selected algorithm, which learns from the patterns and adjusts its internal parameters. This process aims to minimize prediction errors and optimize the model's performance. Model evaluation metrics, such as accuracy, precision, recall, and F1 score, are used to assess the model's effectiveness.

Once the model is trained and validated, it can be used for prediction purposes. New, unseen data is fed into the model, and it generates predictions based on the learned patterns. These predictions enable businesses to anticipate customer behavior, market trends, risk assessments, demand forecasting, and other critical insights.

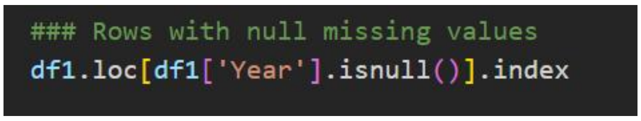
However, it's important to note that predictive analysis using machine learning is not infallible. The accuracy of predictions depends on the quality of input data, the appropriateness of the chosen algorithm, and potential biases present in the dataset. Regular monitoring and retraining of models are necessary to ensure they remain relevant and reliable over time.

DATA PREPROCESSING

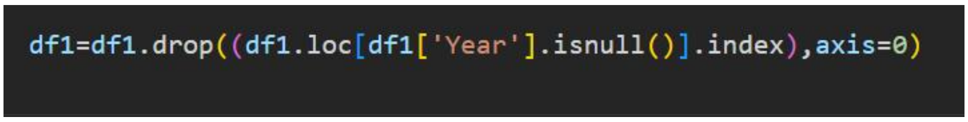
Since the dataset contained details of countries from all over the world, many rows had to be eliminated, leaving only the rows with information about India. It was achieved using the command df[df.loc[‘country’]==’India’], where df is the name assigned to the dataset.

Data preprocessing is vital since the dataset contains multiple values which are not numerical, and algorithms can be trained only on numerical features. The steps usually followed are:

1. Handling missing values: Columns with missing values are usually replaced with mean or median of all the values in the feature. This dataset had columns with upto 7 missing values. Since only the dates were missing, these rows were dropped completely, because taking the mean, median value of dates can give a completely inaccurate value. First, the row numbers with missing values were obtained using the df.isnull().index command, then these rows were dropped using the df.drop() command.



```
### Rows with null missing values
df1.loc[df1['Year'].isnull()].index
```

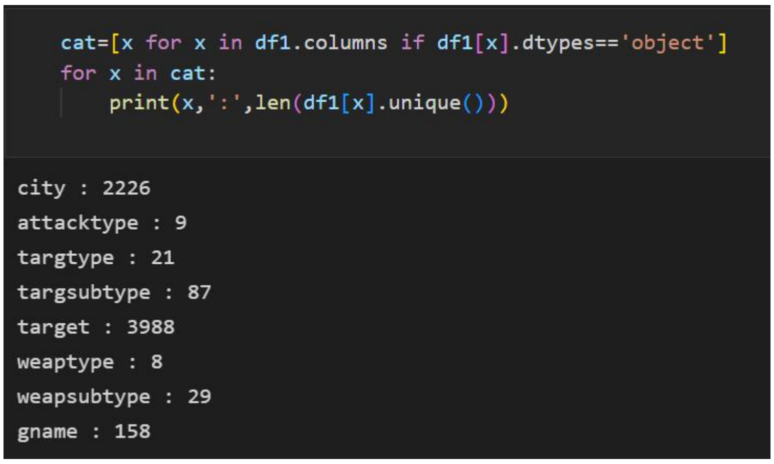


```
df1=df1.drop((df1.loc[df1['Year'].isnull()].index),axis=0)
```

2. Handling categorical values: Columns with categorical values are obtained using the following command:

```
cat=[x for x in df1.columns if df1[x].dtypes=='object']
```

Then, the number of unique values in each column is obtained:

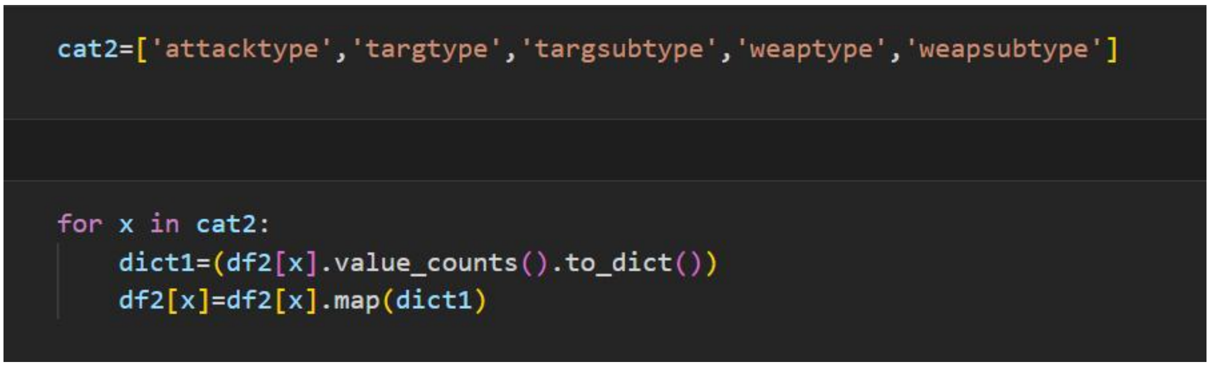


```
cat=[x for x in df1.columns if df1[x].dtypes=='object']
for x in cat:
    print(x,':',len(df1[x].unique()))
```

```
city : 2226
attacktype : 9
targtype : 21
targsubtype : 87
target : 3988
weaptype : 8
weapsubtype : 29
gname : 158
```

The “city” and “target” columns have a large number of unique values, hence they will not be selected as input labels. Hence, the final categorical labels are: attacktype, targtype, targsubtype, weaptype, weapsubtype, gname.

Then, the categorical values have to be converted to numerical values. Few common methods are: one hot encoding, label encoding, frequency encoding. In this case, frequency encoding is used. In this method, each categorical value is replaced by their frequency, i.e, the number of times they are present in the column. For example, suppose a value “a” is present 50 times in a column, then it will be replaced by the number 50.



```
cat2=['attacktype','targtype','targsubtype','weaptype','weapsubtype']
```

```
for x in cat2:
    dict1=(df2[x].value_counts().to_dict())
    df2[x]=df2[x].map(dict1)
```

First, the value_counts() method is used which returns the frequency of each value in a column, then these values are stored in a dictionary. Then the dictionary is mapped to each column, so that the values get replaced. Now that all the values are numerical, independent and dependent features need to be selected. Independent features are the ones which will be used to make the predictions, while dependent features are the ones that will be predicted.

Independent features,
x=['latitude','longitude','targtype','targsubtype','weaptype','weapsubtype','gname','Month','Day','Year']
In this case the value to be predicted is chosen to be the attacktype. Therefore, dependent feature, y=[‘attacktype’]

3. Splitting the data: Data splitting into training, validation, and testing sets is conducted to assess model performance. This prevents overfitting where a model performs well on training data but fails to generalize to unseen data. Splitting is done by the following method:

```
from sklearn.model_selection import train_test_split
X_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
```

Here, x=independent features, y=dependent feature, test_size = percentage of data that will be using for test (0.25= 75% data for training, while 25% for testing), random_state = randomness of selecting features (0 by default).

4. Data scaling and normalisation: Data normalization or scaling is applied to ensure that different features have comparable scales. This process eliminates biases due to differences in measurement units or magnitudes. Common normalization techniques include min-max scaling and z-score standardization. Z-score standardization is preferred for algorithms like linear regression, logistic regression, and artificial neural networks. The standard score of a sample x is calculated as: z = (x - u) / s, where u is the mean of the training samples and s is the standard deviation of the training samples. It is done by:

```
From sklearn.preprocessing import StandardScaler

x_train=StandardScaler().fit_transform(x_train)

x_test=StandardScaler().trasform(x_test)
```

Only the independent features are transformed.

VISUALIZATION

Data visualization is a powerful tool that helps in presenting complex information in a visually appealing and understandable way. By transforming raw data into charts, graphs, maps, or interactive visual representations, it allows individuals to comprehend patterns, trends, and relationships that may otherwise go unnoticed.

Effective data visualization enhances decision-making processes by providing insights and facilitating the communication of findings. It enables data analysts, researchers, and businesses to convey their data-driven narratives more efficiently and engage their audience effectively. Whether it's exploring large datasets, tracking performance metrics, or conveying statistical trends, data visualization simplifies the understanding of information.

For this dataset, countplot of various features have been drawn to indicate the frequency of occurences of each feature. The features visualized are: Year of attack, cities that were attacked, attack type, target type, target subtype, weapon used,weapon subtype, target and name of the terror group. Seaborn library has been used, which has various useful visualization functions inbuilt. Commands used:

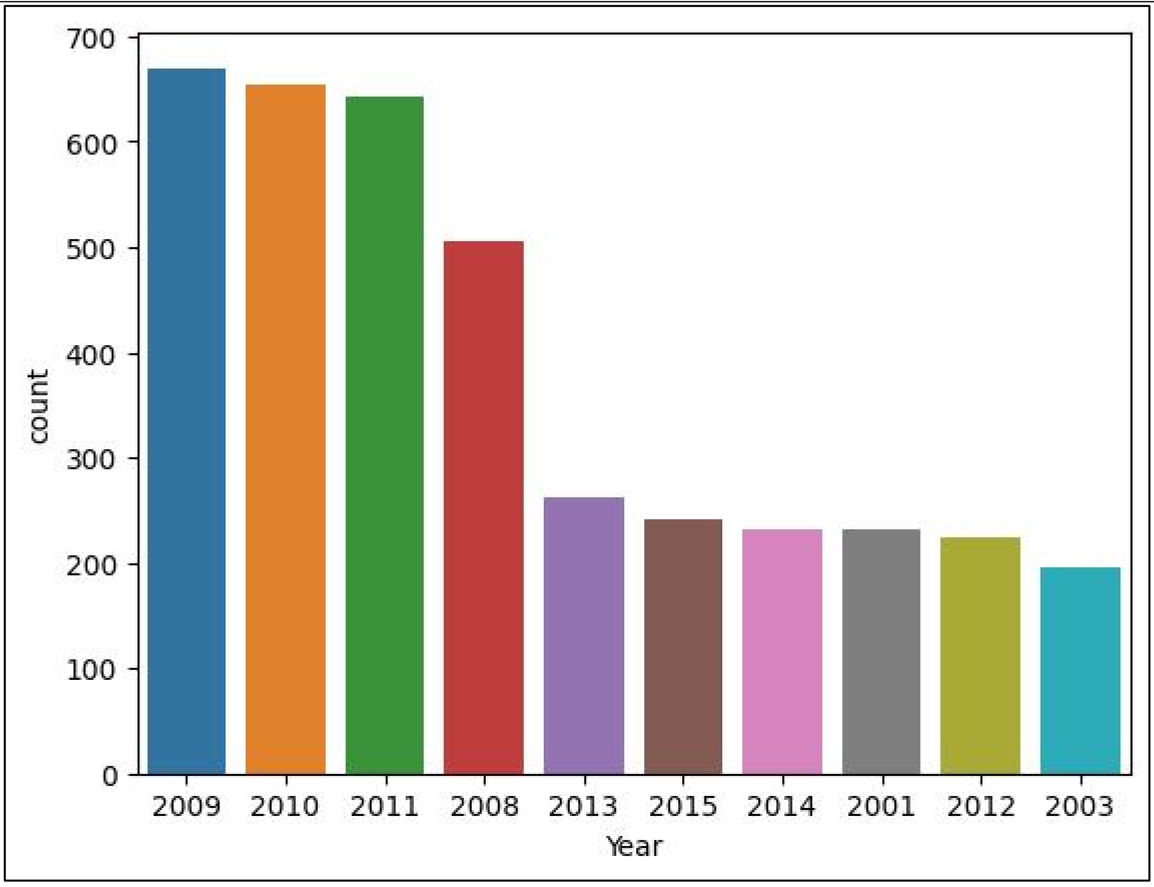
```
Import seaborn as sns
```

Feature and visualization:

Year:

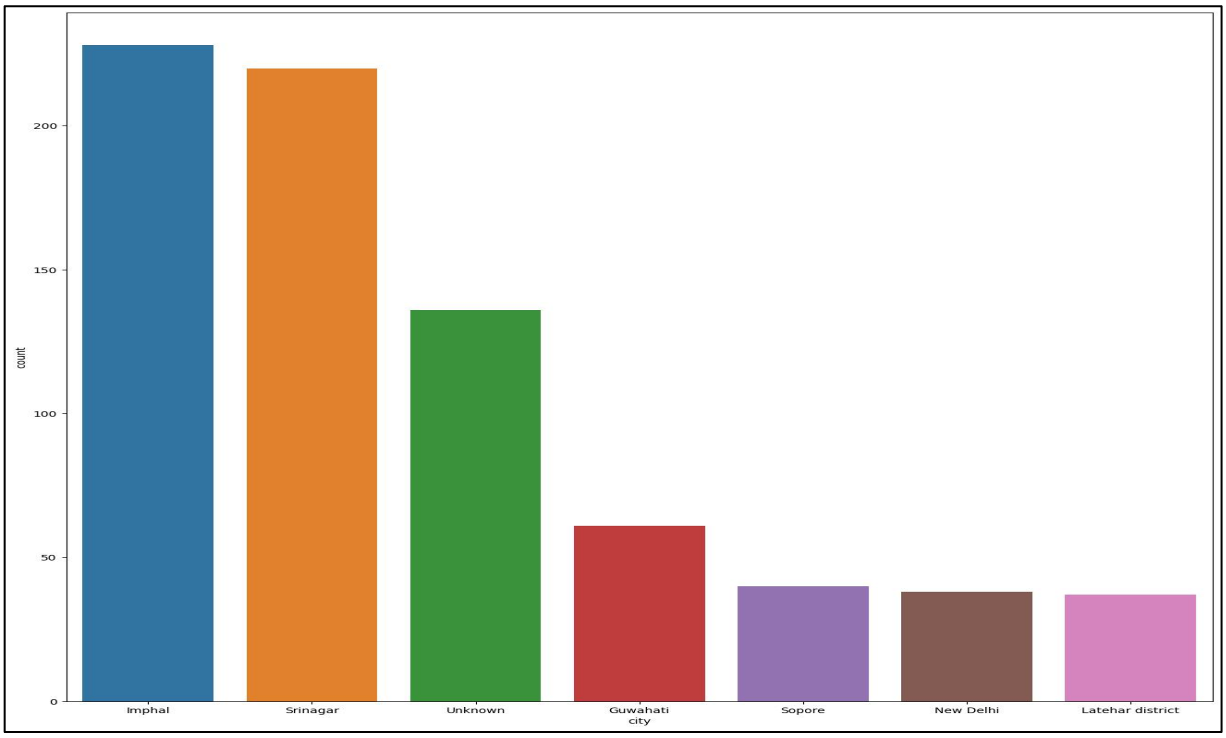
```
sns.countplot(x=df1['Year'],order=df1['Year'].value_counts().iloc[:10].index)
```

This gives the top 10 years which faced the attacks



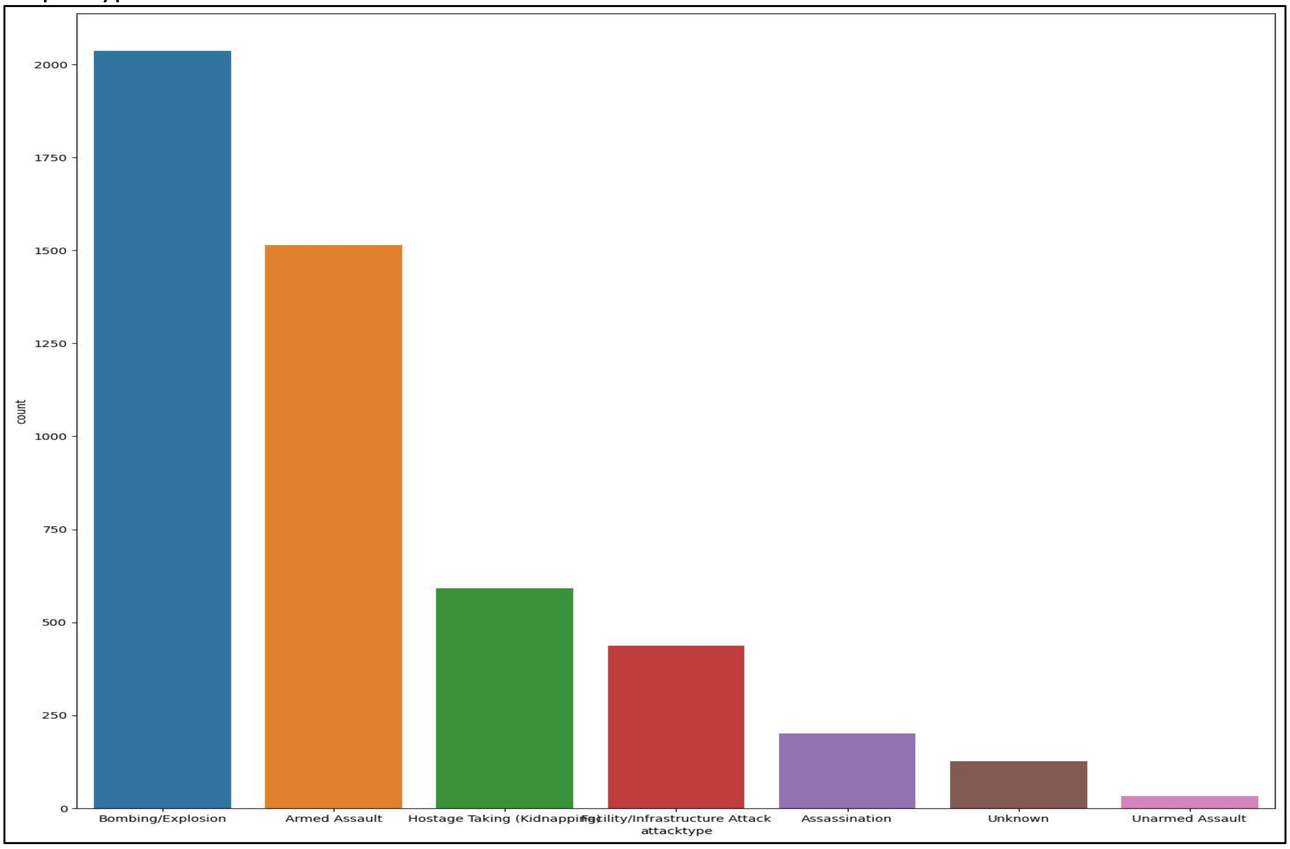
2009 suffered the most attacks

Top 7 most impacted cities:



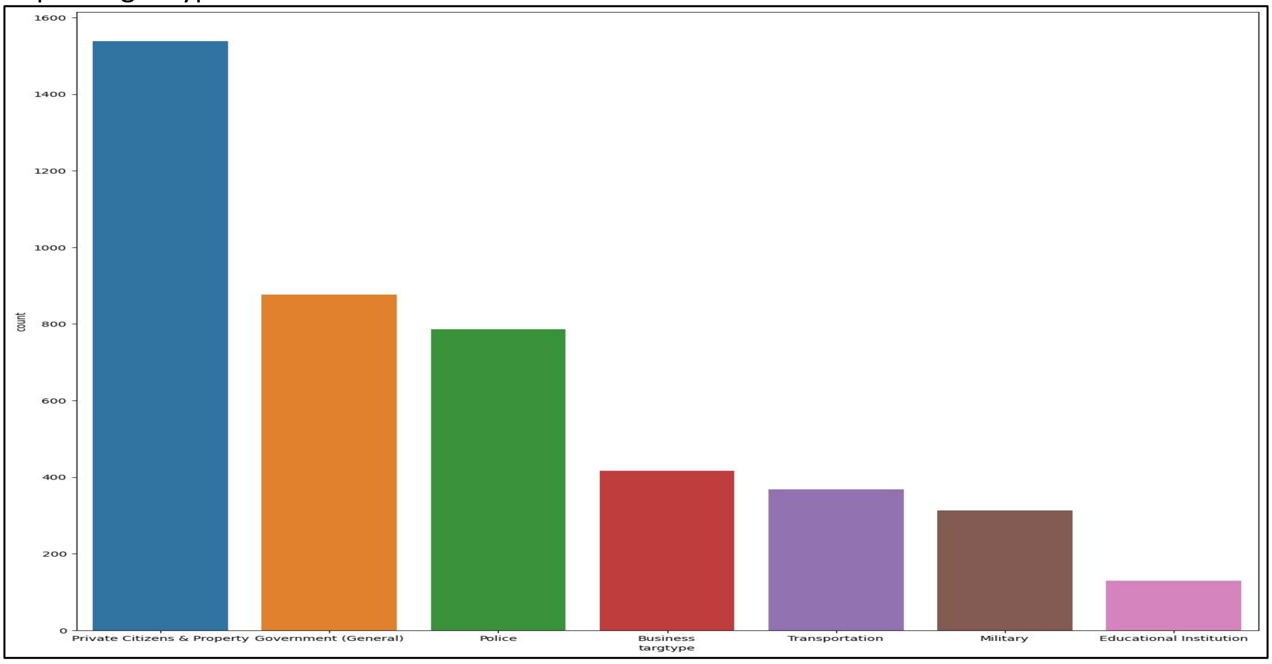
Imphal was targeted the most

Top 7 types of attack:



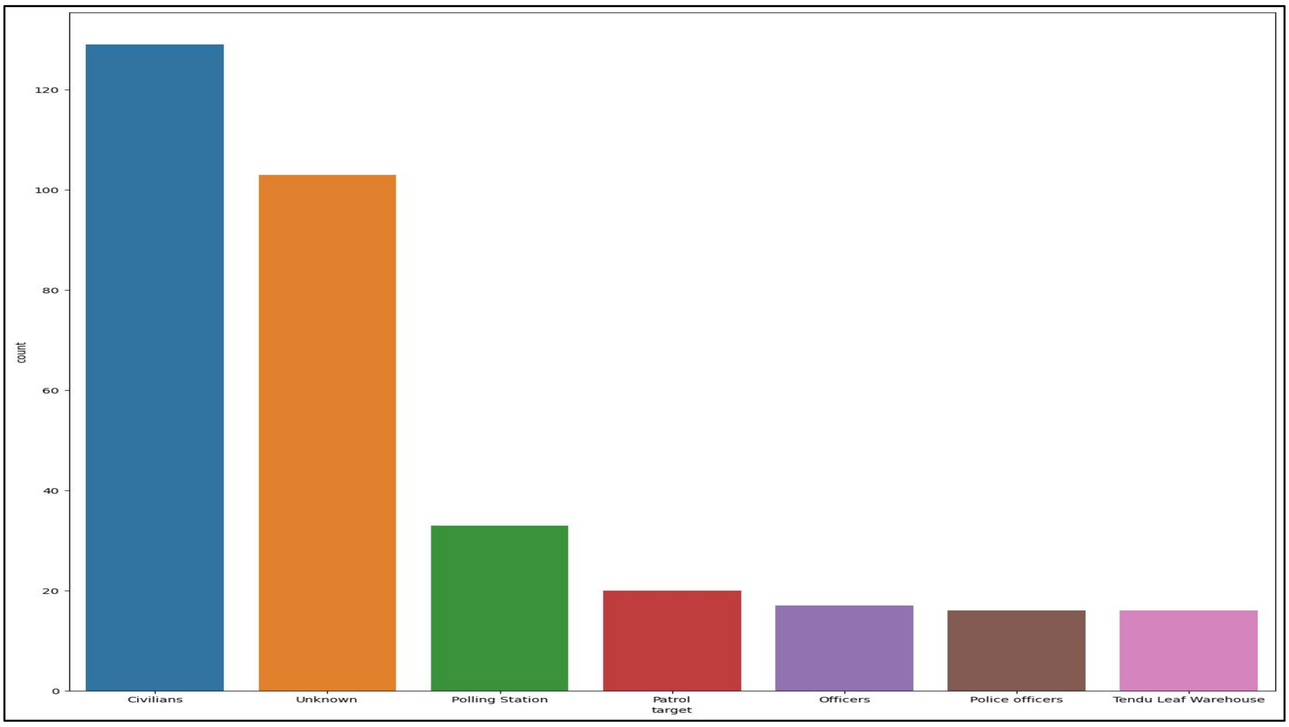
Most of the attacks were bombings, with weapons being the second most

Top 7 target types:



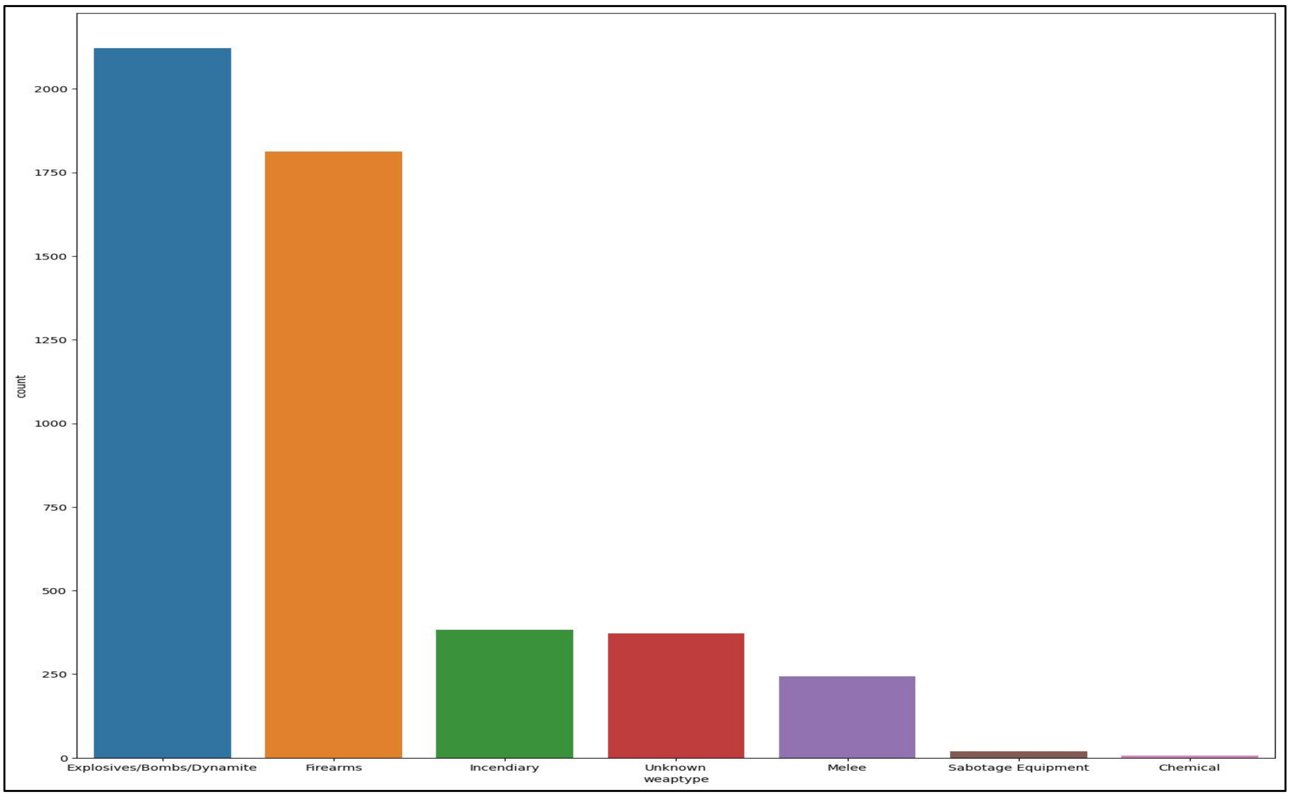
Private property was targeted the most, followed by government buildings

Types of targets:



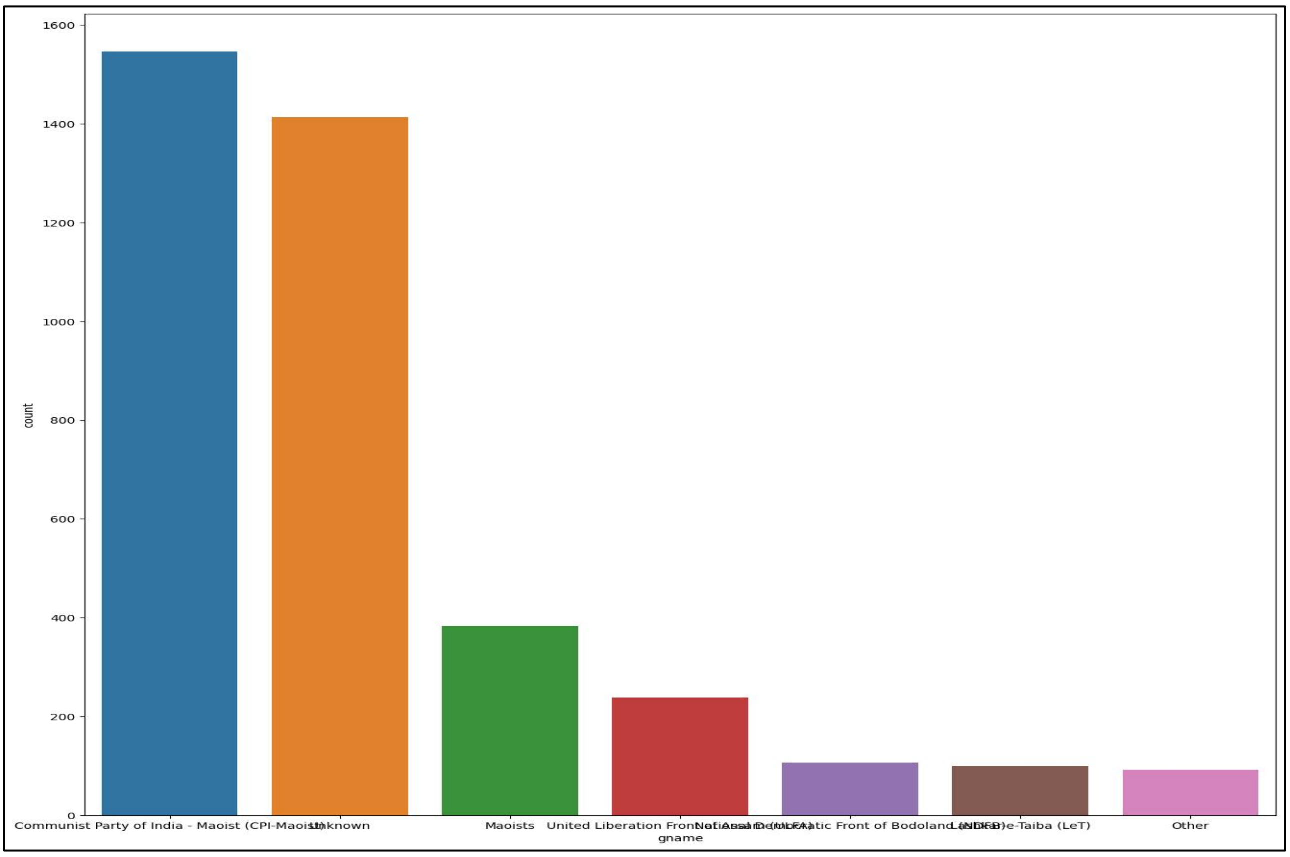
Civilians were affected the most

Types of weapons used:



Explosives were used the most, followed by firearms

Groups that led the attacks:



Most of the attacks were led by maoists

MAKING PREDICTIONS

1. Using deep learning with artificial neural networks:

Deep learning is a subfield of machine learning, where artificial neural networks (ANNs) play a crucial role. ANNs are inspired by the structure and functioning of the human brain. Deep learning models consist of multiple layers of interconnected artificial neurons, allowing them to process vast amounts of data and learn intricate patterns.

One of the key advantages of deep learning is its ability to automatically extract relevant features from raw data. Traditional machine learning methods often require manual feature engineering, but with deep learning, the neural network learns to identify important features on its own through a process called training.

Training deep learning models involves feeding them labeled data and optimizing their internal parameters through a process known as backpropagation. This iterative optimization allows the model to improve its predictions over time. The availability of massive computing power, along with frameworks like TensorFlow and PyTorch, has contributed to the widespread adoption of deep learning.

Although deep learning has achieved remarkable success, it also poses challenges. Training deep neural networks requires substantial computational resources and extensive labeled datasets. Overfitting, where the model memorizes the training data instead of generalizing well, is another issue that researchers address through regularization techniques.

Working of artifical neural networks:

Network Structure: ANNs consist of interconnected nodes called artificial neurons or simply "neurons." Neurons are organized into layers: an input layer, one or more hidden layers, and an output layer. The connections between neurons are represented by weighted edges.

Feedforward Process: Information flows through the network in a feedforward manner, starting from the input layer, passing through the hidden layers, and finally reaching the output layer. Each neuron receives inputs from the previous layer, multiplies them by their corresponding weights, and applies an activation function to produce an output.

Weights and Bias: The weights associated with the connections between neurons determine the strength of influence one neuron has on another. These weights are initially assigned random values and are adjusted during the learning process. Additionally, each neuron typically has a bias term that allows for fine-tuning of its activation.

Activation Function: The activation function of a neuron introduces non-linearities into the network. It determines whether a neuron should fire (i.e., produce an output signal) based on the weighted sum of its inputs. Common activation functions include sigmoid, tanh, ReLU, and softmax.

Learning and Training: ANNs learn by adjusting the weights and biases to minimize the difference between their predicted outputs and the desired outputs. This process is known as training or learning. Supervised learning methods, such as backpropagation, are commonly employed to update the weights by calculating gradients and propagating them backward through the network.

Backpropagation: Backpropagation is a widely used algorithm for updating the weights in multilayer ANNs. It calculates the gradient of the error with respect to each weight and adjusts them accordingly. This iterative process continues until the network achieves satisfactory performance on the training data.

Generalization: Once trained, ANNs can generalize their learned knowledge to make predictions or classify new, unseen inputs. The idea is that the network has learned patterns from the training data and can apply this knowledge to similar but previously unseen examples.

Optimizers:
Optimizers play a crucial role in training artificial neural networks (ANNs) by adjusting the weights and biases during the learning process. They help to optimize the network's performance and convergence speed. Here are some commonly used optimizers for ANNs:

Stochastic Gradient Descent (SGD): SGD is one of the simplest optimizers. It updates the weights by taking small steps in the direction of the negative gradient of the loss function with respect to the weights. It works well for large datasets but can be slow to converge.

Momentum: Momentum builds upon SGD by adding a momentum term that accumulates the previous gradients' influence. This helps accelerate learning through flat or noisy areas and helps overcome local minima. It has a dampening effect on oscillations and can converge faster than plain SGD.

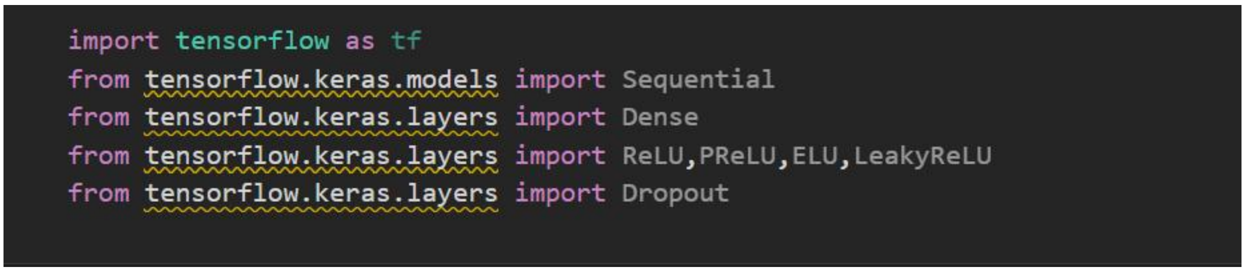
Adaptive Moment Estimation (Adam): Adam combines the benefits of both AdaGrad and RMSProp optimizers. It adapts the learning rate for each parameter based on both the first and second moments of the gradients. Adam is popular due to its efficiency and ability to handle sparse gradients.

Adaptive Gradient Algorithm (AdaGrad): AdaGrad adapts the learning rate for each parameter based on the historical gradients. It assigns larger learning rates to parameters associated with infrequent features and smaller learning rates to frequently occurring ones. However, AdaGrad's learning rate can become too small over time, leading to premature convergence.

Root Mean Square Propagation (RMSProp): RMSProp addresses the diminishing learning rate issue of AdaGrad by using an exponentially decaying average of squared gradients. It divides the learning rate by the square root of the accumulated squared gradients. This optimizer performs well in practice and helps converge faster than AdaGrad.

For regression problems, the activation function most preferred is ReLU, and adam optimizer.
The function used for backpropagation is Mean Squared Error (MSE)

Tensorflow library, along with keras is used for training neural networks.



```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import ReLU,PReLU,ELU,LeakyReLU
from tensorflow.keras.layers import Dropout
```

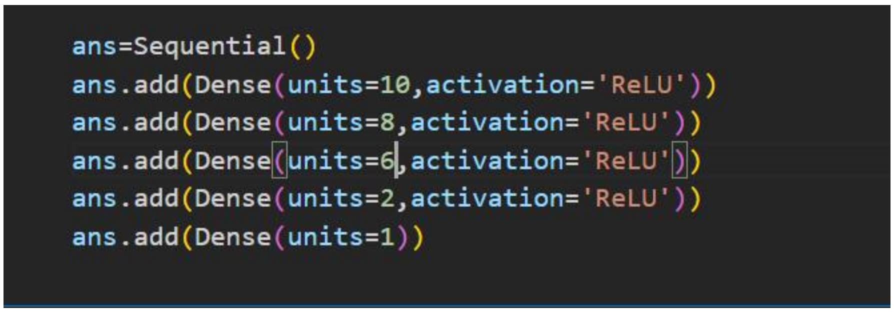
The Sequential model is a linear stack of layers used for building neural networks. It's a straightforward and commonly used model type that allows you to build deep learning models by stacking layers on top of each other.

The Dense layer is a fully connected layer that applies an affine transformation to the input. It is one of the most commonly used layer types in deep learning models. It implements the operation: output = activation(dot(input, kernel) + bias), where:

input represents the input tensor to the layer.
kernel refers to the weight matrix (learnable parameters) of the layer.
bias represents the bias vector (learnable parameters) of the layer.
activation is the element-wise activation function applied to the layer's output.

Dropout is a regularization technique used to prevent overfitting in neural networks. It randomly sets a fraction of input units to 0 at each update during training time, which helps to reduce the interdependencies between neurons.

The Dropout layer in Keras is typically inserted between dense layers or convolutional layers. It randomly sets a fraction of inputs to 0 during training, and the remaining inputs are scaled by the inverse of the dropout rate to maintain the expected sum of inputs. During inference or prediction, no units are dropped, but the output values are scaled down by the dropout rate to ensure consistent behavior.

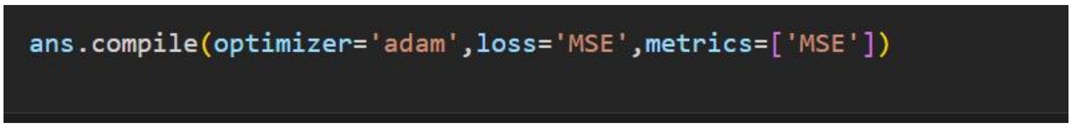


```
ans=Sequential()
ans.add(Dense(units=10,activation='ReLU'))
ans.add(Dense(units=8,activation='ReLU'))
ans.add(Dense(units=6,activation='ReLU'))
ans.add(Dense(units=2,activation='ReLU'))
ans.add(Dense(units=1))
```

The first layer is the input layer. Since the training dataset has 10 features, the number of neurons in the input layer is 10, along with ReLU as activation function.

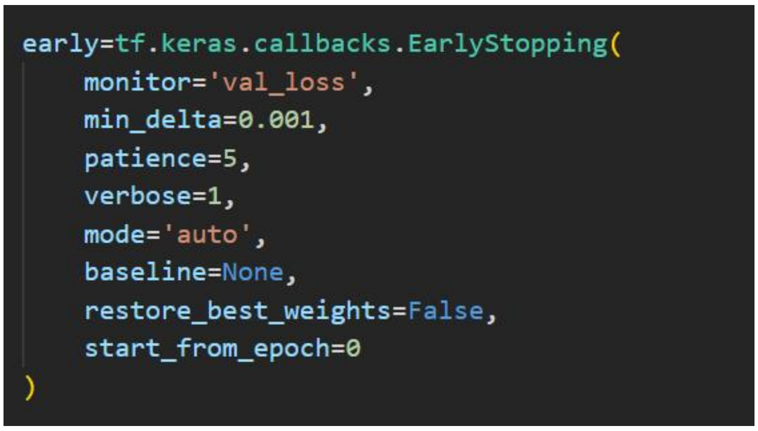
The second layer is the first hidden layer. It has 8 neurons. The number of eurons in each layer is reduced gradually. The activation function used is ReLU

The last layer is the output layer, which produces the required output. It has only 1 neuron, since it produces just one output for every row. The activation function used is linear activation, which is implemented by default.



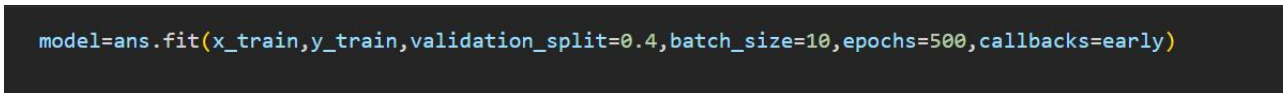
```
ans.compile(optimizer='adam',loss='MSE',metrics=['MSE'])
```

The model is then compiled using adam optimizer and MSE as loss function for back propagation.



```
early=tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.001,
    patience=5,
    verbose=1,
    mode='auto',
    baseline=None,
    restore_best_weights=False,
    start_from_epoch=0
)
```

Early stopping is a technique commonly used in machine learning, including deep learning with Keras, to prevent overfitting and determine the optimal number of training epochs. It helps to stop the training process early when the model's performance on a validation set stops improving or starts degrading. Here, the metric “val_loss” is monitor as a factor for early stopping.

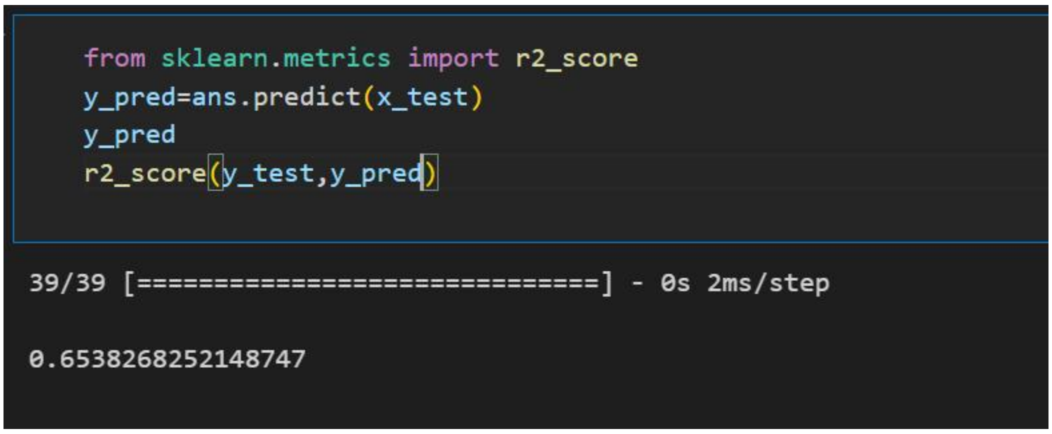


```
model=ans.fit(x_train,y_train,validation_split=0.4,batch_size=10,epochs=500,callbacks=early)
```

The model is then fit. The datasets x_train and y_train are used for fitting and training.
One entire cycle of forward and back propagation is called an epoch. Here, 500 epochs are used.
Batch size refers to the number of rows being used as input at once. Here, batch size of 10 is used.

Validation split refers to the portion of the training data that is set aside for validation during model training. It allows you to evaluate your model's performance on a separate dataset that was not used for training. Here validation split of 0.4, I.e, 40% is used.

The model is then trained, until the conditions in the earlystopping function are met.



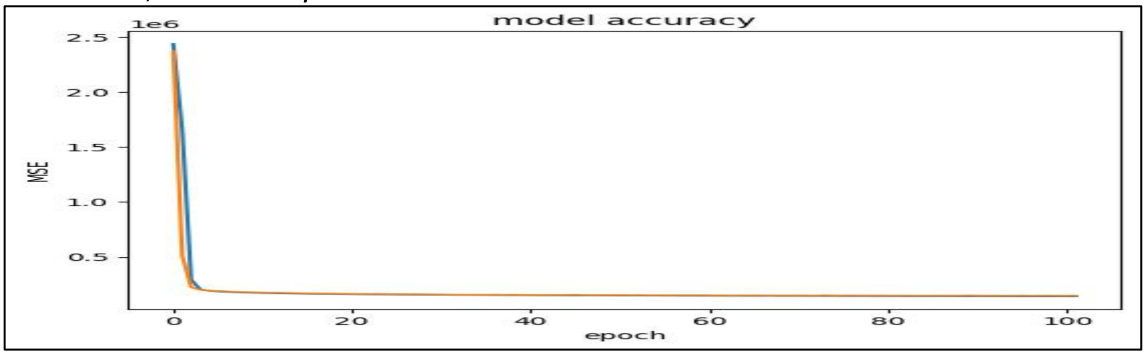
```
from sklearn.metrics import r2_score
y_pred=ans.predict(x_test)
y_pred
r2_score(y_test,y_pred)
```

```
39/39 [==============================] - 0s 2ms/step

0.6538268252148747
```

The Sequential model provides a simple way to build and train neural networks with a linear stack of layers. Once you have trained your model, you can use the predict() method of the Sequential model to make predictions on new data.

The R-squared (R2) score, also known as the coefficient of determination, is a commonly used metric for evaluating regression models. It represents the proportion of the variance in the dependent variable that is predictable from the independent variables.
The r2_score function calculates the R2 score by comparing the variation explained by the model to the total variation in the data. The closer the R2 score is to 1, the better the model fits the data, indicating that it explains a larger portion of the variance. Conversely, an R2 score close to 0 suggests that the model does not capture much of the variance in the data.
From this model, an accuracy of 65% is obtained.

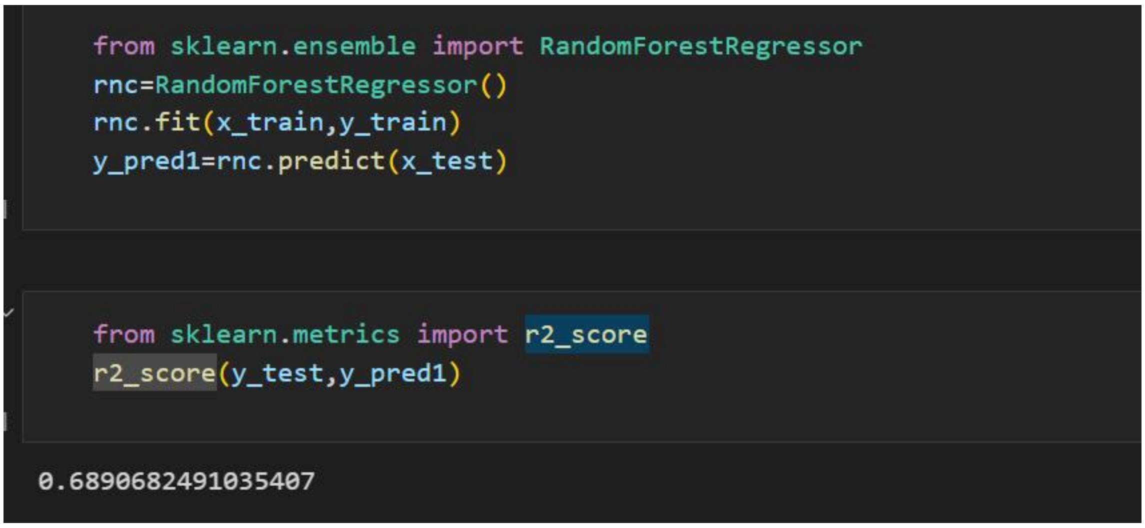


2. Using random forest regressor:

Random forest is an ensemble learning method that combines multiple decision trees to make predictions.

Random forest regressor builds an ensemble of decision trees, where each tree is trained on a random subset of the training data and features. This randomness helps in reducing overfitting and improving generalization.

Random forest regressor applies bagging, which means that each tree is trained on a different bootstrap sample drawn from the original training data. Additionally, at each split within a tree, only a random subset of features is considered. This combination of random sampling reduces the correlation between the trees and leads to more diverse and robust predictions.



```
from sklearn.ensemble import RandomForestRegressor
rnc=RandomForestRegressor()
rnc.fit(x_train,y_train)
y_pred1=rnc.predict(x_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred1)
```

```
0.6890682491035407
```

Using random forest, an accuracy of 69% is obtained.

Therefore, artificial neural networks gives an accuracy of 65%, while random forests gives an accuracy of 69%.

SUGGESTED IMPROVEMENTS

1. Using Principal Component Analysis for reducing number of input features:

Principal Component Analysis (PCA) is a popular technique for dimensionality reduction. It transforms high-dimensional data into a lower-dimensional representation while preserving the most important information or variability in the original data.

Steps to perform dimensionality reduction using PCA:

Standardize the Data: Start by standardizing the features of your dataset to have zero mean and unit variance. This step ensures that all features are on the same scale and prevents variables with larger variances from dominating the PCA process.

Compute Covariance Matrix or Correlation Matrix: Calculate the covariance matrix or correlation matrix of the standardized data. The covariance matrix measures the pairwise covariances between features, while the correlation matrix represents the pairwise correlations. Both matrices provide insights into the relationships between the features.

Perform Eigendecomposition: Perform eigendecomposition on the covariance matrix or correlation matrix. This step yields a set of eigenvalues and corresponding eigenvectors. The eigenvalues represent the amount of variance explained by each eigenvector (principal component), and the eigenvectors represent the directions or axes along which the data varies the most.

Select Principal Components: Sort the eigenvalues in descending order and choose the top-k eigenvectors (principal components) based on the desired number of dimensions or the cumulative explained variance ratio. The cumulative explained variance ratio represents the proportion of the total variance accounted for by the selected principal components. Higher cumulative explained variance indicates more information retained from the original data.

Project Data onto Lower-dimensional Space: Project the original data onto the selected principal components to obtain the reduced-dimensional representation. Each data point is transformed by multiplying it with the transpose of the selected principal components.

By retaining only a subset of the most informative principal components, the dimensionality of the data is reduced. The reduced-dimensional representation captures the main patterns and structure of the original data while discarding less significant variations.

2. Using K-Means clustering:

K-means clustering is used for partitioning data into distinct groups or clusters. It aims to find K centroids that represent the centers of these clusters, where K is a user-defined parameter.

How K-Means clustering works:

Initialization: Randomly select K data points as initial centroids. These centroids serve as the starting points for the clustering process.

Assignment: Assign each data point to the nearest centroid based on a distance metric, typically Euclidean distance. This step forms K clusters, with each data point belonging to the cluster whose centroid it is closest to.

Update: Recalculate the centroids of the clusters by taking the mean of the data points assigned to each cluster. This step moves the centroids closer to the center of their respective clusters.

Iteration: Repeat the assignment and update steps until convergence or a specified number of iterations. Convergence occurs when the centroids no longer move significantly between iterations or when the algorithm reaches the maximum number of iterations.

Final Result: Once convergence is reached, the algorithm outputs the final K clusters, each represented by its centroid. The data points are now grouped based on similarity, with points in the same cluster being more similar to each other than to those in other clusters.

K-Means clustering can be used on the latitude and longitude columns to group them into clusters, then doing predictive analysis on the clusters.