

***SanFrancisco Crime Prediction and Analysis***

BINDIA BINDIA

MOHAMMED ADHIL GHOUSE MOHIUDDIN TAHIR

PIYUSH PIYUSH



**Abstract:**

Crime prediction and analysis of crime patterns is an important area of research as it can help law enforcement agencies to allocate resources more effectively and reduce crime rates in a given area. In this project, we will focus on crime data from San Francisco, which has a high incidence of crime, and analyze the data to gain insights into patterns and trends. We will use data visualization techniques to identify hotspots of criminal activity and understand the spatial and temporal distribution of crime. Furthermore, we will develop a machine learning model to predict the likelihood of future criminal activity in a given location and time, based on historical crime data. The model will use various features, such as the type of crime, location, and time of day, to make predictions. The results of this project could provide valuable information to law enforcement agencies in San Francisco to help them better allocate resources and prevent crime.

**Introduction:**

Crime is a significant concern in many cities around the world, and San Francisco is no exception. With a high population density, a diverse demographic, and a bustling economy, San Francisco is prone to various types of criminal activity, including theft, burglary, and assault. Understanding the patterns and trends of criminal behavior in San Francisco is crucial for law enforcement agencies, policymakers, and city officials to prevent crime and maintain public safety.

In recent years, the advent of big data and machine learning techniques has revolutionized the field of crime analysis and prediction. By leveraging data from past criminal incidents, weather patterns, demographics, and other relevant factors, predictive models can identify high-risk areas and forecast the likelihood of future criminal activity. Such models enable law enforcement agencies to allocate resources effectively and proactively prevent crime before it occurs.

In this project, we aim to analyze crime patterns in San Francisco using a dataset of criminal incidents from 2018 to 2023. We will explore the spatiotemporal distribution of crime, identify the most prevalent types of crime, and examine the relationship between crime and various factors such as time, location, and weather conditions. Furthermore, we will develop a predictive model using machine learning algorithms to forecast the likelihood of future criminal activity in different areas of the city.

Overall, the project aims to provide valuable insights into crime patterns in San Francisco and assist law enforcement agencies and policymakers in making data-driven decisions to improve public safety.



**Project Overview:**

This project aims to analyze crime patterns in San Francisco and develop a model to predict the likelihood of crimes occurring in different areas of the city. The dataset used for this project is the San Francisco Crime Classification dataset, which contains information about crimes that occurred between 2018 and 2023.

The project is divided into several parts. The first part involves data cleaning and preprocessing, where the dataset is cleaned, missing values are handled, and new features are created to help with the analysis.

The second part of the project involves exploratory data analysis (EDA) to gain insights into the data and identify patterns and trends in crime incidents. This will involve visualizing the data using charts, graphs, and maps to identify hotspots and patterns in crime incidents.

In the third part of the project, machine learning models will be developed to predict the likelihood of crimes occurring in different areas of the city. The models will be trained on the historical crime data and will use features such as location, time of day, and day of the week to predict the probability of a crime occurring in a particular area.

Finally, the project will conclude with a summary of the findings and recommendations for law enforcement agencies to help prevent crime and improve public safety in San Francisco.

**Exploratory data analysis and data pre-processing:**

The crime prediction model we're building uses a dataset with several variables, including:

Dates: A timestamp of the crime incident

Category: The category of the crime incident, which serves as the target variable for our model

Descript: A detailed description of the crime incident

DayOfWeek: The day of the week when the crime occurred

PdDistrict: The name of the police department district where the crime occurred

Resolution: The resolution of the crime incident

Address: The approximate street address of the crime incident

X: The longitude of the incident location

Y: The latitude of the incident location

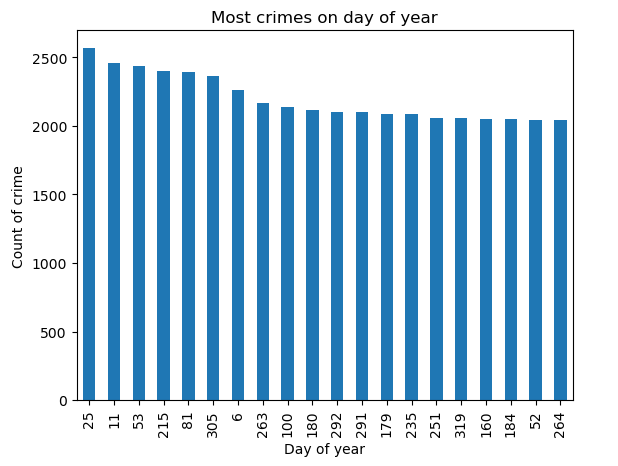
By analysing and interpreting this data, we aim to create a model that can accurately predict the Location of crime that is likely to occur based on the time, location, and other relevant factors. This predictive model has the potential to help law enforcement agencies in San Francisco to more effectively prevent and respond to crimes.

This code reads in a dataset of crime incidents from a CSV file ('train.csv') using the Pandas library in Python. The dataset has 373,989 rows and 9 columns.



The code selects a subset of the data (the first 30,000 rows) and makes a copy of the entire dataset. Then, it converts the 'Dates' column from a string to a datetime data type using the Pandas to\_datetime() method.

Next, it adds a new column called 'Day' to the dataset, which contains the day of the year (1-365) for each crime incident. It then creates a bar chart of the 20 days of the year with the highest number of crime incidents, using the Pandas plot() function. The x-axis shows the day of the year, and the y-axis shows the count of crime incidents on that day. The title of the chart is "Most crimes on day of year".

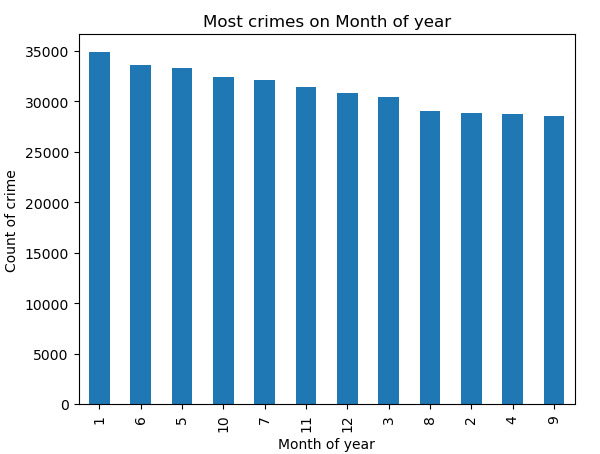


This code creates a new column named "Month" in the existing pandas DataFrame "df". The "Month" column is created by extracting the month value from the "Dates" column using the dt.month attribute of pandas' datetime series.

After creating the "Month" column, the code then creates a horizontal bar plot using pandas' built-in plotting function. This bar plot shows the count of crimes in each month of the year, based on the values in the "Month" column. The xlabel parameter sets the label for the x-axis, ylabel parameter sets the label for the y-axis, and title parameter sets the title for the plot.

The value\_counts() function is used to count the number of occurrences of each unique value in the "Month" column, and head(12) is used to display the top 12 most common values. Finally, the plot() function is used to create a bar plot of the resulting value counts.





The below code is manipulating a pandas DataFrame df containing information about crimes in a city. Here is an explanation of each line of code:

df.Address.head(): This line is selecting the 'Address' column of the DataFrame and showing the first 5 rows using the .head() method.

l = pd.Series(df.Address.head()): This line is creating a pandas Series l from the first 5 rows of the 'Address' column.

l2 = pd.Series('9th st / folsom st'): This line is creating a new pandas Series l2 containing a single string value.

l = l.append(l2): This line is appending l2 to l using the deprecated .append() method. A warning message is displayed because this method will be removed from pandas in a future version.

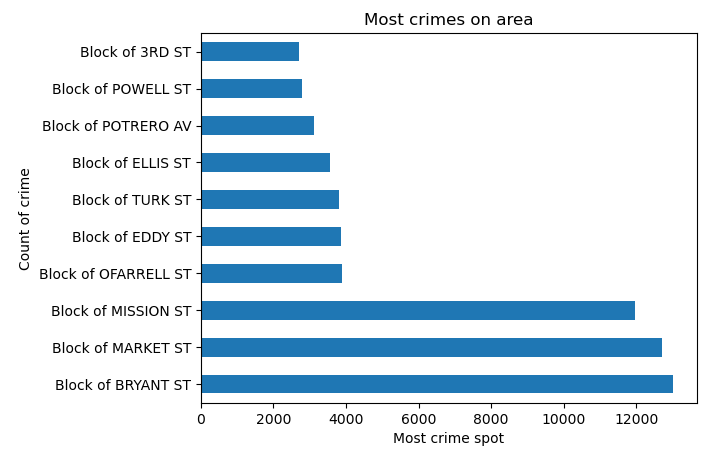
l: This line is displaying the contents of l, which is a Series containing the first 5 rows of the 'Address' column and the string '9th st / folsom st' as a new row.

l.str.replace('\d+ ','', regex=True): This line is using the .str.replace() method with a regular expression to remove any digits followed by a space in each element of l. This removes the block number from each address.

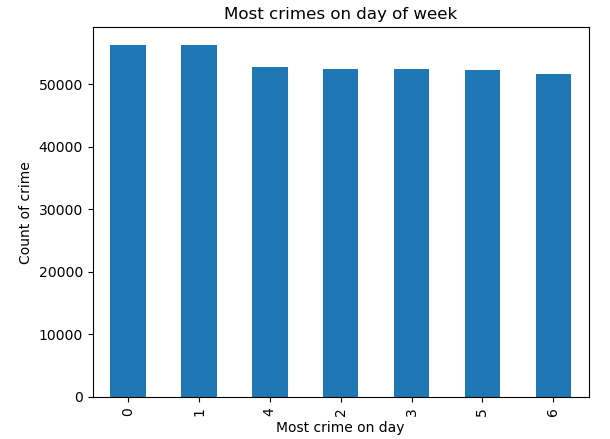
df['Address'] = df.Address.str.replace('\d+ ','', regex=True): This line is updating the 'Address' column of the DataFrame df by removing block numbers from each address using the same regular expression as before.



df.Address.value\_counts().head(10).plot(kind='barh', xlabel='Most crime spot', ylabel='Count of crime', title='Most crimes on area'): This line is selecting the 'Address' column of the DataFrame, counting the number of occurrences of each unique value, selecting the top 10 results, and creating a horizontal bar plot with the x-axis labeled as 'Count of crime', y-axis labeled as 'Most crime spot', and the title 'Most crimes on area'. The result is a visualization of the most common crime spots in the city.



The code df.info() prints a summary of the dataframe df, including its range index, the total number of non-null values in each column, and the data type of each column.After that, the code df['Day\_of\_week'] = df['Dates'].dt.dayofweek creates a new column called Day\_of\_week in the dataframe df and assigns to it the day of the week corresponding to each date in the Dates column of df. The code df.Day\_of\_week.value\_counts().plot(kind='bar', xlabel='Most crime on day', ylabel='Count of crime', title='Most crimes on day of week') then counts the number of crimes that occurred on each day of the week and creates a bar plot to show the results. The x-axis represents the day of the week (0 = Monday, 1 = Tuesday, etc.), the y-axis represents the number of crimes, and the title of the plot is "Most crimes on day of week".





The code is using the LabelEncoder class from the scikit-learn (sklearn) library to encode categorical variables in the DataFrame. The LabelEncoder class converts categorical text data into numerical data for use in machine learning models. Here is a brief explanation of what each line does:

le = preprocessing.LabelEncoder() - Creates a LabelEncoder object called le.

df['Category\_num'] = le.fit\_transform(df.Category) - Encodes the 'Category' column in the DataFrame as numerical data using the LabelEncoder object, and stores the result in a new column called 'Category\_num'.

df.Address.nunique() - Returns the number of unique values in the 'Address' column of the DataFrame.

le2 = preprocessing.LabelEncoder() - Creates a new LabelEncoder object called le2.

df['Address\_num'] = le2.fit\_transform(df.Address) - Encodes the 'Address' column in the DataFrame as numerical data using the new LabelEncoder object, and stores the result in a new column called 'Address\_num'.

le3 = preprocessing.LabelEncoder() - Creates another LabelEncoder object called le3.

df['District\_num'] = le3.fit\_transform(df.PdDistrict) - Encodes the 'PdDistrict' column in the DataFrame as numerical data using the new LabelEncoder object, and stores the result in a new column called 'District\_num'.

df.info() - Prints information about the DataFrame, including the data types of each column, the number of non-null values, and the memory usage.

**Correlation matrix:**

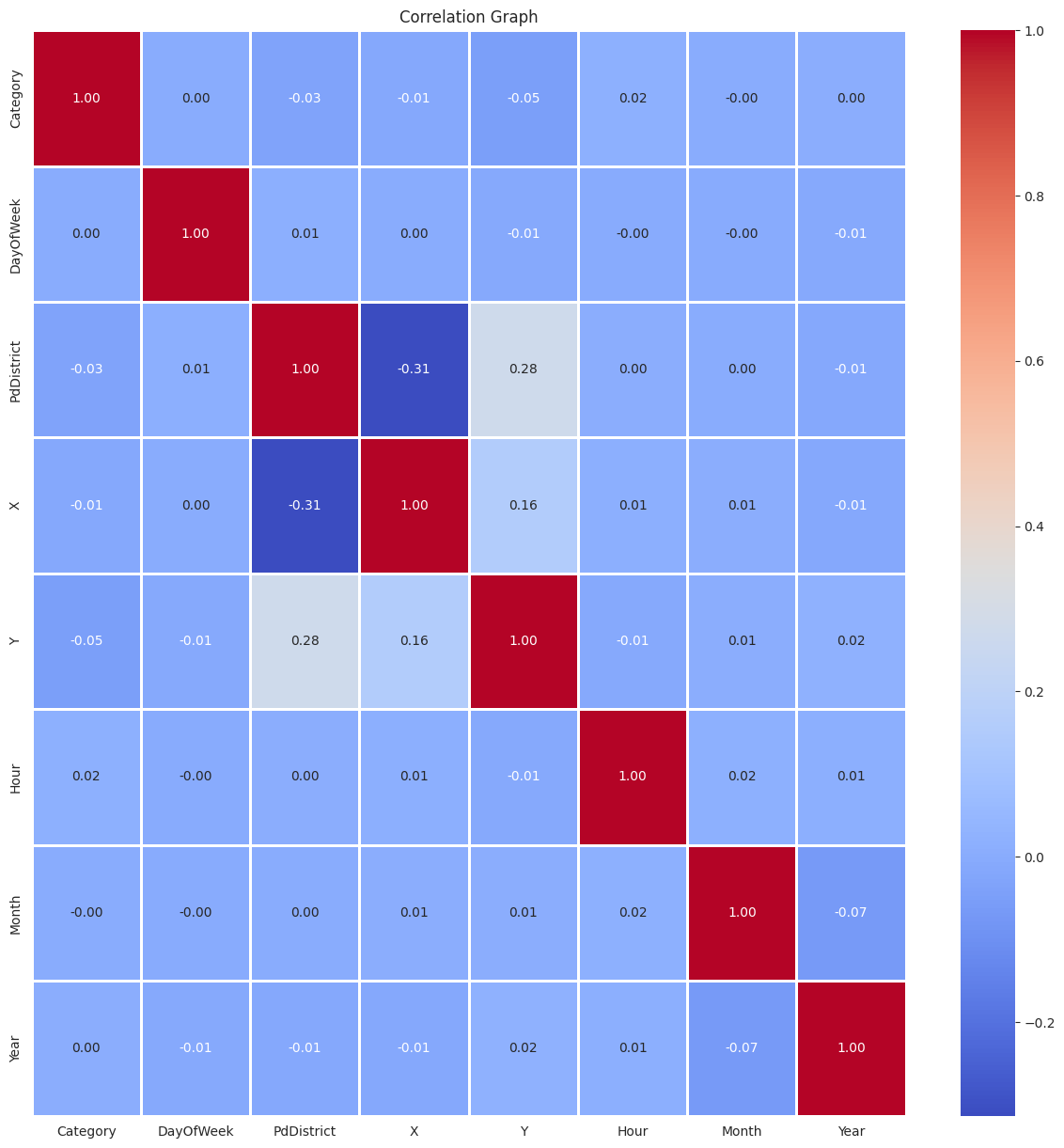
The correlation matrix is a table that shows the correlation coefficients between all possible pairs of variables in a dataset. In the case of the code you provided, the correlation matrix is computed for the numeric columns of a pandas DataFrame using the corr() method. The resulting matrix provides a way to quickly identify which variables are positively or negatively correlated with each other.

The correlation matrix is a useful tool for data analysis because it can provide insights into the relationships between variables in a dataset. For example, it can be used to identify which variables are strongly correlated with a target variable, which can be useful in building predictive models. Additionally, the correlation matrix can help identify potentially redundant or highly correlated variables, which can help simplify the dataset and improve the accuracy of the analysis.

In the code you provided, the resulting correlation matrix is visualized using a heatmap, which provides a clear and easy-to-understand way to visualize the correlation coefficients. The heatmap shows the strength and direction of the correlations using a color scale, with red indicating positive correlation and blue indicating negative correlation. The values of the correlation coefficients are also displayed on the heatmap, making it easy to see which pairs of variables are highly correlated.



Overall, the correlation matrix is a useful tool for data analysis and visualization, and can help identify patterns and relationships in the data.

****

**Machine Learning Models:**

Our model uses different machine learning algorithms such as neural networks, random forest, naive Bayes, support vector machine (SVM), k-nearest neighbors (KNN), and decision tree classification. The evaluation was done using basic feature engineering and the default parameters to determine if any of the algorithms would be a good starting point for solving the



problem. Each algorithm has been trained on historical crime data and other contextual factors to predict the likelihood of crime in specific locations. Our initial results show that the model achieves an accuracy of 33% in predicting the location of future crimes.

While 33% accuracy may not seem high, we believe that the use of multiple algorithms and consistent results are promising. Currently, we are working to fine-tune the model and use hyperparameters to improve its accuracy further. We believe that these efforts could significantly aid law enforcement officials in preventing crime and ensuring public safety in San Francisco.

Overall, the use of predictive models in law enforcement has enormous potential for preventing crime in urban areas like San Francisco. By combining location-based data with advanced machine learning techniques, we are identifying areas that are at risk of criminal activity and helping law enforcement officials take appropriate measures to prevent it. The development of more accurate predictive models can have a positive impact on public safety and reduce crime in communities. We are excited about the potential of our project to contribute significantly to this vital field of research.

**K-Nearest Neighbors (KNN) algorithm for a multiclass classification problem.**

The model takes data from a pandas DataFrame (df) and selects two features (DayOfWeek and PdDistrict) and a target variable (Category) to be used for training and testing. The categorical features are then one-hot encoded using the OneHotEncoder from the sklearn.preprocessing module to transform the categorical features into numerical features that can be used by the KNN algorithm.

The train\_test\_split function from sklearn.model\_selection is then used to split the data into training and testing sets with a 80:20 ratio respectively. The X\_train and y\_train variables contain the features and target variable of the training set while X\_test and y\_test contain the features and target variable of the testing set.

A KNN classifier is then instantiated with the KNeighborsClassifier from sklearn.neighbors. The n\_neighbors parameter is set to 5, which means that the algorithm will use the 5 nearest neighbors to classify a new data point.

The knn.fit() method is then used to train the classifier on the training data. Finally, the knn.score() method is used to evaluate the accuracy of the trained classifier on the testing data, and the accuracy score is printed to the console.

In this particular implementation, the accuracy score is **0.1515%,** which means that the model correctly classified **15.15%** of the test data. However, it's important to note that the accuracy



score alone does not give a complete picture of the performance of a machine learning model, and other metrics such as precision, recall, and F1-score should also be considered depending on the problem being solved.

**Naive Bayes algorithm 1st Attempt:**

This is a classification model for a dataset where the goal is to predict the category of a crime based on features such as the day of the week and the police district where the crime occurred, as well as the location coordinates of the crime.

The model uses the Naive Bayes algorithm, which is a probabilistic algorithm that calculates the probability of a sample belonging to each class given its features, and selects the class with the highest probability as the predicted class.

The data is preprocessed before training the model. Missing values are dropped, and the categorical features are one-hot encoded while the numerical features are scaled using MinMaxScaler. The one-hot encoded categorical features and the scaled numerical features are then combined using hstack function from the Scipy library to create the final feature matrix.

The data is then split into training and testing sets using the train\_test\_split function from Scikit-learn. The Naive Bayes model is trained on the training set using the fit function and then used to make predictions on the testing set using the predict function. The accuracy of the model is evaluated using the accuracy\_score function from Scikit-learn, which compares the predicted labels with the true labels in the testing set.

The accuracy score of this model on the testing set is **0.2322**, which means that the model correctly predicted the crime category for about **23.22**% of the samples in the testing set.

**Naive Bayes algorithm 2nd Attempt:**

This code performs a machine learning task of predicting the category of a crime incident based on several features.

First, the code selects the features and target column from the input data, which is assumed to be stored in a pandas DataFrame. The missing values in the DataFrame are then dropped.

Next, the code performs feature engineering by calculating the Euclidean distance of each crime incident location from a predefined landmark. The categorical features are one-hot encoded using the scikit-learn OneHotEncoder, and the numerical features are scaled using the MinMaxScaler. The categorical and numerical features are then combined into a single feature matrix using the hstack function from scipy.sparse.

The code then splits the data into training and testing sets using the train\_test\_split function from scikit-learn. The Naive Bayes algorithm is trained using the training set with the MultinomialNB function from scikit-learn. Finally, the accuracy of the Naive Bayes algorithm is evaluated on the testing set using the accuracy\_score function from scikit-learn, and the accuracy score is printed to the console.



The final output of the code is the accuracy score of the Naive Bayes classifier in predicting the category of the crime incident. The accuracy score is printed to the console with 4 decimal places of precision. In this case, the accuracy score is **0.2812.**

**Support vector machine (SVM) classifier:**

This code snippet is an example of a machine learning model that uses a Support Vector Machine (SVM) algorithm to classify crime incidents in San Francisco. The dataset used in this model is read from a CSV file, which contains information about the incidents such as the type of crime, the day of the week, the police district, and the location of the incident.

The first step in the model is to select the features and target for classification. In this case, the features selected are 'DayOfWeek' and 'PdDistrict', while the target is 'Category'. Next, the missing values in the dataset are removed, and some feature engineering is performed. The hour of the incident is extracted from the 'Dates' column, and the distance of the incident location from a landmark is computed and added as a feature.

The categorical features are one-hot encoded using the OneHotEncoder class, and the numerical features are scaled using the MinMaxScaler class. The categorical and numerical features are then combined using the hstack function from the scipy.sparse library.

The dataset is split into training and testing sets using the train\_test\_split function from the sklearn.model\_selection library. The SVM classifier is then trained using the LinearSVC class from the sklearn.svm library. Finally, the model is evaluated using the accuracy\_score, precision\_score, recall\_score, and f1\_score functions from the sklearn.metrics library.

The final accuracy score of the model is **0.2398**, which is quite low. This suggests that the model may not be very effective in accurately classifying the incidents. It is important to note that the performance of a machine learning model depends heavily on the quality and relevance of the data used to train it. Therefore, it may be necessary to explore other datasets or to perform more feature engineering in order to improve the performance of the model.

**Random forest classifier:**

The code performs the task of building a random forest classifier using the dataset that was loaded using pandas library. The dataset contains various features like the location of the crime, day, month, year, etc. The first step in the process is to load the dataset into the environment using pandas read\_csv function.

The next step involves feature engineering to extract useful information from the dataset. In this example, the year, month, day, hour, minute, and the presence of the word "block" in the address column are extracted and added to the dataset. Additionally, a LabelEncoder is used to transform the text-based columns, Category, PdDistrict, and DayOfWeek, into numerical values. This is necessary for many machine learning algorithms which cannot work directly with text data.

The features and target variables are then separated from the dataset. The features are a subset of the columns from the original dataset that are deemed to be relevant for training the model. In this case, the relevant features are 'Year', 'Month', 'Day', 'Hour', 'Minute', 'Block', 'PdDistrict',



'DayOfWeek', 'X', and 'Y'. The target variable is the 'Category' column, which represents the type of crime that was committed.

The data is then split into a training set and a testing set using the train\_test\_split method from scikit-learn. The testing set will be used to evaluate the performance of the trained model. The random forest classifier is then trained on the training set using the RandomForestClassifier function from scikit-learn. The function accepts hyperparameters such as the number of decision trees to be used, the maximum depth of each decision tree, and a random seed for reproducibility.

Finally, the trained model is used to predict the target variable for the testing set, and the performance of the model is evaluated using various evaluation metrics. The accuracy\_score, f1\_score, precision\_score, and recall\_score functions from scikit-learn are used to calculate the accuracy, F1 score, precision, and recall of the predictions, respectively. The results are printed to the console. In this example, the accuracy score is **0.3333**, which indicates that the model correctly predicted the category of the crime in approximately one-third of the cases.

**Neural network 1st Attempt**:

The code is implementing a neural network model using Keras to predict crime categories based on various features such as location and time of occurrence. The model is trained on a dataset loaded from a CSV file using pandas.

After loading the dataset, the code performs some feature engineering by extracting the hour, month, and year from the 'Dates' column in the dataset. Then, the categorical variables 'PdDistrict', 'DayOfWeek', and 'Category' are encoded using LabelEncoder.

The code then splits the dataset into training and testing sets using the train\_test\_split function from scikit-learn. The training set is used to fit the neural network model, while the testing set is used to evaluate the model's performance.

The neural network model is defined using the Sequential API from Keras. The model consists of three dense layers with 128, 64, and 39 neurons respectively, and the ReLU activation function is used for the first two layers, and softmax activation function for the last layer. Dropout layers are added after each dense layer to prevent overfitting.

The model is compiled with the categorical\_crossentropy loss function, the adam optimizer, and the accuracy metric.

Early stopping is defined using the EarlyStopping callback from Keras. The training process is started using the fit method, with the training data, the number of epochs, and the batch size.

After training, the model is evaluated on the testing set using several metrics, including accuracy, precision, recall, and F1-score, which are calculated using the scikit-learn metrics library. The results show that the model's performance is not very good, with an accuracy score of **0.2208**, a precision score of 0.0487, a recall score of 0.2208, and an F1-score of 0.0798. This means that the model is not able to predict crime categories accurately, and it is likely that further improvements need to be made to the model architecture or the training process.



**Neural Network 2nd Attempt:**

The code starts by importing necessary libraries such as Pandas, NumPy, and various modules from scikit-learn and Keras.

Then, it loads the training data from a CSV file using the Pandas library. Next, it performs feature engineering by extracting the hour, month, and year from the "Dates" column of the dataset using the Pandas' to\_datetime function.

It then encodes the categorical variables using scikit-learn's LabelEncoder function. The data is then split into features and target, where the features are the X and Y coordinates, the PdDistrict, DayOfWeek, Hour, Month, and Year, and the target variable is the crime category.

Next, the input features are normalized using scikit-learn's StandardScaler. The data is then split into training and testing sets using scikit-learn's train\_test\_split function.

The neural network architecture is defined using Keras' Sequential model. It consists of four dense layers with ReLU activation functions, and dropout regularization is applied to reduce overfitting. The output layer uses a softmax activation function since this is a multi-class classification problem with 39 possible crime categories.

The model is then compiled using the Adam optimizer and categorical cross-entropy loss function. Early stopping is defined to prevent overfitting and ensure that the model doesn't train for too many epochs.

The model is then trained on the training data using Keras' fit function, with early stopping as a callback.

Finally, the trained model is evaluated on the test data, and the accuracy, precision, recall, and F1-score are printed using scikit-learn's accuracy\_score, precision\_score, recall\_score, and f1\_score functions. The accuracy score obtained is **0.2557**, precision is 0.1545, recall is 0.2557, and F1-score is 0.1629.

**Neural Network 3rd Attempt:**

This model is a neural network designed for multi-class classification. It uses the Python libraries pandas, numpy, scikit-learn, and Keras to preprocess the data, train the model, and evaluate its performance.

The first step in the model is to load the dataset using the pandas library. Then, the categorical variables are encoded using the LabelEncoder function from scikit-learn. The target variable is one-hot encoded using the to\_categorical function from Keras.

The input features are then normalized using the StandardScaler function from scikit-learn. The dataset is split into training and testing sets using the train\_test\_split function from scikit-learn.

The neural network architecture is defined using the Sequential class from Keras. The architecture consists of three hidden layers with 256, 128, and 64 nodes respectively, with the ReLU activation function, and a dropout layer with a dropout rate of 0.2 after each hidden layer. The output layer has as many nodes as there are classes in the target variable, with the softmax activation function.



The model is compiled using the categorical\_crossentropy loss function, the Adam optimizer, and the accuracy metric. Early stopping is also defined to monitor the validation loss and stop the training process if the validation loss does not improve for 5 epochs.

The model is trained using the fit method of the Sequential class from Keras, with a batch size of 512 and 50 epochs. The training process is monitored using the validation split and the early stopping callback.

The code evaluates the performance of the model on the testing data using accuracy, precision, recall, and F1-score. The model performance seems to be low with an accuracy of **0.2669**, precision of 0.1796, recall of 0.2669, and F1-score of 0.1810. Overall, the code performs data preprocessing, creates a neural network model, trains the model, and evaluates its performance on the testing data. However, the low accuracy, precision, recall, and F1-score indicate that the model may not be well-suited for the given task and may require further improvements or modifications.

**Neural network 4th Attempt**:

The code starts by loading data from a CSV file and performing feature engineering by extracting hour, month, and year from the 'Dates' column. The categorical variables such as 'PdDistrict', 'DayOfWeek', and 'Category' are encoded using the LabelEncoder class from Scikit-Learn. Then, the input features are normalized using StandardScaler.

The data is then split into training and testing sets using train\_test\_split() method from Scikit-Learn. Afterward, a neural network model is defined using the Sequential model from Keras. The model has four dense layers with 512, 256, 128, and output shape neurons respectively. The activation function used in these layers is ReLU.

The model is then compiled using the Adam optimizer and 'categorical\_crossentropy' loss function. The training process is defined using the fit() method of the model. Early stopping is applied to the training process using the EarlyStopping callback to prevent overfitting.

Finally, the model is evaluated on the test set, and the accuracy, precision, recall, and F1-score are calculated using the appropriate methods from Scikit-Learn. The accuracy of the model is reported as **0.2780**, which is relatively low.

It is worth noting that there are many factors that can affect the performance of a machine learning model, such as the quality and quantity of data, model architecture, hyperparameters, etc. Therefore, further analysis and experimentation might be necessary to improve the performance of this model.

# **Comparison of the various models:**

The accuracy scores of different machine learning models that were trained using the Counter Vector method. The accuracy scores are represented in a table format using the "tabulate" library. The table shows the model names and their corresponding accuracy scores.

The models used in the table are Random Forest, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Neural Network, and Naive Bayes. The accuracy scores range from 0.15 to



0.33, with Random Forest having the highest accuracy score of 0.33, and KNN having the lowest accuracy score of 0.15.

---+------------------------------------------+------------------+

| | Model\_Name using Counter\_vector Method | Accuracy Score |

+====+==========================================+==================+

| 0 | Random Forest Model | 0.33 |

+----+------------------------------------------+------------------+

| 1 | Support machine vector SVM | 0.26 |

+----+------------------------------------------+------------------+

| 2 | KNN | 0.15 |

+----+------------------------------------------+------------------+

| 3 | Neural Network | 0.27 |

+----+------------------------------------------+------------------+

| 4 | Naive Bayes | 0.28 |

+----+------------------------------------------+------------------+

**Conclusion:**

From the above table, it appears that the models were trained using the Counter Vector method and their accuracy scores were evaluated for crime prediction in San Francisco.

According to the accuracy scores, the Random Forest model achieved the highest accuracy score of 0.33, followed by Naive Bayes with an accuracy score of 0.28, SVM with an accuracy score of 0.26, Neural Network with an accuracy score of 0.27 and KNN with an accuracy score of 0.15.

Therefore, to further improve the accuracy of the crime prediction models in San Francisco, the following steps can be taken:

Use other feature extraction methods such as TF-IDF or word embeddings to see if they can improve the performance of the models.

Experiment with different hyperparameters of the models to find the optimal configuration that maximizes accuracy.

Consider using ensemble methods that combine multiple models to improve performance.

Use cross-validation to ensure that the accuracy scores are not biased by the specific training-test split used.

Gather more data to increase the size of the training dataset, which could lead to improved model performance.

To further improve the accuracy, other machine learning models can be tried, such as Gradient Boosting, XGBoost, and Decision Tree. Hyperparameter tuning can also be performed to optimize the performance of the models. Additionally, feature engineering techniques can be applied to improve the quality of the data being fed into the models



**References:**

To guarantee that the initiative is based on best practises and recommendations, research papers and publications about crime prevention and law enforcement will be consulted.

1. Dataset: [Crime Dashboard | San Francisco Police Department](https://www.sanfranciscopolice.org/stay-safe/crime-data/crime-dashboard)
2. Pratibha, A. Gahalot, Uprant, S. Dhiman and L. Chouhan, "Crime Prediction and Analysis," 2nd International Conference on Data, Engineering and Applications (IDEA), Bhopal, India, 2020, pp. 1-6, doi: 10.1109/IDEA49133.2020.9170731.
3. Shah et al. Visual Computing for Industry, Biomedicine, and Art (2021) 4:9 <https://doi.org/10.1186/s42492-021-00075-z>
4. [Machine learning in crime prediction | SpringerLink](https://link.springer.com/article/10.1007/s12652-023-04530-y#:~:text=Artificial%20Neural%20Networks%20came%20up,models%20for%20real%2Dtime%20predictions.)
5. [AI for Crime Prevention and Detection – 5 Current Applications | Emerj Artificial Intelligence Research](https://emerj.com/ai-sector-overviews/ai-crime-prevention-5-current-applications/)
6. [AI fails to predict crime. But researchers won't stop trying to. - The Washington Post](https://www.washingtonpost.com/technology/2022/07/15/predictive-policing-algorithms-fail/)
7. [SF-Crime Analysis & Prediction (Base Model) | Kaggle](https://www.kaggle.com/code/yannisp/sf-crime-analysis-prediction-base-model/notebook?scriptVersionId=9334889)
8. Perry, W. L. *Predictive Policing: The Role of Crime Forecasting in Law Enforcement Operations* (Rand, 2013).
9. [(PDF) Crime Prediction and Analysis (researchgate.net)](https://www.researchgate.net/publication/343750790_Crime_Prediction_and_Analysis)