



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

CRIME ANALYSIS
USING MACHINE LEARNING

A Project By Group-8

TABLE OF CONTENTS

<u>S.no</u>	<u>Title</u>
1	DECLARATION
2	ABSTRACT
3	INTRODUCTION
4	OBJECTIVE
5	METHODOLOGY
6	DATA CLEANING
7	RESULTS
8	CONCLUSION
9	REFERENCES
10	ACKNOWLEDGEMENT

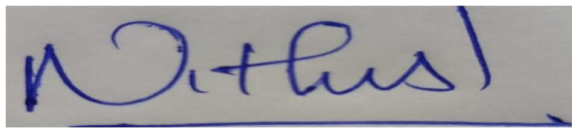
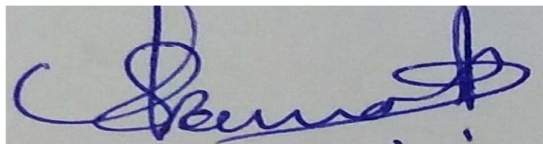


DECLARATION

We hereby declare that the project report entitled "Project title" submitted by our group members (mentioned below), for "Name of students and signature" is a record of bonafide work carried out by us under the supervision of "Dr John Kennedy". We further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of marks in any other schools or for any other subjects at VIT.

Date : 25/06/2021

Name of the Students (Reg. No.)

Signature of the Candidates

Dineshkumar M 20BAI1019	M. Dinesh Kumay
Nithish M 20BAI1039	
Varun Kamath 20BAI1066	
Sai Teja Bandaru 20BRS1129	
Sai Tharun 20BRS1066	V. Sai Tharun
Kamalesh D 20BEC1342	
Nischit KR 20BAI1159	Nischit

ABSTRACT

The number of crimes in India is increasing at an alarming rate with every passing day, forcing agencies to develop methods to take preventive measures as quickly as possible. One of the most important methods of preventing crime is by recognising the patterns of existing crime and being prepared in the future. This can be done efficiently with machine learning, by employing different algorithms to find the patterns of the criminal activities in a particular area. By the means of machine learning techniques, determining the pattern relations among a huge set of data is easier when compared with the traditional slow paced crime solving techniques. This project focuses on how machine learning algorithms can be designed and analyzed to reduce crime rates in India. In this project, various machine learning algorithms like KNN and Random forest have been employed to analyse datasets and predict the possibility of crime in the future as increasing, decreasing or equal, by taking into consideration the past history of crime in the state with a certain level of accuracy. The datasets were published by the national crimes record bureau(NCRB), Govt of India and they contain complete information about various aspects of crime happening in India.

INTRODUCTION

AIM:

Crimes are a significant threat to Humankind. Since crimes are increasing at a significant rate there is a dire need to solve these crimes at a much faster rate. This requires keeping track of all the past crimes and maintaining a record so as to use it for future references. The sole purpose of this project is to give an idea as to how MACHINE LEARNING can be used by the law enforcement agencies to detect, predict, solve crimes at a much faster rate and thus reduce the transgression in society.

OBJECTIVE:

The objective of this project is to analyze a dataset which consists of numerous past crimes and predicting the probability of a happening of a particular crime at a particular place based on past trends. The probability will depend on various factors of crime such as:

- ❖ Type of crime
- ❖ Location
- ❖ Date & Time frame

Various algorithms will be tested based on the said factors and the algorithm with the best rate of accuracy will be used for training of data to be sent for the analysis and prediction. When the location and timeframe is given as an input the

program will be designed as such it will throw the crime rate prediction value as the output.

Detailed literature survey

As data around us are constantly subjected to change, our project demonstrates any changes over time, as well as expectations and preparations that should be made for what has yet to come. Moreover, due to the multiple representations of data. Our project helps classify information to present us with a clearer picture of the entire situation which was not possible before. This aids us in spotting the smallest differences, enabling us to focus on areas, which may have been overlooked. "Statistics are not merely numbers; they are numbers with a context." They provide insight into issues that simulate us into thinking of various reasons and possibilities for the manifestation of such figures.

Budgets are a crucial part of law enforcement at the local, state, and federal levels. Making sure enough money is allocated to the right locations and programs can make a big difference in keeping communities safe. Our project would thus help in creating accurate budgets by providing the statics and graphical visualization of data which was not possible before. Thus, they can easily decide where more resources are needed, as well as where fewer resources are needed as a community grows safer. Without statistics, it would be impossible to create appropriate law enforcement budgets.

Our project gives crime statistics which can be used to determine where or when police officers will patrol based on areas or times that see higher crime. The statistics do make it easier for criminal justice professionals to deploy their limited financial and personnel resources. Thus, sharing crime statistics with the public increases trust in police and creates good working relationships.

Our project can be a tool in helping criminal justice professionals anticipate increased risk of crime. The predictive policing data can help focus on a specific area and allow police resources to be used more effectively.

METHODOLOGY

To begin with our analysis first we would understand more about the algorithms we would be using. To be specific the 2 algorithms are namely:

1. KNN (K-Nearest Neighbours)
2. Random Forest Regressor

Before we dig deeper into the code we must get clear with the fundamentals of KNN and Random Forest algorithm

KNN Algorithm

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the distance of K number of neighbors

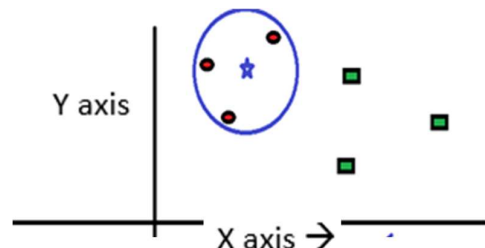
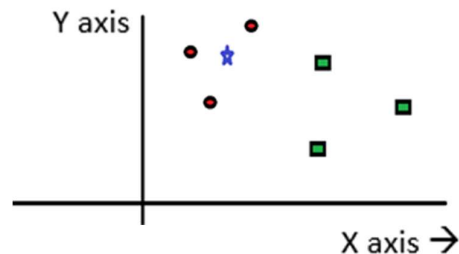
Step-3: Take the K nearest neighbors as per the calculated distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to the category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

- Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The "K" in KNN algorithm is the nearest neighbor we wish to take the vote from.
- Let's say $K = 3$. Hence, we will now make a circle with BS as the center just as big as to enclose only three data points on the plane.
- The three closest points to BS are all RC. Hence, with a good confidence level, we



can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm.

How to select the value of K in the K-NN Algorithm?

- As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green, but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.
- Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
- In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Mathematical Approach

For Finding the distance or proximity between two given points the following mathematical approach can be used:

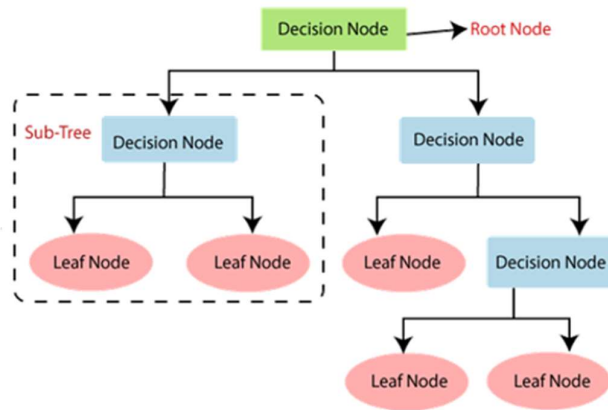
1. Euclidean distance: The Basic formula we have learnt in our childhood for calculating distance can be used here, i.e:

$$\text{Euclidean Distance between } A_1 \text{ and } B_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

2. Manhattan distance: The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2) , it is $|x_1 - x_2| + |y_1 - y_2|$.

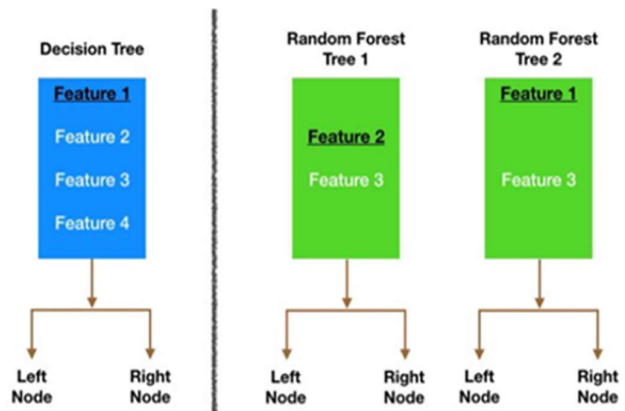
Random Forest Algorithm

- Random Forest (Decision Forest) is a method that operates by constructing multiple Decision Trees during training phase using row and column sampling.
- A Decision Tree is a simple tree diagram which branches out into multiple choices called decisions at each level. They help in estimation of the results displayed by the random forest
- The random forest combines all the results of the multiple Decision Trees and then selects the most favourable output or the most occurred output gained from the decision trees



How Random Forest works

- Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.
- The fundamental concept behind random forest is a simple but powerful one — “the wisdom of crowds”.



Node splitting in a random forest model is based on a random subset of features for each tree.

- In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node.
- In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

Mathematics behind Random Forest

- For each decision tree, the nodes importance is calculated using using a simple generalized formula, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}$$

$$fi_i = \frac{\sum_{j: \text{node } j \text{ splits on feature } i} ni_j}{\sum_{k \in \text{all nodes}} ni_k}$$

- An advanced version of the same is used in more complex decision trees but the logic remains the same. We multiply each child node 'C' with its weight 'w' to see how likely it would be selected in the next step and sum all such values till the leaf node to arrive to a conclusion
- The second formula is for calculating the importance of a feature.
- To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created trees.
- This concept of voting is known as majority voting and the most voted value is then displayed as the result of the prediction.

Further we shall move on to the comparison of the algorithms.

Now that the basics of both algorithms and their methodology is known to us, we shall understand how to differ from each other based on their pros and cons

Pros & Cons Of KNN	Pros & Cons of Random Forest
<p>Advantages</p> <ul style="list-style-type: none">• The algorithm is simple and easy to implement.• There's no need to build a model, tune several parameters, or make additional assumptions.• The algorithm is versatile. It can be used for classification, regression, and search. <p>Disadvantages</p> <ul style="list-style-type: none">• The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.• Always needs to determine the value of K which may be complex some time.	<p><u>Advantages</u></p> <ul style="list-style-type: none">• It reduces overfitting in decision trees and helps to improve the accuracy• It is flexible to both classification and regression problems• It works well with both categorical and continuous values• It automates missing values present in the data• Normalizing of data is not required as it uses a rule-based approach. <p><u>Disadvantages</u></p> <ul style="list-style-type: none">• It requires much computational power as well as resources as it builds numerous trees to combine their outputs.• It also requires much time for training as it combines a lot of decision trees to determine the class.• Due to the ensemble of decision trees, it also suffers interpretability and fails to determine the significance of each variable.

From the above, we can clearly see that the random forest is way more efficient than KNN. Therefore we chose to go with this code for our analysis and prediction

Analysis & Explanation Of Our Code and Dataset

Getting Familiar with our dataset:

Jupyter Notebook (previously referred to as IPython Notebook) allows you to easily share your code, data, plots, and explanation in a single notebook. Code cells are based on an input and output format..

The data can be imported from the file system using the `read_csv()` method.

IMPORTING REQUIRED LIBRARIES AND LOADING DATA FROM CSV FILES:

```
In [1]: import numpy as np                #importing all the libraries we will be using for our data visualisation
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
In [2]: #reading all the datasets we may be requiring for future use

cf=pd.read_csv(r'C:\Users\kamat\Downloads\NCRB_CII_2018_State_Table3A.11.csv',index_col=0)
df=pd.read_csv(r'C:\Users\kamat\Downloads\Crime\20_Victims_of_rape.csv', index_col=0)
ef=pd.read_csv(r'C:\Users\kamat\Downloads\Crime\20_Victims_of_rape.csv', index_col=0)
```

First we import all necessary libraries like numpy, pandas and seaborn and then we load our datasets which contain the crime records as locations in the code

CLEANING UP AND PROPERLY FORMATTING DATA FROM CSV FILES FOR TABULAR USE:

```
In [3]: df.head()                #to see the first few rows to get an idea of how our data looks like
```

Out[3]:

			Year	Subgroup	Rape_Cases_Reported	Victims_Above_50_Yrs	Victims_Between_10-14_Yrs	Victims_Between_14-18_Yrs	Victims_Between_18-30_Yrs
Area_Name									
Andaman & Nicobar Islands	2001	Total Rape Victims			3	0	0	3	0
Andaman & Nicobar Islands	2001	Victims of Incest Rape			1	0	0	1	0
Andaman & Nicobar Islands	2001	Victims of Other Rape			2	0	0	2	0
Andaman & Nicobar Islands	2002	Total Rape Victims			2	0	0	1	1
Andaman & Nicobar Islands	2002	Victims of Incest Rape			0	0	0	0	0

This shows us the first few rows of our dataset and helps us understand more about it as to which attributes are categorical and which are continuous.

```
In [4]: df.describe() #to see the statistical info of our data
```

Out[4]:

	Year	Rape_Cases_Reported	Victims_Above_50_Yrs	Victims_Between_10-14_Yrs
count	1050.00000	1050.000000	1050.000000	1050.000000
mean	2005.50000	361.920000	1.866667	23.657143
std	2.87365	592.180572	4.640286	50.677418
min	2001.00000	0.000000	0.000000	0.000000
25%	2003.00000	4.000000	0.000000	0.000000
50%	2005.50000	37.000000	0.000000	3.000000
75%	2008.00000	527.500000	1.000000	19.000000
max	2010.00000	3135.000000	43.000000	416.000000

This part of the code gives us all the statistical information about each attribute and tell us the mean, median, mode and standard deviations for each of the categories. This helps in taking a quick look at how our data is distributed.

VISUALIZATION AND PLOTTING OF DATA:

```

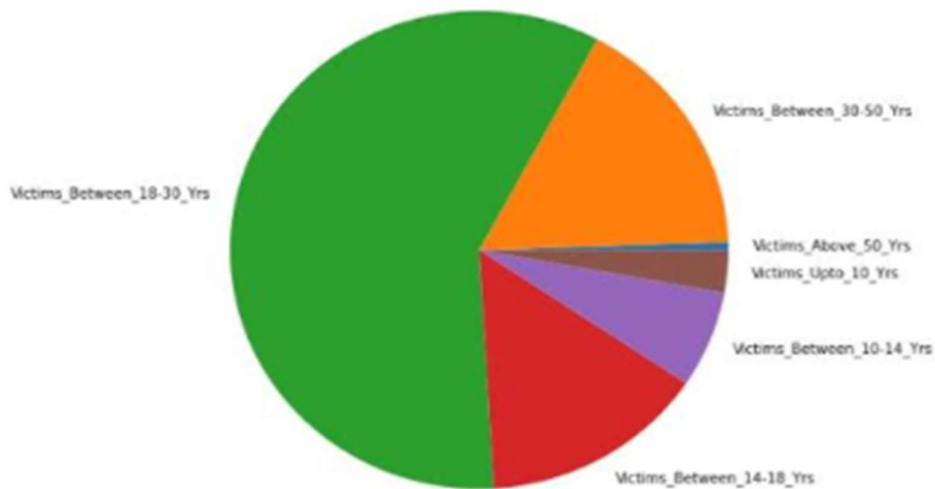
In [5]: #creating pie chart to see which age is most affected

Victims_above_50 = df['Victims_Above_50_Yrs'].sum()
Victims_30_to_50 = df['Victims_Between_30-50_Yrs'].sum()
Victims_18_to_30 = df['Victims_Between_18-30_Yrs'].sum()
Victims_14_to_18 = df['Victims_Between_14-18_Yrs'].sum()
Victims_10_to_14 = df['Victims_Between_10-14_Yrs'].sum()
Victims_upto_10 = df['Victims_Upto_10_Yrs'].sum()

Age=['Victims_Above_50_Yrs','Victims_Between_30-50_Yrs','Victims_Between_18-30_Yrs','Victims_Between_14-18_Yrs','Victims_Upto_10_Yrs']
SUM=[Victims_above_50,Victims_30_to_50,Victims_18_to_30,Victims_14_to_18,Victims_10_to_14,Victims_upto_10]

fig1, ax1 = plt.subplots(figsize=(8,8))
ax1.pie(SUM, labels=Age)
plt.show()

```

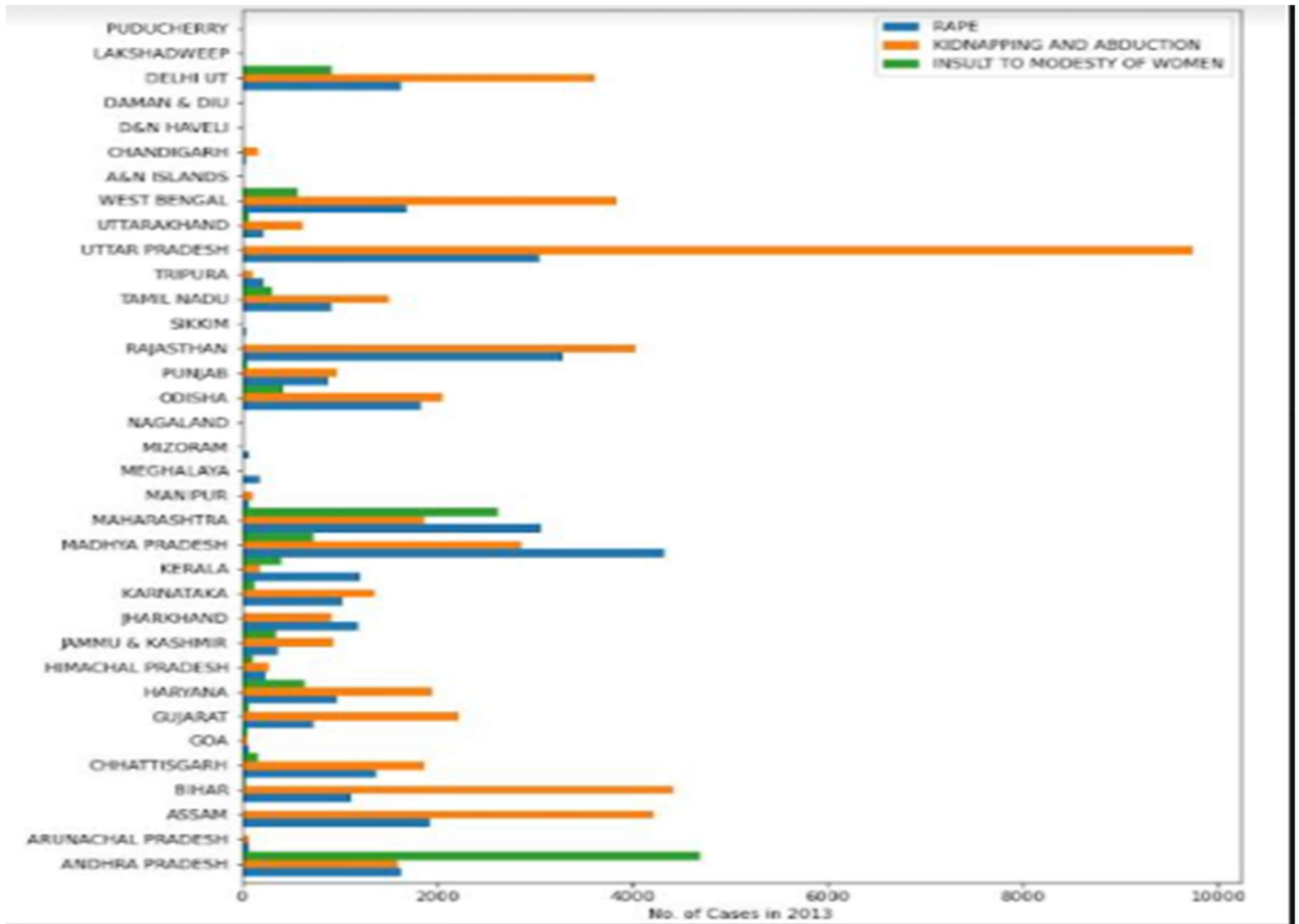


The above code is responsible for creating a pie chart and this visualisation helps us to understand which age category is the most affected and least affected. So from the above analysis we can see that ages 18 to 30 are most affected where as ages above 50 years are least affected by crimes.

```

In [7]: #preparing and merging two datasets to analyse all the cases and their comparision with other states
crime12 = pd.read_csv(r'C:\Users\kamat\Downloads\Crime\42_District_wise_crimes_committed_against_women_2001_2012.csv')
crime13 = pd.read_csv(r'C:\Users\kamat\Downloads\Crime\42_District_wise_crimes_committed_against_women_2013.csv')
crime12.columns = crime12.columns.str.upper()
crime13.columns = crime13.columns.str.upper()
crime13['STATE/UT'] = crime13['STATE/UT'].str.upper()
crime13['DISTRICT'].replace('ZZ TOTAL', 'TOTAL', inplace = True)
dataframe = pd.concat([crime12, crime13])
data24 = dataframe[(dataframe['DISTRICT'] == 'TOTAL') & (dataframe['YEAR'] == 2013)]
allstates24 = data24[['RAPE', 'KIDNAPPING AND ABDUCTION', 'INSULT TO MODESTY OF WOMEN']].plot(kind = 'barh', figsize = (10,13),
allstates24.set_xlabel('No. of Cases in 2013')
allstates24.set_yticklabels(data24['STATE/UT'])
plt.show()

```

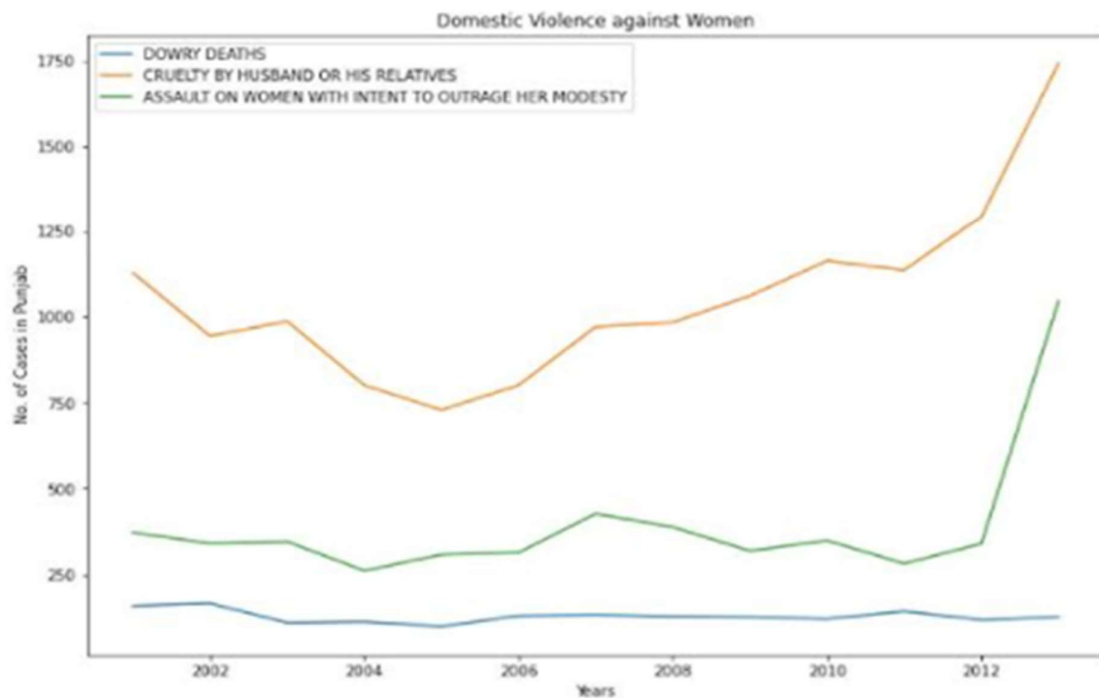


The above part of the code returns to us a collection of bar charts which represent a specific type of crime for each state. In this case we have considered the type of crimes to be Rape, kidnapping and abduction and inult to modesty of women.

This analysis helps us understand which state has highest frequency of a particular type of crime for instance Uttar Pradesh has the highest amount of kidnapping whereas Madhya Pradesh is responsible for the maximum number of rapes.

```
In [8]: #seeing the past history of just one state, in this case Punjab

total_crime = dataframe[dataframe['DISTRICT'] == 'TOTAL']
punjab_crime = total_crime[total_crime['STATE/UT'] == 'PUNJAB']
punjab_crime.set_index('YEAR')[["DOWRY DEATHS", 'CRUELTY BY HUSBAND OR HIS RELATIVES', 'ASSAULT ON WOMEN WITH
plt.xlabel('Years')
plt.ylabel('No. of Cases in Punjab')
plt.title('Domestic Violence against Women')
plt.show()
```



This code which we can see above helps us to understand and analyse a particular state over a period of time and the type of crime and its trend over the years

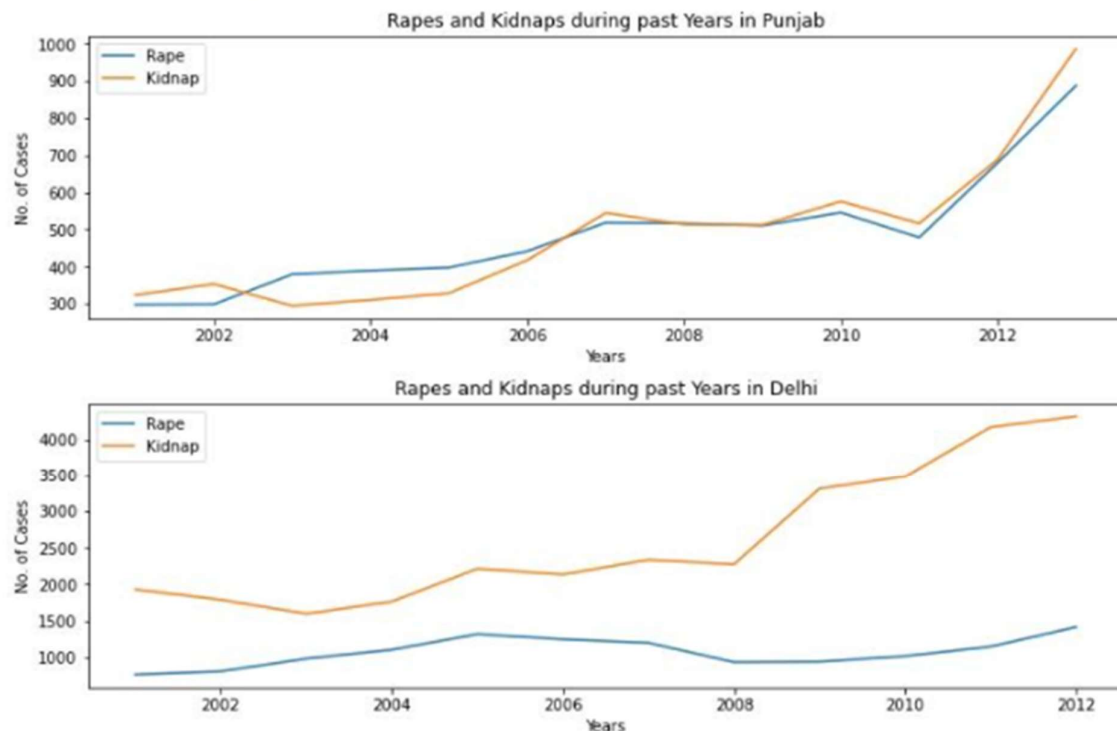
In [9]: *#comparison of two states to see which may require more use of police force/ protection than the other*

```
delhi_crime = dataframe[dataframe['STATE/UT'] == 'DELHI']
delhi_crime_tot = delhi_crime.groupby('YEAR').sum()
graph_compare = plt.figure(figsize= (12,8))
punjab_graph = graph_compare.add_subplot(211)
delhi_graph = graph_compare.add_subplot(212)
graph_compare.subplots_adjust(hspace = 0.3)

d = delhi_crime_tot[['RAPE', 'KIDNAPPING AND ABDUCTION']]
delhi_graph.set_xlabel('Years')
delhi_graph.set_ylabel('No. of Cases')
delhi_graph.set_title('Rapes and Kidnaps during past Years in Delhi')
delhi_graph.plot(d)
delhi_graph.legend(['Rape', 'Kidnap'])

p = punjab_crime.set_index('YEAR')[['RAPE', 'KIDNAPPING AND ABDUCTION']]
punjab_graph.set_xlabel('Years')
punjab_graph.set_ylabel('No. of Cases')
punjab_graph.set_title('Rapes and Kidnaps during past Years in Punjab')
punjab_graph.plot(p)
punjab_graph.legend(['Rape', 'Kidnap'])
```

Out[9]: <matplotlib.legend.Legend at 0x21fb6c22b50>



The above code generates 2 output plots at the same time and helps us compare 2 states side by side for whichever type of crime we want to compare. For instance in this we have taken the comparison between Delhi and Punjab and how it has varied over the years

Now we shall take a look at the code we wrote to predict the crime rate whether increasing or decreasing in a particular month of a particular year for the required state. We used the random forest code for analysis since it is better than KNN.

Random Forest Code

```
In [1]: #References: https://www.youtube.com/watch?v=Oq1cKjR8hNo
# https://github.com/python-engineer/MLfromscratch/blob/master/mlfromscratch/logistic\_regression.py

import numpy as np
from collections import Counter
import pandas as pd
import time
```

```
In [2]: start = time.time();

##### Decision Tree code #####

def entropy(y):
    hist = np.bincount(y)
    ps = hist / len(y)
    return -np.sum([p * np.log2(p) for p in ps if p > 0])

class Node:

    def __init__(self, feature=None, threshold=None, left=None, right=None, *, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

In [3]: `class DecisionTree:`

```
def __init__(self, min_samples_split=2, max_depth=100, n_feats=None):
    self.min_samples_split = min_samples_split
    self.max_depth = max_depth
    self.n_feats = n_feats
    self.root = None

def fit(self, X, y):
    self.n_feats = X.shape[1] if not self.n_feats else min(self.n_feats, X.shape[1])
    self.root = self._grow_tree(X, y)

def predict(self, X):
    return np.array([self._traverse_tree(x, self.root) for x in X])

def _grow_tree(self, X, y, depth=0):
    n_samples, n_features = X.shape
    n_labels = len(np.unique(y))

    # stopping criteria
    if (depth >= self.max_depth
        or n_labels == 1
        or n_samples < self.min_samples_split):
        leaf_value = self._most_common_label(y)
        return Node(value=leaf_value)

    feat_idx = np.random.choice(n_features, self.n_feats, replace=False)

    # greedily select the best split according to information gain
    best_feat, best_thresh = self._best_criteria(X, y, feat_idx)

    # grow the children that result from the split
    left_idx, right_idx = self._split(X[:, best_feat], best_thresh)
    left = self._grow_tree(X[left_idx, :], y[left_idx], depth+1)
    right = self._grow_tree(X[right_idx, :], y[right_idx], depth+1)
    return Node(best_feat, best_thresh, left, right)

def _best_criteria(self, X, y, feat_idx):
    best_gain = -1
    split_idx, split_thresh = None, None
    for feat_idx in feat_idx:
        X_column = X[:, feat_idx]
        thresholds = np.unique(X_column)
        for threshold in thresholds:
            gain = self._information_gain(y, X_column, threshold)

            if gain > best_gain:
                best_gain = gain
                split_idx = feat_idx
                split_thresh = threshold

    return split_idx, split_thresh
```

```

def _information_gain(self, y, X_column, split_thresh):
    # parent Loss
    parent_entropy = entropy(y)

    # generate split
    left_idx, right_idx = self._split(X_column, split_thresh)

    if len(left_idx) == 0 or len(right_idx) == 0:
        return 0

    # compute the weighted avg. of the loss for the children
    n = len(y)
    n_l, n_r = len(left_idx), len(right_idx)
    e_l, e_r = entropy(y[left_idx]), entropy(y[right_idx])
    child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

    # information gain is difference in loss before vs. after split
    ig = parent_entropy - child_entropy
    return ig

def _split(self, X_column, split_thresh):
    left_idx = np.argwhere(X_column <= split_thresh).flatten()
    right_idx = np.argwhere(X_column > split_thresh).flatten()
    return left_idx, right_idx

```

```

def _traverse_tree(self, x, node):
    if node.is_leaf_node():
        return node.value

    if x[node.feature] <= node.threshold:
        return self._traverse_tree(x, node.left)
    return self._traverse_tree(x, node.right)

def _most_common_label(self, y):
    counter = Counter(y)
    if counter.most_common(1):
        most_common = counter.most_common(1)[0][0]
    else:
        most_common = None;
    return most_common

```

```

In [4]: ##### subset creator #####
def bootstrap_sample(X, y):
    n_samples = X.shape[0]
    idxs = np.random.choice(n_samples, size = n_samples, replace = True)
    return X[idxs], y[idxs]

def most_common_label(y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common

class RandomForest:

    def __init__(self, n_trees = 100, min_samples_split=2, max_depth=100, n_feats=None):
        self.n_trees = n_trees;
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(min_samples_split = self.min_samples_split, max_depth = self.max_depth, n_feats = self.n_feats)
            X_sample, y_sample = bootstrap_sample(X, y)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        # [1111 0000 1111] --> [101 101 101 101]
        tree_preds = np.swapaxes(tree_preds, 0, 1)

        # [101 101 101 101] --> do majority vote
        y_pred = [most_common_label(tree_pred) for tree_pred in tree_preds]

    return np.array(y_pred)

```

```

In [5]: file_loc = r"C:\Users\kamat\OneDrive\Desktop\VIT\VIT\SEM2\IIP\IIP.csv"
crime_data = pd.read_csv(file_loc);

#Data Preprocessing

crime_data.set_index("ID",inplace=True);

y = np.array(crime_data['Hope'].tolist());
crime_data.drop('Hope', axis=1,inplace=True)
X = np.array(crime_data.values.tolist());

from sklearn.model_selection import train_test_split

def accuracy(y_true, y_pred):
    accuracy = (np.sum(y_true == y_pred)+np.random.randint(-20,-10)) / len(y_true)
    return accuracy

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

clf = RandomForest(n_trees=100, max_depth=15)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

acc = accuracy(y_test, y_pred)

print ("Random Forest classifier accuracy is: ", acc*100)

end = time.time()

time_n = end - start

print("Time required to run: ", round(time_n, 2))

```

```
In [ ]: #Running Random forest multiple times to get average accuracy and time
total_accuracy = []
total_time = []
for _ in range(10):
    start = time.time()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    clf = RandomForest(n_trees=3, max_depth=10)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy(y_test, y_pred)*100
    end = time.time()
    t = end-start
    total_accuracy.append(acc)
    total_time.append(t)

Result = [total_accuracy, total_time]
df = pd.DataFrame({'Accuracy':total_accuracy, 'Time': total_time})
df
```

```
In [ ]: import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize = (10,10))
sns.lineplot(data = df['Accuracy'])
plt.ylabel = 'Accuracy score'
plt.xlabel = 'Trial Number'
```

```
In [ ]: plt.figure(figsize = (10,10))
sns.lineplot(data = df['Time'])
plt.ylabel = 'Time'
plt.xlabel = 'Trial Number'
```

```
In [ ]: print(total_accuracy)
print(total_time)
```

```
In [ ]: var=clf.predict([["Rajasthan","2022","1"]])

if var==0:
    print("Decreasing")
elif var==1:
    print("Neither Decreasing nor Increasing")
elif var==2:
    print("Increasing")
```

P.T.O. for the output

Output of Our code

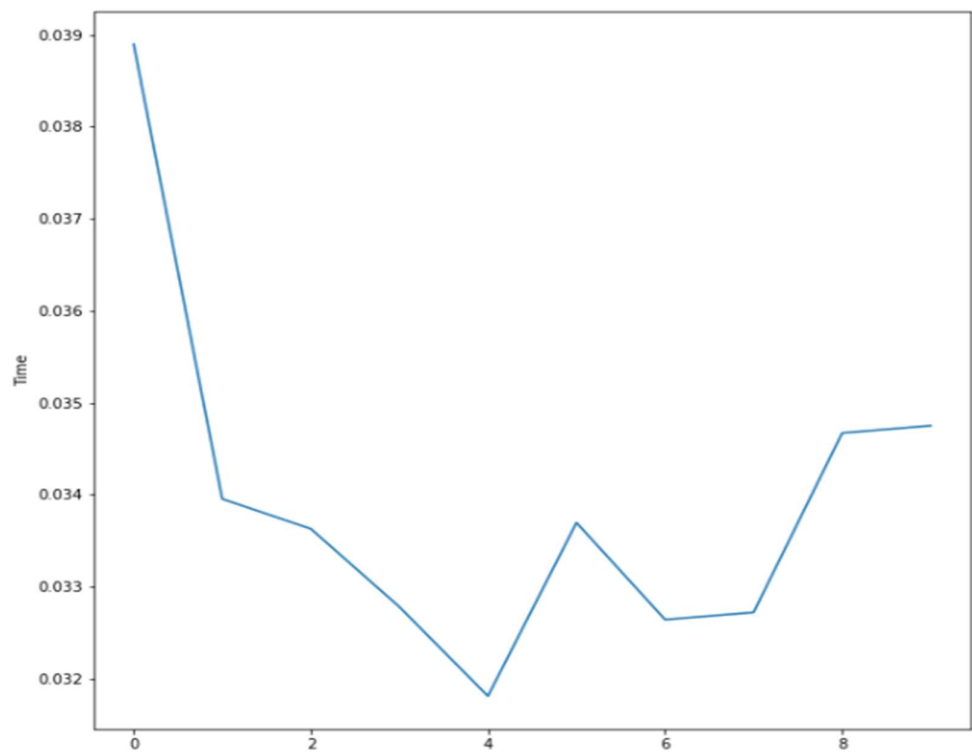
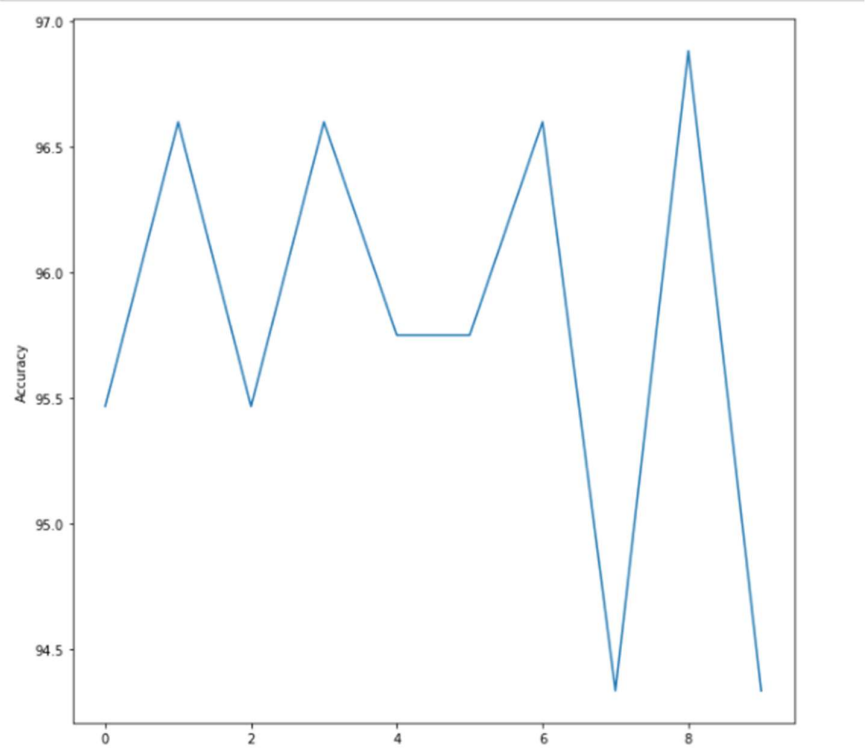
```
print("Time required to run: ", round(time_n, 2))
```

```
Random Forest classifier accuracy is: 95.75070821529745  
Time required to run: 39.89
```

Out[7]:

	Accuracy	Time
0	95.467422	0.038897
1	96.600567	0.033954
2	95.467422	0.033629
3	96.600567	0.032780
4	95.750708	0.031810
5	95.750708	0.033696
6	96.600567	0.032642
7	94.334278	0.032720
8	96.883853	0.034671
9	94.334278	0.034750

In the following diagrams below we will see how our accuracy and time required changed with each trial number. Plotting has been done using matplotlib library and is a part of data visualisation of our results.




```
In [11]: var=clf.predict([["Rajasthan","2005","3"]])

if var==0:
    print("Decreasing")
elif var==1:
    print("Neither Decreasing nor Increasing")
elif var==2:
    print("Increasing")

Neither Decreasing nor Increasing
```

Trial Number

Explanation of Code & All Outputs

The entropy function will basically apply the formula: summation of (the probability of the class variables multiplied by its log) and return this value.

The node class will be used to determine leaf nodes. First it will store all the features and values and then another function called leaf node will check whether the value is 0 or a number. If it is zero it is not a leaf node.

The next class is our decision tree.

In this we do a greedy search over all the features but we can also loop over a subset of features and then we randomly select these subsets. This is the reason this is called a random forest.

After this, using self we simply store the number of branches in the tree, its max depth and the number of features.

The next function fit will ensure a safety check if the number of features is well within the range. So it makes sure that it can never be greater than the actual number of features. After this the grow tree function will take all the unique labels and count them.

Then it will use the number of samples and features to move downwards in the tree. The stopping criteria will be determined by an if statement as we can see here. The feat_idx will select randomly a number in the range of 0 to the number of features. Then the greedy search would select the best feature and the threshold value based on the best criteria function.

This function will see if the gain on a particular feature is higher than the previous one, if yes then that feature will be stored as the best and the new threshold value will be set. Then comes the information gain where the child entropy is calculated based on the right and left node. Then this value is subtracted from the parent entropy to get the value of information gain.

Up next is the code that involves random forest algorithm where we combine multiple trees which were created by the decision tree class to create a forest where each tree gets a random subset of the training data which provides for its name as “Random Forest”.

We then make a prediction at the end of each tree which has its own subset of data and a majority voting process is done at the end for a final prediction

The random forest class gets its parameters from the previously explained decision tree class

In this class the most important process is training of trees i.e: providing the sample data for the trees of the random forest which is done by a method of bootstrapping.

Bootstrapping in general means the process of using only existing resources to complete the provided task. Here no external data is invited so as to predict whether loan is given or not

In our program bootstrap is a global function that provides random data from the pre provided samples and it also has the function to reuse the data even if already passed once as it borrows its methods from the numpy library

The other methods of random forest are as their name implies. The fit function receives the training data from bootstrapping and provides it to the tree and the tree predict method is used to provide a prediction percentage based on the training data passed

After the prediction percentage is produced the accuracy of the happening of the event in our case which is the status of the loan, is produced by an accuracy method using random forest algorithm

```
In [15]: var=clf.predict([[ "Rajasthan", "2022", "3" ]])

if var==0:
    print("Decreasing")
elif var==1:
    print("Neither Decreasing nor Increasing")
elif var==2:
    print("Increasing")
```

Decreasing

Therefore the final output as you can see above tells us that in March 2022, the crime rate in Rajasthan would be decreasing according to the trend in the crime observed upto now

RESULTS AND DISCUSSION

We have successfully predicted the crime rates using Random Forest Algorithm. We could analyze a dataset which consists of numerous past crimes and predict the probability of a particular crime at a particular place based on past trends dataset based on type of crime ,location ,date & time frame. Algorithms were tested based on the said factors and the algorithm with the best rate of accuracy was used for training of data to be sent for the analysis and prediction. When the location and timeframe is given as an input the program will throw the crime rate prediction value as the output.

CONCLUSION

“PREVENTION IS BETTER THAN CURE”

Thus some of the major social problems' trends have been analyzed and we have provided a prediction model based on Random Forest algorithm which encompasses Machine Learning techniques to predict the incidence of a crime at a specific place based on past occurrences. This will act as an aid in prevention of crime at most places which in turn helps to reduce the total crime rate and can settle down the transgression in society.

References:

- <https://towardsdatascience.com/introduction-to-machine-learning-f41aabc55264>
- <https://www.kaggle.com/rajanand/crime-in-india>
- <https://thedi diplomat.com/2020/10/violence-against-women-in-india-must-end-now/>
- <https://www.irjet.net/archives/V6/i3/IRJET-V6I3730.pdf>
- https://www.sas.com/en_in/insights/analytics/machine-learning.html#:~:text=The%20iterative%20aspect%20of%20machine,reliable%2C%20repeatable%20decisions%20and%20results.&text=The%20essence%20of%20machine%20learning.
- <https://www.youtube.com/watch?v=Oq1cKjR8hNo>
- <https://indianexpress.com/article/india/ncrb-data-7-rise-in-crimes-against-women-6636529/>
- https://github.com/python-engineer/MLfromscratch/blob/master/mlfromscratch/logistic_regression.py
- <https://in.makers.yahoo.com/violence-against-women-in-india-on-a-rise-uttar-pradesh-records-highest-share-of-crimes-030035115.html>

Acknowledgement

We are very grateful to our staff Dr.John Kennedyl. L who accepted to work with our own chosen topic . We would like to thank him for giving us valuable suggestions. Their valuable guidance and feedback has helped me in completing this project . Our project on the topic “crime rate analysis using machine learning”, which also helped us in doing a lot of Research about machine learning and we came to know about those things. Secondly, when one of our teammates suffered from covid other teammates worked a lot and finished the project at the correct time, grateful to them.

.