VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

SUBJECT: DATA MINING

PROJECT REPORT

TOPIC:

PREDICT THE TERRORIST LOCATIONS IN FUTURE

Lecturer: Mrs. Cao Thi Nhan
Mr. Vu Minh Sang

Class: IS252.O21.HTCL

Group: Team 13

Student performance:

Chau Thanh Binh	21521871
Nguyen Thi Thao Ngan	21521176
Ngo Anh Tuan	21521629
Phi Quang Thanh	21521449

Ho Chi Minh City, June 2024

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

———□🕮□———

SUBJECT: DATA MINING

FINAL PROJECT REPORT

TOPIC:

PREDICT THE SUCCESS OF TERRORISM IN FUTURE

Lecturer: Mrs. Cao Thi Nhan
Mr. Vu Minh Sang

Class: IS252.O21.HTCL

Group: Team 13

Student performance:

Chau Thanh Binh	21521871
Nguyen Thi Thao Ngan	21521176
Ngo Anh Tuan	21521629
Phi Quang Thanh	21521449

Ho Chi Minh City, June 2024

TEACHER'S COMMENTS

ACKNOWLEDGEMENT

In fact, there is no success that is not tied to the support and help, whether more or less, directly or indirectly from others. With the deepest gratitude, first of all, our group would like to express our sincere thanks to the teachers of the University of Information Technology - Vietnam National University, Ho Chi Minh City and the teachers of the Faculty of Information Systems helped the group to have the basic knowledge as a basis to carry out this topic.

In particular, our team would like to express our sincere thanks to Mrs. Cao Thi Nhan - theoretical lecturer and Mr. Vu Minh Sang - practical lecturer of Data Mining who wholeheartedly helped, directly instructed and guided the group throughout the process of the project.

Thanks to that, we have gained a lot of useful knowledge in applying as well as project-making skills. Without the guidance and teachings of the teacher, our group thinks this project of the group would be very difficult to complete. Once again, I sincerely thank the teacher. In addition, for the project to be completed, it is impossible to thank the people who did it, thank you to the team members who worked hard and completed the task on schedule.

Finally, thank you to all the team members who worked at their best to complete their thesis well. Sincerely, thank!

TABLE OF CONTENTS

TEACHER'S COMMENTS 3
ACKNOWLEDGEMENT 4
TABLE OF CONTENTS 5
CHAPTER I: INTRODUCTION 7
1.1 Problems 7
1.2 Project goal 7
1.3 Developer Tools & Technology 7
CHAPTER II: DATA PREPROCESSING 8
2.1 Description of original data 8
2.1.1 Data Sources 8
2.1.2 Data file 8
2.1.4 Statistics of Attribute Values 9
2.2 Data preprocessing 10
2.2.1 Import Library 10
2.2.2 Import Dataset 11
2.2.3 Check data type, information 11
2.2.4 Overview of the data 12
2.2.5 Description of dataset information 13
2.2.6 Handle the null data 14
2.2.7 Check the balance data of the predictor attribute 15
CHAPTER III: ALGORITHMS AND EXPERIMENTS 16
3.1 Algorithm used 16
3.1.1 Decision Tree 16
3.1.2 Random Forest 18
3.1.2.1 Random Forest explaination 18
3.1.2.2 Random Forests Algorithm 19
3.1.3 Naïve Bayes 22
3.1.3.1 Bayes Theorem 22
3.1.3.2 Naïve Bayes classification algorithm 24
3.1.4 K-Nearest Neighbors 26
3.1.4.1 K-Nearest Neighbors explain 26
3.1.4.2 K-nearest neighbors Algorithm 26
3.2 Experiments on Jupyter Notebook 28
3.2.1 Separating train and test data 28

3.2.2 Decision Trees Algorithm 28
3.2.3 Random Forest algorithm 30
3.2.4 Naïve Bayes Algorithm 32
3.2.5 K-Nearest Neighbors Algorithm 35
3.2.6 Comparison, Evaluation 36
CHAPTER IV: CONCLUSION 41
4.1 Advantages and limitations of each algorithm 41
4.1.1 Decision Tree 41
4.1.2 Random Forest 42
4.1.3 Naïve Bayes 44
4.1.4 K–Nearest Neighbors 45
REFERENCES 46

CHAPTER I: INTRODUCTION

1.1 Problems

Terrorism significantly impacts societies globally, with diverse forms and motivations across different regions. Understanding and countering terrorism requires comprehensive data analysis. The Global Terrorism Database (GTD) provides detailed information on over 180,000 terrorist attacks worldwide from 1970 to 2017. Leveraging advances in Data Science and predictive modeling, this project aims to automatically classify and analyze terrorist incidents using the GTD data to develop effective counter-terrorism strategies.

1.2 Project goal

- Develop a machine learning system to predict and analyze terrorist activities.
- Assist in identifying patterns and trends to enhance security and inform counter-terrorism efforts.

1.3 Developer Tools & Technology

In the process of implementation, the group used a number of software for researching and developing the topic:

- Information collection and analysis using the Python library and programming language.
- Data sources: Global Terrorism Database | Kaggle

CHAPTER II: DATA PREPROCESSING

2.1 Description of original data

2.1.1 Data Sources

Collaborators: START Consortium (Owner), Erin Miller (Admin)

Sources: [Global Terrorism Database | Kaggle]

2.1.2 Data file

Total data rows: Over 180.000 rows

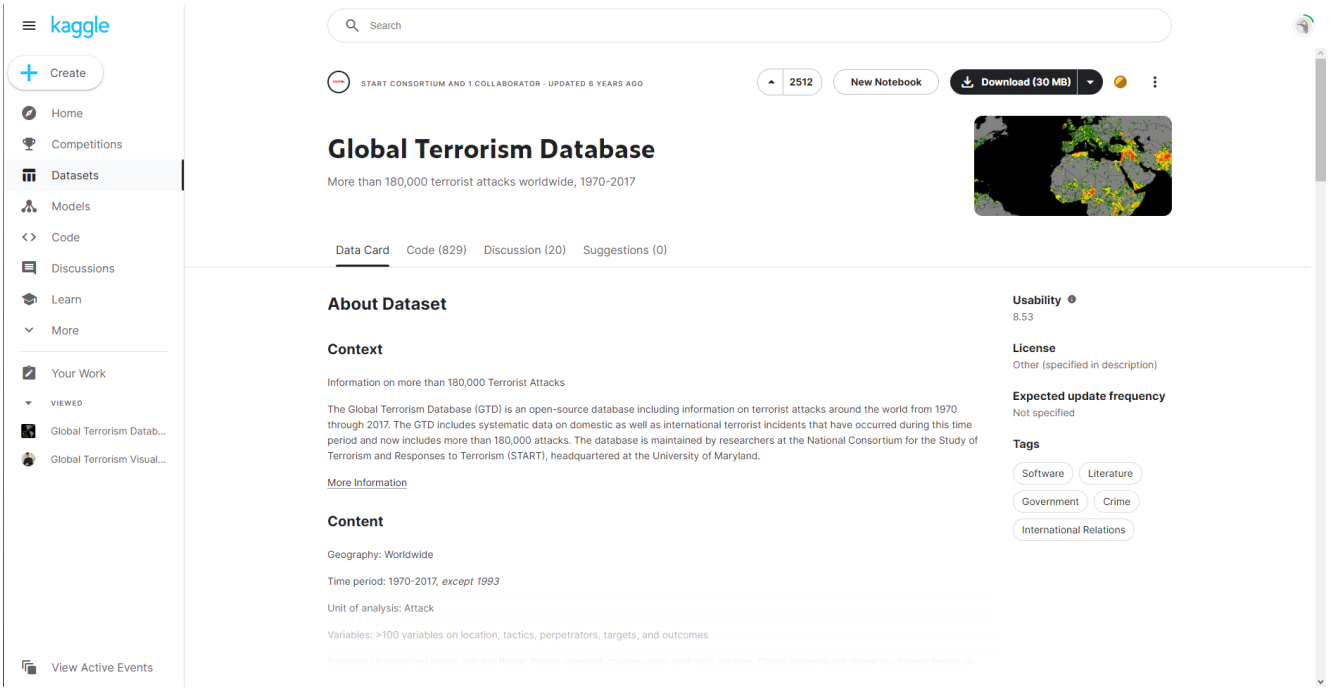
Total columns: 135

Dataset characteristics: Multivariable

Attribute number characteristics: Characters, real numbers, integers, bool

Lost value: None

Data Information: Global Terrorism Database made public in 2017 and covers over 40 years (1970-2017). The data provides information on terrorist incidents, including the time, target, weapons used, and locations of the attacks.



2.1.4 Statistics of Attribute Values

Attribute statistics table:

STT	Attribute	Data type	Meaning
1	**FactID**	Int	Attack number
2	**dYear**	Int	Year of the attack
3	**dMonth**	Int	Month of the attack
4	**dDay**	Int	Day of the attack
5	**dDate**	Date	Date of the attack
6	**CountryID**	Int	Country number
7	**CountryName**	Varchar(255)	Country where the attack occurred
8	**RegionID**		Region number
9	**RegionName**	Varchar(255)	Region where the attack occurred
10	**City**	Varchar(255)	City where the attack occurred
11	**Latitude**	Decimal(8,6)	Latitude of the attack location
12	**Longitude**	Decimal(9,6)	Longitude of the attack location
13	**ProvState**	Varchar(255)	Province/State where the attack occurred
14	**AttackTypeID**	Int	Number of the attack type
15	**AttackType**	Varchar(255)	Type of attack
16	**TargetTypeID**	Int	Number of the target type

17	**TargetType**	Varchar(255)	Target of the attack
18	**WeaponTypeID**	Int	Number of the weapon type
19	**WeaponType**	Varchar(255)	Weapon used in the attack
20	**Kills**	Int	Number of fatalities in the attack
21	**Wounds**	Int	Number of injuries in the attack
22	**Success**	Int	Successful rate of the attack
23	**Suicide**	Int	Whether the attack was a suicide attack
24	**PropertyDamage**	Int	Extent of property damage in the attack
25	**GName**	Varchar(255)	Name of the terrorist group responsible for the attack

2.2 Data preprocessing

Purpose:

- DataTransformation
- Data Collection
- Data visualization and comments

2.2.1 Import Library

```
[ ]  import pickle
     import pandas as pd
     import numpy as np
     import seaborn as sns
     from sklearn import tree
     import matplotlib.pyplot as plt
     from sklearn.tree import export_graphviz
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import train_test_split, RandomizedSearchCV
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

2.2.2 Import Dataset

```
[ ] df = pd.read_csv("https://media.githubusercontent.com/media/nhanna1219/Global-Terrorism-OLAP/main/Terrorism.csv", encoding='latin1')
    df
```

	FactID	dYear	dMonth	dDay	dDate	CountryID	CountryName	RegionID	RegionName	City	...	TargetTypeID	TargetType	WeaponTypeID	WeaponType	Kills	Wounds	Success	Suicide	PropertyDamage	GName
0	197000000001	1970	7	2	02/07/1970	58	Dominican Republic	2	Central America & Caribbean	Santo Domingo	...	14	Private Citizens & Property	13	Unknown	1.0	0.0	1	0	0	MANO-D
1	197000000002	1970	1	1	01/01/1970	130	Mexico	1	North America	Mexico city	...	7	Government (Diplomatic)	13	Unknown	0.0	0.0	1	0	0	23rd of September Communist League
2	197001000001	1970	1	1	01/01/1970	160	Philippines	5	Southeast Asia	Unknown	...	10	Journalists & Media	13	Unknown	1.0	0.0	1	0	0	Unknown
3	197001000002	1970	1	1	01/01/1970	78	Greece	8	Western Europe	Athens	...	7	Government (Diplomatic)	6	Explosives	-1.0	-1.0	1	0	1	Unknown
4	197001000003	1970	1	1	01/01/1970	101	Japan	4	East Asia	Fukouka	...	7	Government (Diplomatic)	8	Incendiary	-1.0	-1.0	1	0	1	Unknown
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
181686	201712310022	2017	12	31	31/12/2017	182	Somalia	11	Sub-Saharan Africa	Ceelka Geelow	...	4	Military	5	Firearms	1.0	2.0	1	0	-9	Al-Shabaab
181687	201712310029	2017	12	31	31/12/2017	200	Syria	10	Middle East & North Africa	Jableh	...	4	Military	6	Explosives	2.0	7.0	1	0	1	Muslim extremists
181688	201712310030	2017	12	31	31/12/2017	160	Philippines	5	Southeast Asia	Kubentog	...	14	Private Citizens & Property	8	Incendiary	0.0	0.0	1	0	1	Bangsamoro Islamic Freedom Movement (BIFM)
181689	201712310031	2017	12	31	31/12/2017	92	India	6	South Asia	Imphal	...	2	Government (General)	6	Explosives	0.0	0.0	0	0	-9	Unknown
181690	201712310032	2017	12	31	31/12/2017	160	Philippines	5	Southeast Asia	Cotabato City	...	20	Unknown	6	Explosives	0.0	0.0	0	0	0	Unknown

181691 rows × 25 columns

2.2.3 Check data type, information

```
df.dtypes
```

```
FactID              int64
dYear               int64
dMonth              int64
dDay                int64
dDate              object
CountryID           int64
CountryName        object
RegionID            int64
RegionName         object
City               object
Latitude          float64
Longitude         float64
ProvState          object
AttackTypeID        int64
AttackType         object
TargetTypeID        int64
TargetType         object
WeaponTypeID        int64
WeaponType         object
Kills             float64
Wounds            float64
Success             int64
Suicide             int64
PropertyDamage      int64
GName              object
dtype: object
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 181691 entries, 0 to 181690
Data columns (total 25 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   FactID          181691 non-null  int64
 1   dYear           181691 non-null  int64
 2   dMonth          181691 non-null  int64
 3   dDay            181691 non-null  int64
 4   dDate           181691 non-null  object
 5   CountryID       181691 non-null  int64
 6   CountryName     181691 non-null  object
 7   RegionID        181691 non-null  int64
 8   RegionName      181691 non-null  object
 9   City            181256 non-null  object
 10  Latitude        181691 non-null  float64
 11  Longitude       181691 non-null  float64
 12  ProvState       181270 non-null  object
 13  AttackTypeID    181691 non-null  int64
 14  AttackType      181691 non-null  object
 15  TargetTypeID    181691 non-null  int64
 16  TargetType      181691 non-null  object
 17  WeaponTypeID    181691 non-null  int64
 18  WeaponType      181691 non-null  object
 19  Kills           181691 non-null  float64
 20  Wounds          181691 non-null  float64
 21  Success         181691 non-null  int64
 22  Suicide         181691 non-null  int64
 23  PropertyDamage  181691 non-null  int64
 24  GName           181691 non-null  object
dtypes: float64(4), int64(12), object(9)
memory usage: 34.7+ MB
```

2.2.4 Overview of the data

Data is a combination of string and integer values.

- **FactID:** Serial number of the attack in the dataset.
- **dYear:** Year of the attack in the dataset.
- **dMonth:** Month of the attack in the dataset.
- **dDay:** Day of the attack in the dataset.
- **dDate**: The actual date of the attack in the dataset which is shown day, month and year.
- **CountryID**: Serial number of the country in the dataset.
- **CountryName**: Name of the country where the attack occurred.
- **RegionID**: Serial number of the region in the dataset.
- **RegionName**: Name of the region where the attack occurred.
- **City**: Name of the city where the attack occurred
- **Latitude**: This value describes the latitude of the attack location.
- **Longitude:** This value describes the longitude of the attack location.
- **ProvState:** Name of the Province/State where the attack occurred.
- **AttackTypeID:** Serial number of the attack type in the dataset.

- **AttackType:** Name of the attack type where the attack occurred.
- **TargetTypeID:** Serial number of the target type in the dataset.
- **TargetType:** Name of the target type where the attack occurred.
- **WeaponTypeID:** Serial number of the weapon in the dataset.
- **WeaponType:** Name of the weapon used in the attack.
- **Kills:** The actual number of fatalities in the attack.
- **Wounds:** The actual number of injuries in the attack.
- **Success:** The actual successful rate of the attack. **This is our target variable.**
- **Suicide:** Whether the attack was a suicide attack.
- **PropertyDamage:** The actual property damage in the attack.
- **GName:** Name of the terrorist group responsible for the attack.

2.2.5 Description of dataset information

```
# Information description of the numeric data
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
FactID	181691.0	2.002705e+11	1.325957e+09	1.970000e+11	1.991021e+11	2.009022e+11	2.014081e+11	2.017123e+11
dYear	181691.0	2.002639e+03	1.325943e+01	1.970000e+03	1.991000e+03	2.009000e+03	2.014000e+03	2.017000e+03
dMonth	181691.0	6.467387e+00	3.388110e+00	1.000000e+00	4.000000e+00	6.000000e+00	9.000000e+00	1.200000e+01
dDay	181691.0	1.551055e+01	8.805691e+00	1.000000e+00	8.000000e+00	1.500000e+01	2.300000e+01	3.100000e+01
CountryID	181691.0	1.319685e+02	1.124145e+02	4.000000e+00	7.800000e+01	9.800000e+01	1.600000e+02	1.004000e+03
RegionID	181691.0	7.160938e+00	2.933408e+00	1.000000e+00	5.000000e+00	6.000000e+00	1.000000e+01	1.200000e+01
Latitude	181691.0	2.290911e+01	1.869944e+01	-5.315461e+01	9.518645e+00	3.112665e+01	3.453856e+01	7.463355e+01
Longitude	181691.0	-4.471911e+02	2.021946e+05	-8.618590e+07	1.231572e+00	4.314357e+01	6.835734e+01	1.793667e+02
AttackTypeID	181691.0	3.247547e+00	1.915772e+00	1.000000e+00	2.000000e+00	3.000000e+00	3.000000e+00	9.000000e+00
TargetTypeID	181691.0	8.439719e+00	6.653838e+00	1.000000e+00	3.000000e+00	4.000000e+00	1.400000e+01	2.200000e+01
WeaponTypeID	181691.0	6.447325e+00	2.173435e+00	1.000000e+00	5.000000e+00	6.000000e+00	6.000000e+00	1.300000e+01
Kills	181691.0	2.210098e+00	1.124089e+01	-1.000000e+00	0.000000e+00	0.000000e+00	2.000000e+00	1.570000e+03
Wounds	181691.0	2.793523e+00	3.431848e+01	-1.000000e+00	0.000000e+00	0.000000e+00	2.000000e+00	8.191000e+03
Success	181691.0	8.895983e-01	3.133907e-01	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
Suicide	181691.0	3.650704e-02	1.875486e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
PropertyDamage	181691.0	-5.445564e-01	3.122889e+00	-9.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

```
[ ] # Information description of the string data
    df.describe(include=['O'])
```

	dDate	CountryName	RegionName	City	ProvState	AttackType	TargetType	WeaponType	GName
count	181691	181691	181691	181256	181270	181691	181691	181691	181691
unique	16081	205	12	36673	2855	9	22	12	3537
top	21/09/2016	Iraq	Middle East & North Africa	Unknown	Baghdad	Bombing/Explosion	Private Citizens & Property	Explosives	Unknown
freq	130	24636	50474	9775	7645	88255	43511	92426	82782

We can see that there are 24 features and 1 label column (Success). Out of the features, 16 are numerical and 9 are categorical.

2.2.6 Handle the null data

Check the null data

```
df[df.isnull().any(axis=1)]
```

	FactID	dYear	dMonth	dDay	dDate	CountryID	CountryName	RegionID	RegionName	City	...	TargetTypeID	TargetType	WeaponTypeID	WeaponType	Kills	Wounds	Success	Suicide	PropertyDamage
0	197000000001	1970	7	2	02/07/1970	58	Dominican Republic	2	Central America & Caribbean	Santo Domingo	...	14	Private Citizens & Property	13	Unknown	1.0	0.0	1	0	0
162	197003310002	1970	3	31	31/03/1970	101	Japan	4	East Asia	Fukouka	...	6	Airports & Aircraft	6	Explosives	0.0	0.0	1	0	0
1229	197204040002	1972	4	4	04/04/1972	38	Canada	1	North America	Montreal	...	2	Government (General)	6	Explosives	1.0	7.0	1	0	1
1568	197209280011	1972	9	28	28/09/1972	217	United States	1	North America	Washington	...	7	Government (Diplomatic)	6	Explosives	0.0	0.0	0	0	0
1617	197211080002	1972	11	8	08/11/1972	130	Mexico	1	North America	Monterrey	...	6	Airports & Aircraft	5	Firearms	0.0	0.0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
103530	201110210005	2011	10	21	21/10/2011	93	Indonesia	5	Southeast Asia	NaN	...	14	Private Citizens & Property	5	Firearms	3.0	1.0	1	0	0
103543	201110230002	2011	10	23	23/10/2011	4	Afghanistan	6	South Asia	NaN	...	2	Government (General)	6	Explosives	1.0	0.0	0	1	-9
104385	201112030021	2011	12	3	03/12/2011	93	Indonesia	5	Southeast Asia	NaN	...	3	Police	12	Other	2.0	0.0	1	0	0
170978	201701070015	2017	1	7	07/01/2017	217	United States	1	North America	Hudson Bend	...	15	Religious Figures/Institutions	8	Incendiary	0.0	0.0	1	0	1
171830	201702060025	2017	2	6	06/02/2017	217	United States	1	North America	Manchester	...	1	Business	8	Incendiary	0.0	0.0	1	0	1

856 rows × 25 columns

There are 856 rows that contain NaN values. We'll replace the "NaN" with some suitable values such as "-1" for numberic values, "N/A" for text values.

```
[ ] # Replace null numeric values with -1
    numeric_columns = ['nkill', 'nwound']
    for column in numeric_columns:
        df[column] = df[column].fillna(-1)

    # Replace null text values with 'N/A'
    text_columns = [col for col in df.columns if col not in numeric_columns]
    for column in text_columns:
        df[column] = df[column].fillna('N/A')

    # Replace day/month/year with 0 value to default day/month/year(1/1/1900)
    df['iyear'] = df['iyear'].replace(0, 1900)
    df['imonth'] = df['imonth'].replace(0, 1)
    df['iday'] = df['iday'].replace(0,1)

    # Create the 'date' column by concatenating the string columns with dashes as separators
    df['date'] = df['iyear'].astype(str) + '/' + df['imonth'].astype(str).str.zfill(2) + '/' + df['iday'].astype(str).str.zfill(2)
    df['date'] = pd.to_datetime(df['date'], format='%Y/%m/%d')
    df['date'] = df['date'].dt.strftime('%d/%m/%Y')

    # Rearrange column date
    columns = [col for col in df.columns if col != 'date']
    dDay_index = columns.index('iday') # Insert after day column
    columns.insert(dDay_index + 1, 'date')
    df = df[columns]
```

2.2.7 Check the balance data of the predictor attribute

```
[ ] # Check if data is balanced
    df['Success'].value_counts()
```

```
Success
1    160838
0     19997
Name: count, dtype: int64
```

There are 2 kind of success with equal distribution (balanced data). This means the accuracy score will be a good metric to use.

CHAPTER III: ALGORITHMS AND EXPERIMENTS

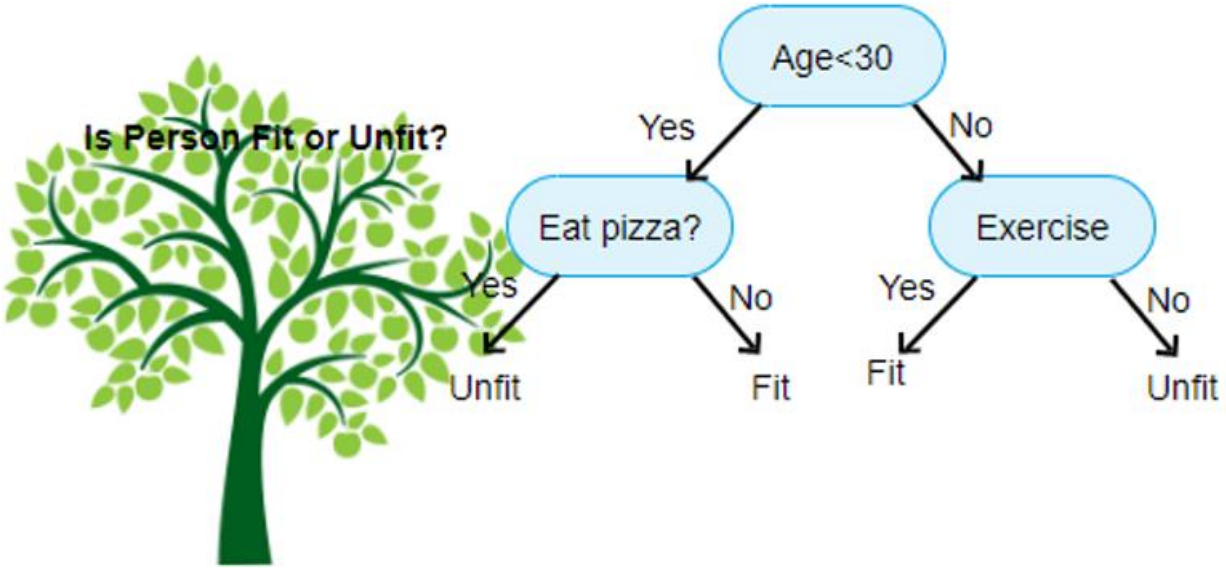
3.1 Algorithm used

3.1.1 Decision Tree

A decision tree is a tree structure such that:

- Each node in the network corresponds to a test on an attribute.
- Each branch represents the test result.
- Leaf nodes represent classes or class distributions.
- The highest node in the tree is the root node.

Decision tree shape:



Basic strategy:

- Start from a single node showing all samples.
- If the samples belong to the same class, the node becomes a leaf node and is labeled with that class.
- In contrast, using the attribute measure to select the attribute will best separate the samples into classes.

- A branch is created for each value of the selected attribute and the samples are partitioned by use the same process recursively to create a decision tree.
- The process ends only if any of the following conditions are true.

❖ All templates for a given node belong to the same class

❖ There are no more attributes that the sample can rely on for further partitioning

❖ No samples left at node

❖ ID3 is an algorithm used in decision trees. This algorithm uses information gain to build a decision tree. The largest Information Gain attribute will be selected as the root node

Information Gain:

$$Info(S) = E(S) = -\sum_{i=1}^{n} f_S(A_i) \log_2 f_S(A_i)$$

Amount of information needed to classify an element in S based on attribute A: InfoA(S)

$$Info_A(S) = -\sum_{i=1}^{v} \frac{|S_j|}{|S|} Info(S_i)$$

Information gain is the difference between the original Info(S) information value (before partitioning) and the new InfoA(S) information value (after partitioning with A)

$$Gain(A) = Info(S) - Info_A(S)$$

CART: Unlike ID3 which uses Information Cain formula, Cart algorithm uses Gini formula. The attribute with the smallest Gini value will be the root node

Gini index of the set S:

$$Gini(S) = 1 - \sum_{j} p(j|S)^2$$

P(j|S) is the frequency of j in S

Gini of attribute:

$$Gini_A(S) = \sum_{i=1}^{k} \frac{n_i}{n} Gini(i)$$

In case: ni is the number of samples in note I, n is the number of samples in note A

3.1.2 Random Forest

3.1.2.1 Random Forest explaination

Random Forest is one of the most popular and commonly used algorithms by Data Scientists. **Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems**. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification. It performs better for classification and regression tasks.

- **Steps Involved in Random Forest Algorithm**

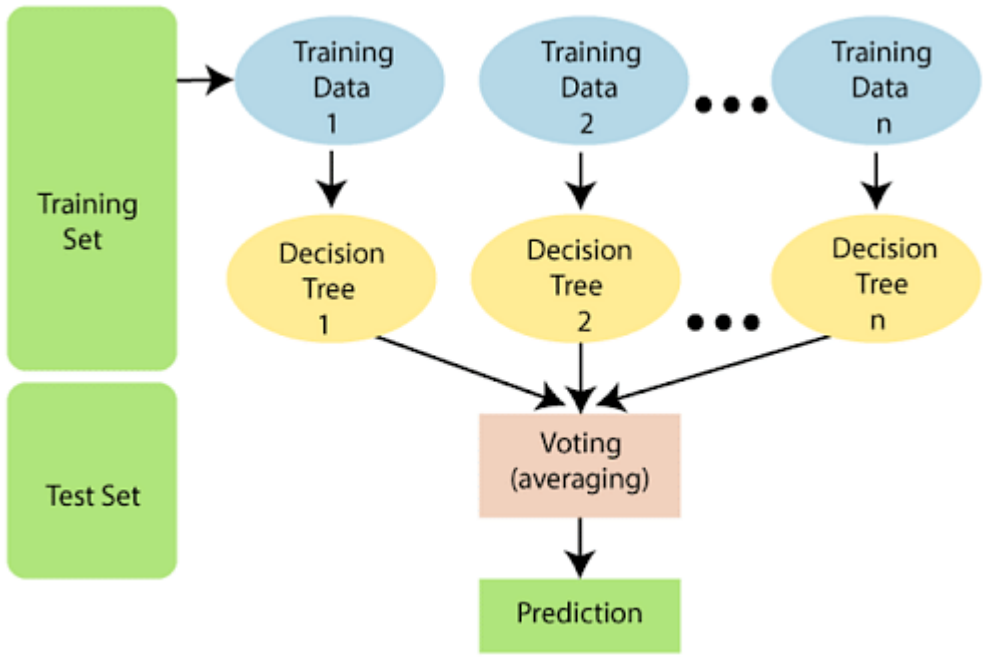
Step 1: In the Random forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression, respectively.

3.1.2.2 Random Forests Algorithm



The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

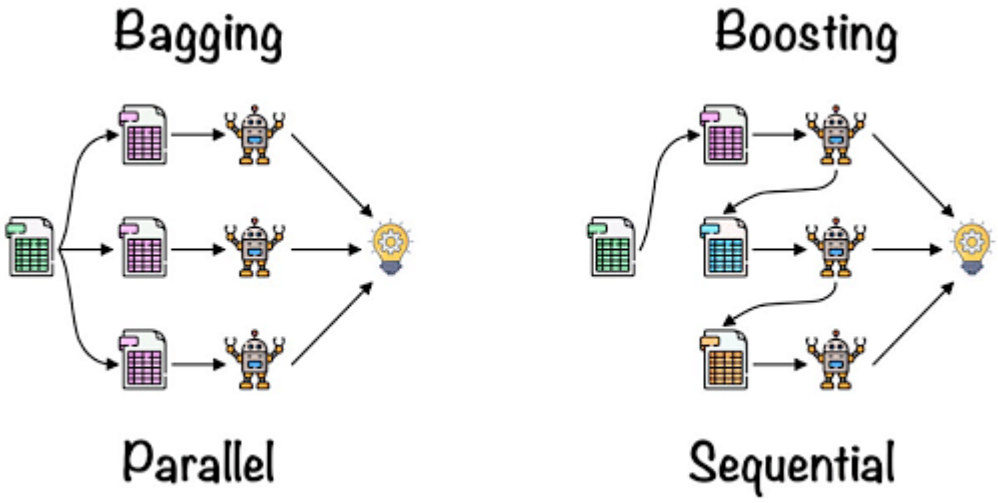
Step 2: This algorithm will construct a decision tree for every training data.

Step 3: Voting will take place by averaging the decision tree.

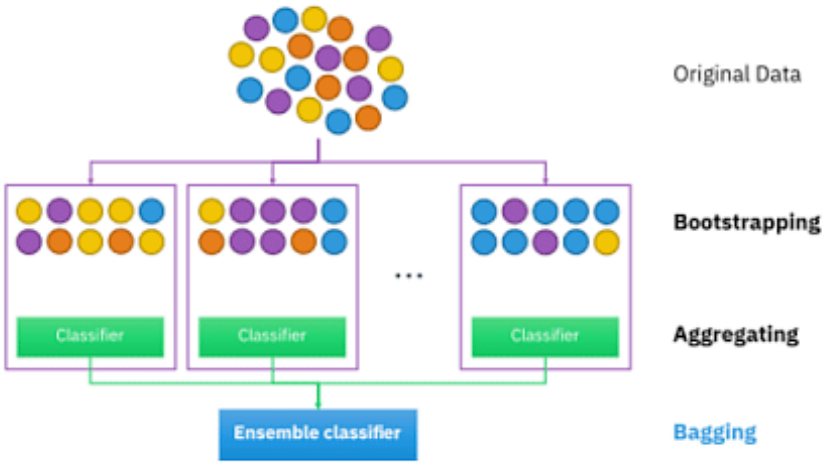
Step 4: Finally, select the most voted prediction result as the final prediction result.

This combination of multiple models is called Ensemble. Ensemble uses two methods:

- **Bagging:** Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.
- **Boosting:** Combing weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.



BaggingBagging is also known as Bootstrap Aggregation used by random forest. The process begins with any original random data. After arranging, it is organised into samples known as Bootstrap Sample. This process is known as Bootstrapping.Further, the models are trained individually, yielding different results known as Aggregation. In the last step, all the results are combined, and the generated output is based on majority voting. This step is known as Bagging and is done using an Ensemble Classifier.



❖ **Essential Features of Random Forest**

- **Miscellany:** Each tree has a unique attribute, variety and features concerning other trees. Not all trees are the same.

- **Immune to the curse of dimensionality**: Since a tree is a conceptual idea, it requires no features to be considered. Hence, the feature space is reduced.
- **Parallelization:** We can fully use the CPU to build random forests since each tree is created autonomously from different data and features.
- **Train-Test split:** In a Random Forest, we don't have to differentiate the data for train and test because the decision tree never sees 30% of the data.
- **Stability:** The final result is based on Bagging, meaning the result is based on majority voting or average.

❖ **How Random Forest is applied?**

Random Forest has a wide range of applications across various domains due to its versatility and robustness.

- **Classification Problems**: Random Forest is often used for classification tasks, such as spam detection, sentiment analysis, customer churn prediction, disease diagnosis, and image classification. Its ability to handle both numerical and categorical features makes it suitable for diverse datasets.
- **Regression Problems:** Random Forest can be applied to regression tasks, including predicting housing prices, stock market trends, energy consumption, and demand forecasting. It can capture complex non-linear relationships between input variables and the target variable.
- **Feature Importance**: Random Forest provides a measure of feature importance, which can be utilized for feature selection in data preprocessing. This information helps identify the most relevant features for prediction and can guide feature engineering efforts.
- **Anomaly Detection**: Random Forest can be used for anomaly detection by training on normal data and identifying instances that deviate significantly from the learned patterns. This is useful in fraud detection, network intrusion detection, and detecting unusual behaviors in various domains.
- **Ensemble Learning**: Random Forest is a type of ensemble learning method, which combines multiple models to improve overall performance. It can be used

as a base learner in ensemble techniques such as bagging, boosting, and stacking to further enhance predictive accuracy.

- **Imputation of Missing Values**: Random Forest can handle missing data effectively. It can be used to impute missing values in a dataset by using the available features to predict missing values, making it valuable for data preprocessing tasks.
- **Recommender Systems**: Random Forest can be employed in building recommendation systems to suggest relevant products, movies, or content to users based on their preferences and behavior.
- **Bioinformatics and Genomics**: Random Forest finds applications in analyzing DNA sequences, gene expression data, and protein-protein interactions. It can be used for tasks like gene expression classification, protein structure prediction, and identifying disease biomarkers.

3.1.3 Naïve Bayes

3.1.3.1 Bayes Theorem

Bayes' Theorem (Bayes' Theorem) is a mathematical theorem that calculates the probability of a random event A, given that the related event B has occurred.

- This theorem is named after the 18th century English mathematician Thomas Bayes.
- This is one of the extremely useful tools, a close friend of Data Scientists who work in data science.
- Bayes theorem allows to calculate the probability of a random event A given that related event B has occurred. This probability is denoted P(A|B) and read as “the probability of A if there is B”. This quantity is called conditional probability or posterior probability because it is derived from a given value of B or depends on that value.

According to Bayes' theorem, the probability that A occurs when B is known will depend on 3 factors:

- **The probability that A occurs on its own, regardless of B**. It is denoted by P(A) and read as the probability of A. This is called the marginal probability or a priori probability, it is "a priori" " in the sense that it is not interested in any information about B.
- **Probability of occurring B on its own, regardless of A**. It is denoted by P(B) and read as "probability of B". This quantity is also called a normalizing constant because it is always the same, regardless of the event A is trying to know.
- **Probability of B happening when A is known**. It is denoted by P(B|A) and read as "probability of B if there is A". This quantity is called the likelihood that B will occur, given that A has occurred. Pay attention not to confuse the probability that B will occur when A is known and the probability that A will occur when B is known.

We can restate it with the following formula: The probability that A and B occur at the same time is:

$$P(A,B) = P(A)\,P(B)$$

- **In case:**

- P(A)P(A) is the probability of a distinct A occurring.
- P(B)P(B) is the probability that B occurs separately.

If A and B are two related events, and the probability that event B occurs is greater than 0, we can define the probability that A will occur, given that B occurs as follows:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

Bayes theorem is based on the definition of conditional probability above, expressed in the form of a formula as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

The symbol ¬A is not A (or A's complement). We have P(A)+P(¬A) = 1

From there: P(B) =P(B,A) + P(B,¬A) = P(B|A)P(A) + P(B|¬A)P(¬A)

Bayes' theorem is written in variant form as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}$$

3.1.3.2 Naïve Bayes classification algorithm

Naïve Bayes Classification (NBC) is a classification algorithm based on probability calculation applying Bayes theorem. This algorithm belongs to the group of supervised learning algorithms.

- Each data sample is represented by X=(x1, x2,..., xn) with attributes A1, A2,..., An
- Grades C1, C2, ..., Cm. Given an unknown sample X.
- Subclassing Naive Bayes will determine that X belongs to class Ci if and only if:

$$P(C_i|X) > P(C_j|X), \text{ với mọi } 1 \le i, j \le m, j \ne i$$

$$P(C_i|X) = \frac{P(X|Ci) \text{ x } P(Ci)}{P(X)}$$

❖ **According to Bayes' theorem**

Since P(X) is constant for all classes, **only the maximum P(X|Ci) x P(Ci) is needed. If P(Ci) is not known, we need to assume P(C1)=P(C2)=...= P(Cm)** and **we will maximize P(X|Ci). Otherwise, we maximize P(X|Ci) x P(Ci)**

However, the problem of calculating P(X|Ci) is impossible!

Admit Naïve: assume attribute independence

$$P(X|C_i) = \prod_{k=1}^{n} P(X_k|C_i)$$

It is possible to approximate P(x1|Ci),…,P(xn|Ci) from the training samples.

If Ak is a qualitative attribute, then P(xk|Ci) = sik/si where sik is the number of training samples of Ci with the value xk for Ak and si is the number of samples belonging to class Ci

If Ak is continuous, then it is assumed to have a Gaussian distribution:

$$P(X_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} e^{-\frac{(x_k-\mu_{C_i})^2}{2\sigma^2 C_i}}$$

3.1.4 K-Nearest Neighbors

3.1.4.1 K-Nearest Neighbors explain

The k-nearest neighbors algorithm, also known as KNN or K-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered "plurality voting", the term, "majority vote" is more commonly used in literature. The distinction between these terminologies is that "majority voting" technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes—e.g. four categories, you don't necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with a vote of greater than 25%.

3.1.4.2 K-nearest neighbors Algorithm

Compute KNN: distance metrics

To recap, the goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point. In order to do this, KNN has a few requirements.

Determine your distance metrics

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partition query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagrams.

While there are several distance measures that you can choose from, this article will only cover the following:

- **Euclidean distance (p=2):** This is the most commonly used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

- **Manhattan distance (p=1)**: This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block distance as it is commonly visualized with a grid, illustrating how one might navigate from one address to another via city streets.

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|\right)$$

- **Minkowski distance**: This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

$$\text{Minkowski Distance} = \left(\sum_{i=1}^{n} |x_i - y_i|\right)^{1/p}$$

- **Hamming distance:** This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result,

it has also been referred to as the overlap metric. This can be represented with the following formula:

$$\text{Hamming Distance} = D_H = \left(\sum_{i=1}^{k} |x_i - y_i|\right)$$

$$x=y \quad D=0$$
$$x \neq y \quad D \neq 1$$

3.2 Experiments on Jupyter Notebook

- Graph, count, and view label ratios to get an overview of terrorism Success.
- Build decision properties, with the decision property as Success.
- Preprocessing before training.

3.2.1 Separating train and test data

Train data accounts for 70%, test accounts for 30%

```
[ ]  # Tách dữ liệu train và test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3.2.2 Decision Trees Algorithm

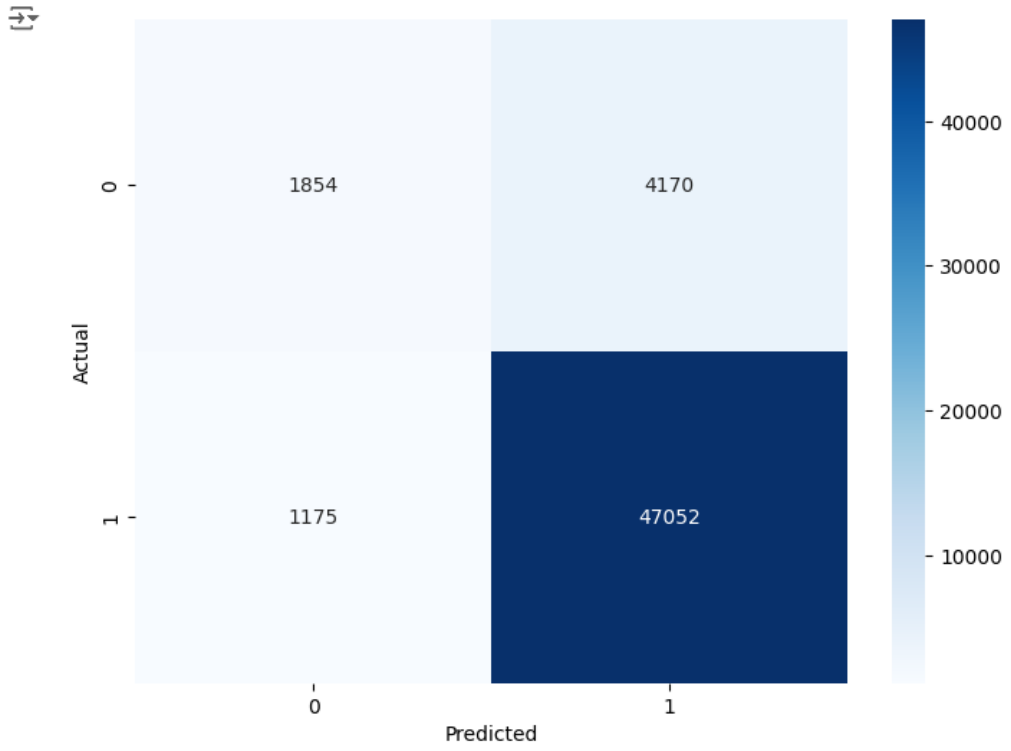
```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     from sklearn import tree
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
     import time
     from datetime import timedelta
     from IPython.display import display
     pd.set_option('display.max_columns', None)
     pd.set_option('display.max_rows', None)
```

```
[ ]
    dt_model = tree.DecisionTreeClassifier(criterion="gini", random_state=0)
    start_dt = time.time()
    dt_pred = dt_model.fit(X_train,y_train).predict(X_test)
    end_dt = time.time()
    times_dt = timedelta(seconds=round(end_dt - start_dt, 4)).total_seconds()
    print("Times: ", times_dt)
    dt_score = round(accuracy_score(y_test, dt_pred)*100,2 )
    accuracy_dt = dt_score
    print("Accuracy: ", accuracy_dt)
    print("Report: ")
    print(classification_report(y_test, dt_pred, digits=5))
```

```
Times:  0.2894
Accuracy:  90.15
Report:
              precision    recall  f1-score   support

           0    0.61208   0.30777   0.40959      6024
           1    0.91859   0.97564   0.94625     48227

    accuracy                        0.90148     54251
   macro avg    0.76534   0.64170   0.67792     54251
weighted avg    0.88456   0.90148   0.88666     54251
```

- Algorithm accuracy: 90.15%
- Algorithm runtime: 0.2894s

```
# Feature importance
feature_importance = best_rf_model.feature_importances_
features = X.columns
feature_df = pd.DataFrame({'Feature': features, 'Importance': feature_importance}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_df)
plt.title('Feature Importance')
plt.show()
```



Through the confused matrix of the Decision Tree algorithm picture tissue (CART), we know:

- Accuracy of the algorithmic Picture tissue: 90.15%

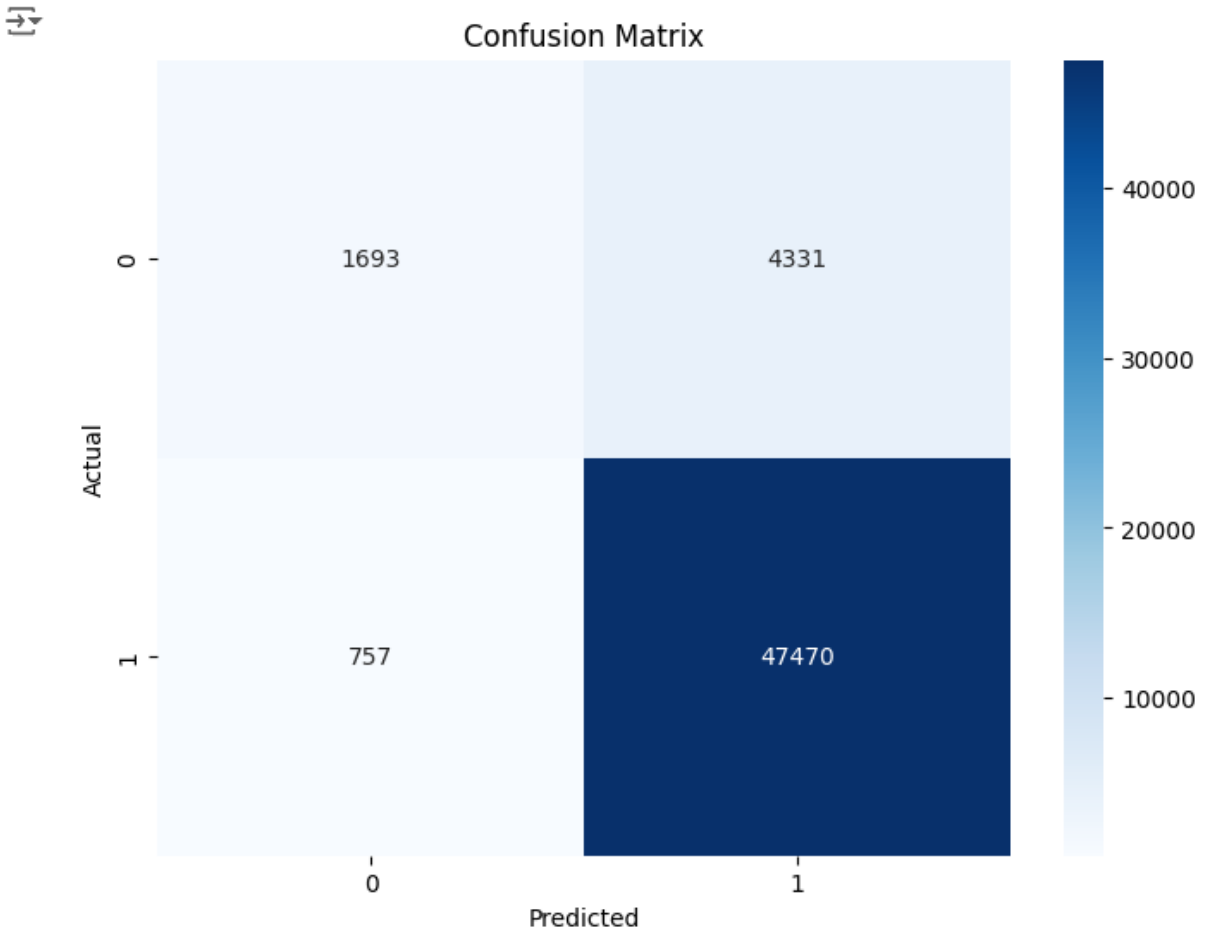
3.2.3 Random Forest algorithm

```
import time
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from datetime import timedelta  # Import timedelta

with open('best_params.pkl', 'rb') as f:
    best_params = pickle.load(f)

# Train the model with best parameters
best_rf_model = RandomForestClassifier(**best_params, random_state=42)
best_rf_model.fit(X_train, y_train)

start_rf = time.time()

# Predictions
y_pred = best_rf_model.predict(X_test)

end_rf = time.time()
times_rf = timedelta(seconds=round(end_rf - start_rf, 4)).total_seconds()
print("Times: ", times_rf)
rf_score = round(accuracy_score(y_test, y_pred)*100,2 )
accuracy_rf = rf_score
print("Accuracy: ", accuracy_rf)
print("Report: ", classification_report(y_test, y_pred))
```

```
Times:  0.1707
Accuracy:  90.62
Report:                precision    recall  f1-score   support

           0       0.69      0.28      0.40      6024
           1       0.92      0.98      0.95     48227

    accuracy                           0.91     54251
   macro avg       0.80      0.63      0.67     54251
weighted avg       0.89      0.91      0.89     54251
```

- Algorithm accuracy: 90.62%
- Algorithm runtime: 0.1707s

```
[ ]  conf_matrix = confusion_matrix(y_test, y_pred)

     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
     plt.title("Confusion Matrix")
     plt.xlabel("Predicted")
     plt.ylabel("Actual")
     plt.show()
```



Through the confused matrix of random forest algorithm picture tissue, we know:

- Accuracy of the algorithmic Picture tissue: 90.62%

3.2.4 Naïve Bayes Algorithm

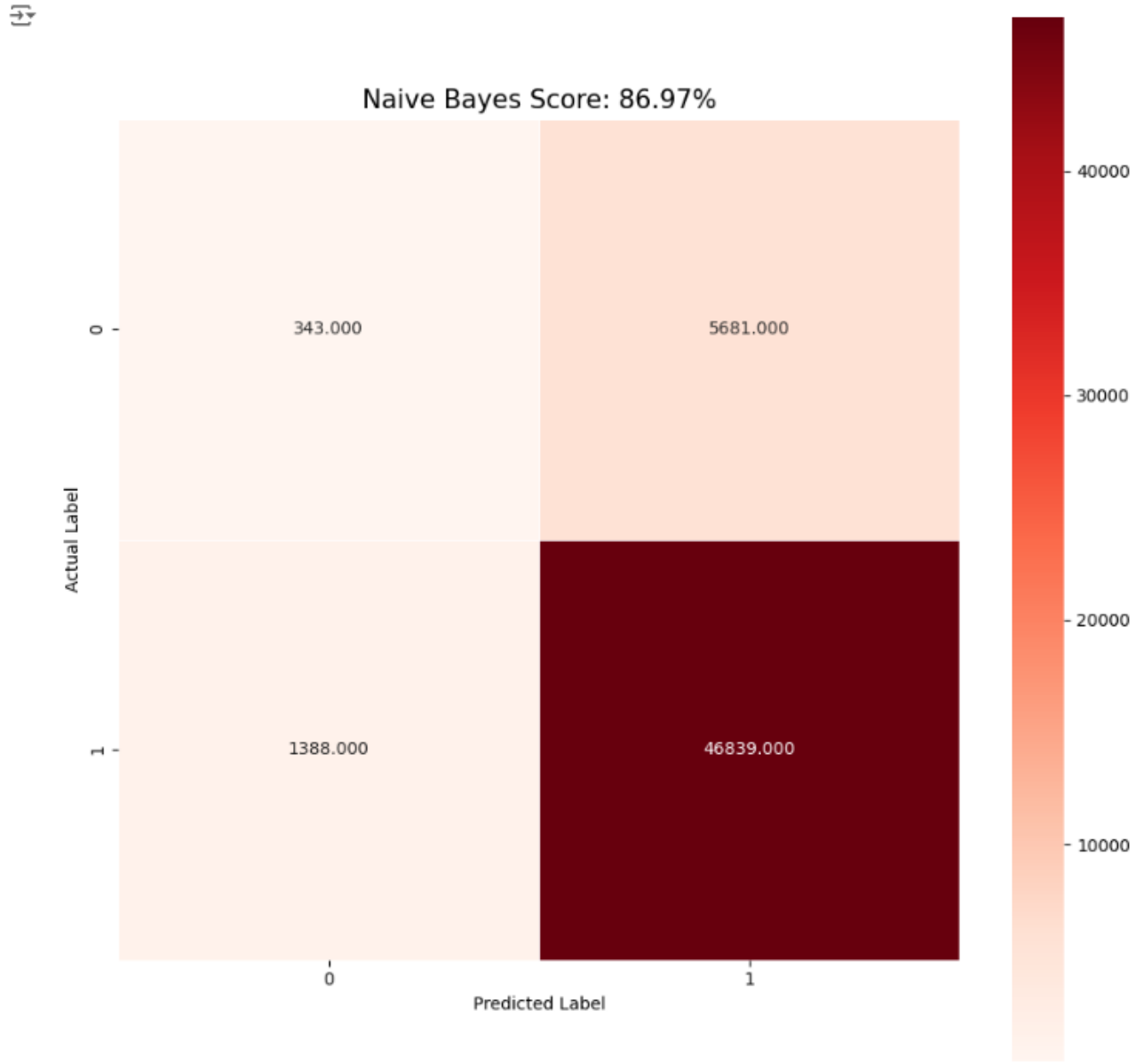
```
[ ]  import pickle
     import pandas as pd
     import numpy as np
     import time
     import pickle
     from datetime import timedelta  # Import timedelta
     import seaborn as sns
     from sklearn import tree
     import matplotlib.pyplot as plt
     from sklearn.tree import export_graphviz
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import train_test_split, RandomizedSearchCV
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
[ ]  # Initialize Naive Bayes Algorithms
     from sklearn.naive_bayes import GaussianNB
     nv = GaussianNB()
     start_nv = time.time()
     nv_pred = nv.fit (X_train, y_train).predict(X_test)
     end_nv = time.time()
     times_nv = timedelta(seconds=round(end_nv - start_nv,4)).total_seconds()
     print("Time Naive Bayes: ", times_nv)
     nv_score = round(accuracy_score(y_test, nv_pred)*100,2 )
     accuracy_nv = nv_score
     print("Accuracy: ", accuracy_nv)
     print("Report: ", classification_report(y_test, nv_pred))
```

```
Time Naive Bayes:  0.0383
Accuracy:  86.97
Report:                precision    recall  f1-score   support

           0       0.20      0.06      0.09      6024
           1       0.89      0.97      0.93     48227

    accuracy                           0.87     54251
   macro avg       0.54      0.51      0.51     54251
weighted avg       0.81      0.87      0.84     54251
```

- Algorithm accuracy: 86.97%
- Algorithm runtime: 0.0383s

```
[ ]  # Draw the confusion matrix for the Naive Bayes Algorithms
     nv_cm = confusion_matrix(y_test, nv_pred)
     plt.figure(figsize=(11,11))
     ax=sns.heatmap(nv_cm, annot=True, fmt=".3f", linewidth=.5, square=True, cmap="Reds")
     ax.set_ylabel("Actual Label")
     ax.set_xlabel("Predicted Label")
     title = 'Naive Bayes Score: {0}%'.format(nv_score)
     plt.title(title, size=15)
     plt.show()
```



Through the confused matrix of the Naive Bayes algorithmic Picture tissue, we learn :

- Accuracy of the algorithmic Picture tissue: 86.97%

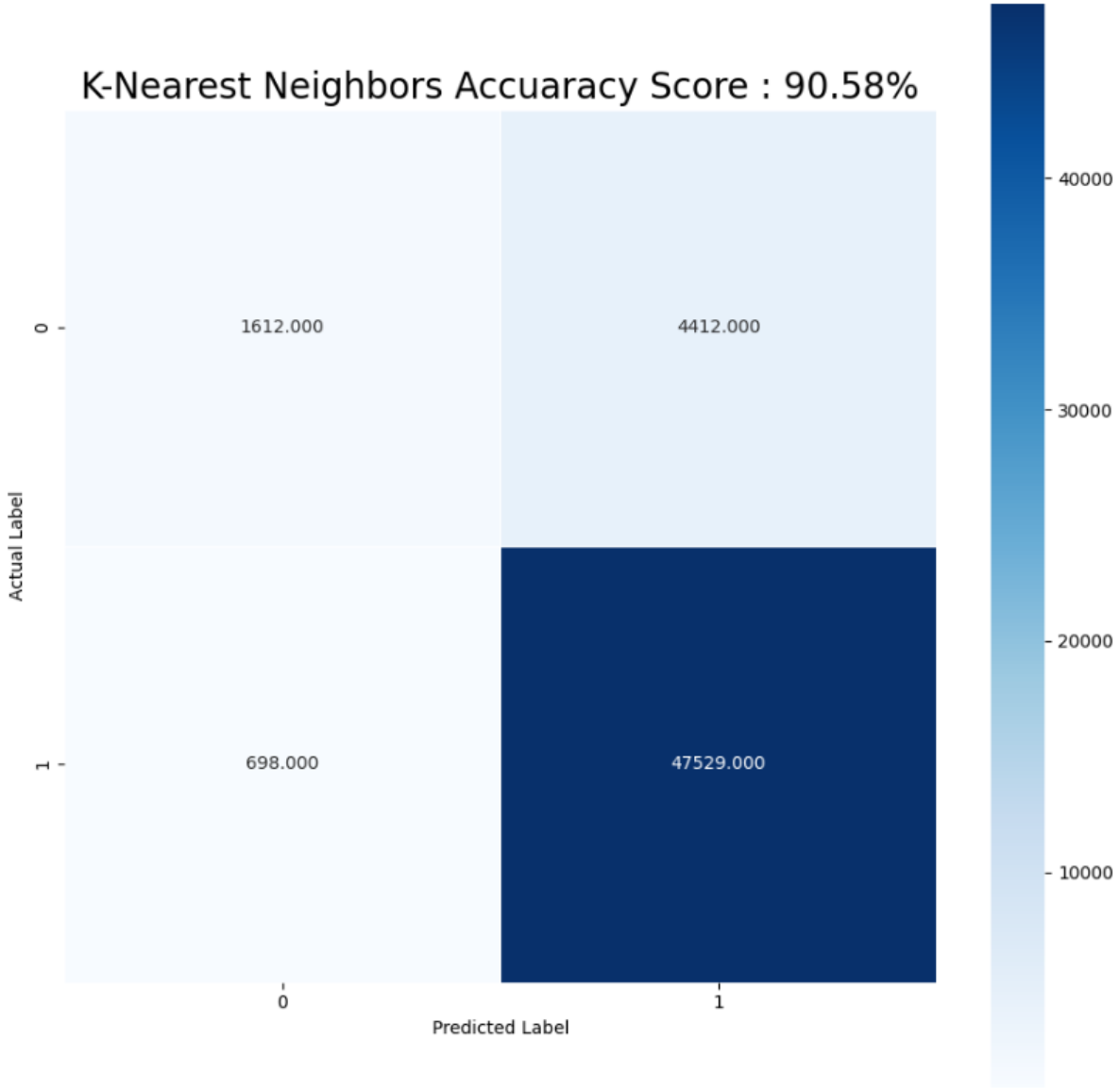
3.2.5 K-Nearest Neighbors Algorithm

```
[ ] # Initialize KNN Algorithms
    from sklearn.neighbors import KNeighborsClassifier
    import time
    from sklearn.metrics import classification_report
    from datetime import timedelta
    import matplotlib.pyplot as plt
    import sklearn.metrics as metrics
    from sklearn.metrics import accuracy_score
    start_knn=time.time()
    knn_scores=[]

    for i in range(1,12):
        knc=KNeighborsClassifier(i)
        knn_pred=knc.fit(X_train,y_train).predict(X_test)
        knn_scores.append(metrics.accuracy_score(y_test,knn_pred))
        max_knn_score=max (knn_scores)

    knn_scores_ind=[i for i, v in enumerate(knn_scores) if v == max_knn_score]
    end_knn=time.time()
    times_knn=timedelta(seconds=round(end_knn-start_knn,4)).total_seconds()
    print("heighest Accuracy Score : {}% with k = {}".format(max_knn_score*100,list(map(lambda x:x+1,knn_scores_ind))))
    print("Time", times_knn)
    knn_score=round(max_knn_score*100,2)
    accuracies_max_knn=knn_score
    print("Accuracy",accuracies_max_knn,"%")
    print("Report",metrics.classification_report(y_test,knn_pred))
```

```
heighest Accuracy Score : 90.58081878675047% with k = [11]
Time 62.0339
Accuracy 90.58 %
Report               precision    recall  f1-score   support

           0       0.70      0.27      0.39      6024
           1       0.92      0.99      0.95     48227

    accuracy                           0.91     54251
   macro avg       0.81      0.63      0.67     54251
weighted avg       0.89      0.91      0.89     54251
```

- Algorithm accuracy: 90.58%
- Algorithm runtime: 62.0339s

```
knn_cm=metrics.confusion_matrix(y_test,knn_pred)
plt.figure(figsize=(11,11))
ax= sns.heatmap(knn_cm, annot=True, fmt=".3f",linewidths=.5,square=True,cmap='Blues')
ax.set_ylabel("Actual Label")
ax.set_xlabel("Predicted Label")
title="K-Nearest Neighbors Accuaracy Score : {0}%".format(knn_score)
plt.title(title,size=20)
```



Through the confused matrix of the K-Nearest Neighbors algorithmic Picture tissue, we learn :

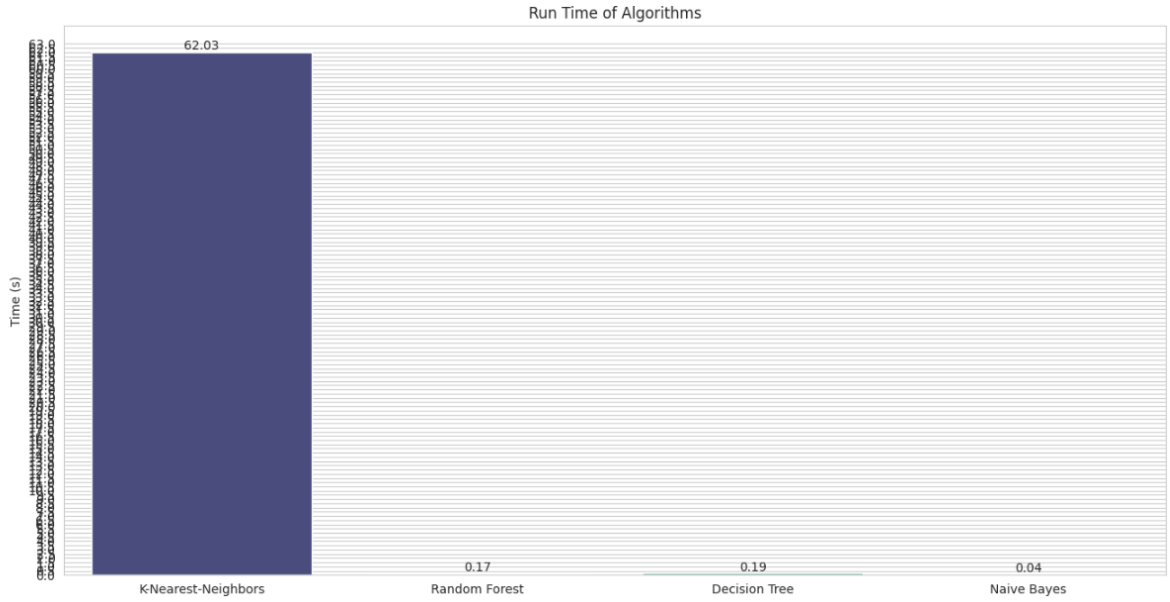
- Accuracy of the algorithmic Picture tissue: 90.58%

3.2.6 Comparison, Evaluation

Use the BarPlot graph to get an overview of runtime and accuracy between algorithms.

Draw a chart comparing the running time of algorithms

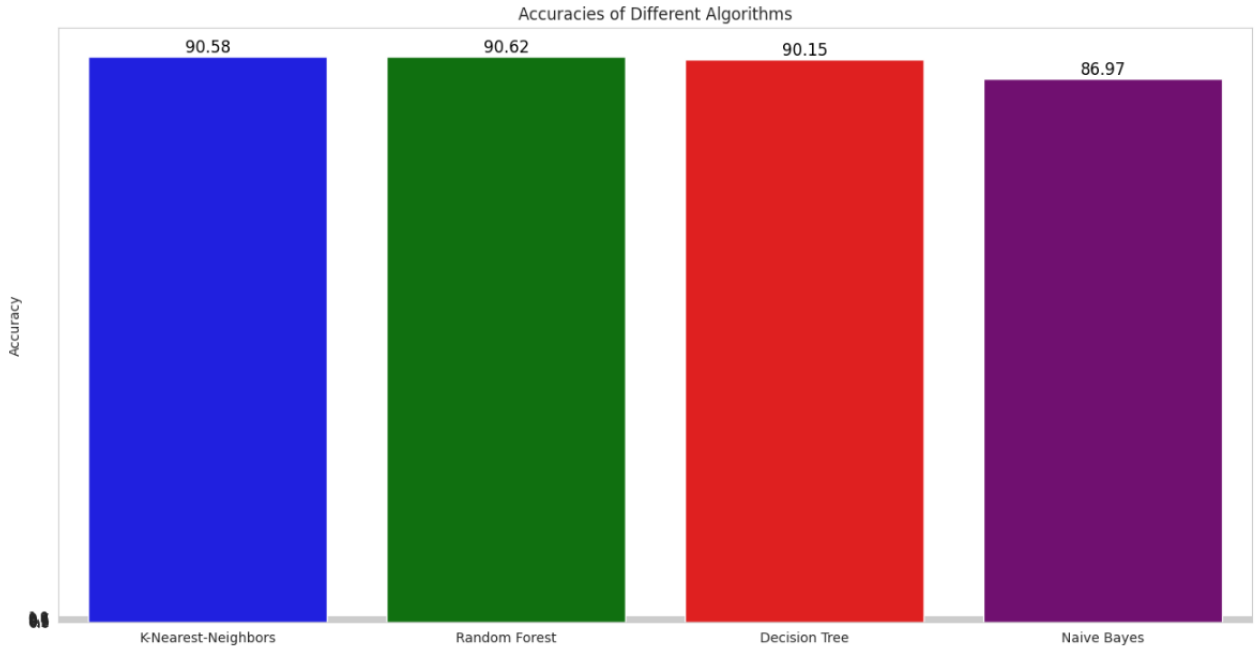
```
[ ]  #Draw a chart comparing the running time of algorithms

     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np

     # Create a list of algorithms and their running times
     algorithms = ['K-Nearest-Neighbors', 'Random Forest', 'Decision Tree', 'Naive Bayes']
     times = [times_knn, times_rf, times_dt, times_nv]

     sns.set_style('whitegrid')
     plt.figure(figsize=(16,8))
     plt.yticks(np.arange(0, max(times) + 1, 0.5))
     plt.ylabel('Time (s)')
     plt.title('Run Time of Algorithms')
     ax = sns.barplot(x=algorithms, y=times, palette='viridis')

     # Annotate each bar with the running time
     for i, time in enumerate(times):
         ax.annotate(f'{time:.2f}', xy=(i, time), ha='center', va='bottom')

     plt.show()
```



Conclusion on the runtime chart:

- The Naïve Bayes algorithm is the algorithm with the fastest running time for datasets, with only 0.039s.
- K-NN algorithm is slowest with 62.0339s.
- Draw a chart comparing the accuracy of algorithms:

```
#Draw a chart comparing the accuracy of algorithms:

# Create a list of algorithms and their accuracies
algorithms = ['K-Nearest-Neighbors', 'Random Forest', 'Decision Tree', 'Naive Bayes']
accuracies = [accuracies_max_knn, accuracy_rf, accuracy_dt, accuracy_nv]

colors = ['blue', 'green', 'red', 'purple']
sns.set_style('whitegrid')
plt.figure(figsize=(16,8))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.ylabel('Accuracy')
plt.title('Accuracies of Different Algorithms')
ax = sns.barplot(x=algorithms, y=accuracies, palette=colors)

for i, accuracy in enumerate(accuracies):
    ax.annotate(f'{accuracy:.2f}', xy=(i, accuracy), ha='center', va='bottom', fontsize=12, color='black')

plt.show()
```



Conclusion on the accuracy chart:

- The algorithms all give very high accuracy results, balanced with each other, most of which is 86~90% accuracy.
- The Naïve Bayes Neighbors algorithm has the lowest accuracy of the four algorithms, with an accuracy of 86.97%.

```
results = pd.DataFrame({
  'Model': ['K-Nearest-Neighbors', 'Random Forest', 'Decision Tree (CART)', 'Naive Bayes'],
  'Score': [ accuracies_max_knn, accuracy_rf, accuracy_dt, accuracy_nv]})
result_df = results.sort_values (by='Score', ascending=False)
result_df = result_df.set_index('Model')
result_df
```

Model	Score
Random Forest	90.62
K-Nearest-Neighbors	90.58
Decision Tree (CART)	90.15
Naive Bayes	86.97

```
[ ] keys = f1_dict, precision_dict, recall_dict
    metrics = ['F1_Score', 'Precision', 'Recall']
    data = pd.DataFrame(keys)
    data.index = metrics
    data
```

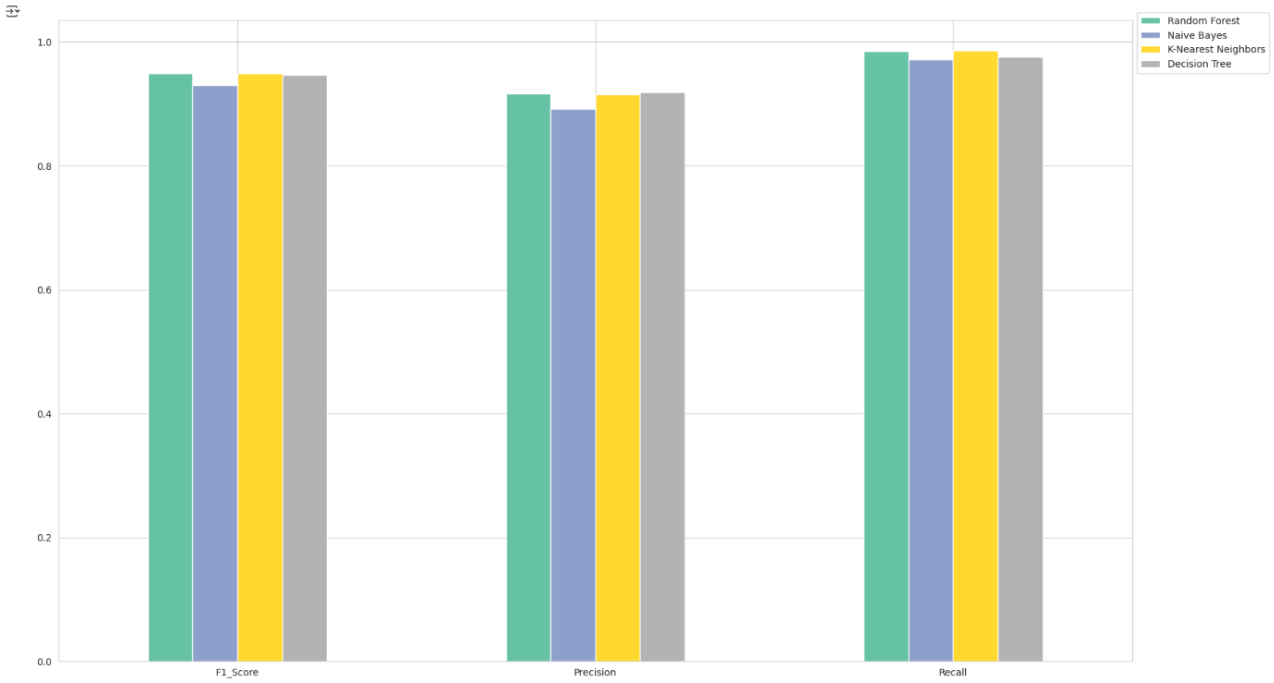
	Random Forest	Naive Bayes	K-Nearest Neighbors	Decision Tree
F1_Score	0.949134	0.929834	0.948986	0.946254
Precision	0.916392	0.891832	0.915057	0.918590
Recall	0.984303	0.971219	0.985527	0.975636

Conclusion on the F1 Score, Precision, Recall chart:

- The algorithms all give very high **F1 Score** results, balanced with each other, most of which is 92~95%.
- The algorithms all give very high **Precision** results, balanced with each other, most of which is 89~92%.
- The algorithms all give very high **Recall** results, balanced with each other, most of which is 97~99%.
- The Naïve Bayes Neighbors algorithm has the lowest **F1 Score** of the four algorithms, with percentage of 92.98%.

- The Naïve Bayes Neighbors algorithm has the lowest **Precision** of the four algorithms, with percentage of 89.18%.
- The Naïve Bayes Neighbors algorithm has the lowest **Recall** of the four algorithms, with percentage of 97.12%.

```
[ ]  result = data.plot(kind='bar', rot=0, figsize=(20, 12), cmap='Set2');

     result.legend(bbox_to_anchor=(1, 1.02), loc='upper left');
```



CHAPTER IV: CONCLUSION

4.1 Advantages and limitations of each algorithm

4.1.1 Decision Tree

❖ **Advantages**

- The algorithm is simple, intuitive, not too complicated to understand the first time.
- The training dataset doesn't have to be too large to build an analytical model.
- Some decision tree algorithms are capable of processing missing data and faulty data without applying methods such as "imputing missing values" or removing. Less affected by the exception data.
- There is no need to make initial assumptions about the laws of distribution as in statistics, and as a result the results of the analysis obtained are the most objective, "natural".
- It can help us classify data objects according to multi-layered, multi-class classifications, especially if the target variable is a complex quantitative distortion.
- Can be applied flexibly to target variables, target variables.
- Delivers highly accurate forecast results, easy to implement, fast in training, no need to switch variables.
- Easy to interpret or explain to listeners, viewers who want to understand the results of analysis but have no knowledge of data science.
- Articulate the connection between variables and data attributes in the most intuitive way.
- In addition to economics, finance, decision tree algorithms can be applied in the fields of health, agriculture, biology.

❖ **Limitations**

- The decision tree algorithm works effectively on a simple dataset that has few data variables that relate to each other, and vice versa if applied to complex datasets.

- When applied with complex datasets, many different variables and attributes can lead to overfitting patterns, which are too consistent with training data leading to the problem of not giving accurate classification results when applied to test data, and new data.
- The variance value is high, when there is a small change in the dataset can affect the structure of the model.
- The tree algorithm decides to apply only to classification trees if misclassification can lead to serious mistakes.
- The tree algorithm decides whether it is likely to be "biased" or biased if the dataset is not balanced.
- Training and testing datasets must be perfectly prepared, good quality must be balanced in layers, groups in target variables.
- There is no technical "support" or "reverse query" capability.

4.1.2 Random Forest

❖ **Advantages**

Random Forest has several advantages that contribute to its popularity and effectiveness in machine learning tasks.

Robustness: Random Forest is robust against overfitting, which occurs when a model performs well on training data but fails to generalize to new, unseen data. The algorithm's ensemble approach, combined with bootstrapping and feature randomness, helps to reduce overfitting and improve generalization performance.

Versatility: Random Forest can handle a wide range of data types, including numerical and categorical features. It can also handle missing values and outliers without requiring extensive data preprocessing.

Feature Importance: Random Forest provides a measure of feature importance, indicating the relative contribution of each feature to the model's predictions. This information can be used for feature selection, dimensionality reduction, and gaining insights into the underlying data.

Non-linearity and Interactions: Random Forest can capture non-linear relationships between features and the target variable. It can also handle interactions between features, allowing for more complex and expressive modeling capabilities.

Scalability: Random Forest can efficiently handle large datasets with a high number of features. The algorithm's parallelizability makes it suitable for parallel and distributed computing environments, enabling faster training and prediction times.

Resistance to Noise: Random Forest is less affected by noisy data compared to individual decision trees. By aggregating predictions from multiple trees, the impact of noise is mitigated, leading to more reliable results.

❖ Limitations:

Model Interpretability: Random Forest models can be less interpretable compared to individual decision trees. As the algorithm combines multiple trees, understanding the exact decision-making process can be more challenging.

Computationally Intensive: Training a Random Forest model can be computationally expensive, especially for large datasets with numerous features. The algorithm constructs multiple decision trees, which increases the overall training time and memory requirements.

Memory Usage: Random Forest models consume more memory compared to simpler algorithms. As each decision tree is stored separately, the memory footprint of the model can become substantial, particularly when dealing with a large number of trees or complex datasets.

Overfitting in Noisy Data: While Random Forest is generally robust to overfitting, it can still be affected by noise or outliers in the data. Noisy features can lead to overfitting, where individual trees capture spurious patterns in the data**.**

Biased Class Distribution: Random Forest may struggle with imbalanced class distributions, where one class is significantly more prevalent than others. The algorithm can have a bias towards the majority class, resulting in poorer performance for minority classes.

Training Time Sensitivity: Random Forest training can be sensitive to the choice of hyperparameters, such as the number of trees or the depth of each tree. Finding the optimal set of hyperparameters can require some experimentation and tuning.

4.1.3 Naïve Bayes

❖ Advantages

- Independent assumptions: works well for multiple problems/data domains and applications.
- Simple but good enough to solve many problems such as text layering, spam filtering
- Easy to use and fast when it comes to guessing the label of test data. It's pretty good in multi-class prediction (test later).
- When assuming that the features of the data are independent of each other, Naive Bayes runs better than other algorithms such as logistic regression and also needs less data.
- Allows the succession of prior knowledge and observed data.
- It is good that there is a numerical difference between the classification classes.
- Model training (parameter estimation) is easy and fast.

❖ Limitations:

- The accuracy of Naive Bayes compared to other algorithms is not high.
- In the real world, it is almost impossible when the features of test data are independent of each other.
- Problem zero (stated how to solve it above).
- The model is not trained by a strong and rigorous optimization method.
- The parameters of the model are estimates of the probability of single conditions. Do not take into account the interaction between these estimates.

4.1.4 K–Nearest Neighbors

❖ **Advantages**

Easy to implement: Given the algorithm's simplicity and accuracy, it is one of the first classifiers that a new data scientist will learn.

Adapts easily: As new training samples are added, the algorithm adjusts to account for any new data since all training data is stored into memory.

Few hyperparameters: KNN only requires a k value and a distance metric, which is low when compared to other machine learning algorithms.

❖ **Limitations**

Does not scale well: Since KNN is a lazy algorithm, it takes up more memory and data storage compared to other classifiers. This can be costly from both a time and money perspective. More memory and storage will drive up business expenses and more data can take longer to compute. While different data structures, such as Ball-Tree, have been created to address the computational inefficiencies, a different classifier may be ideal depending on the business problem.

Curse of dimensionality: The KNN algorithm tends to fall victim to the curse of dimensionality, which means that it doesn't perform well with high-dimensional data inputs. This is sometimes also referred to as the peaking phenomenon (PDF, 340 MB), where after the algorithm attains the optimal number of features, additional features increases the amount of classification errors, especially when the sample size is smaller.

Prone to overfitting: Due to the "curse of dimensionality", KNN is also more prone to overfitting. While feature selection and dimensionality reduction techniques are leveraged to prevent this from occurring, the value of k can also impact the model's behavior. Lower values of k can overfit the data, whereas higher values of k tend to "smooth out" the prediction values since it is averaging the values over a greater area, or neighborhood. However, if the value of k is too high, then it can underfit the data.

REFERENCES

1. **https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf**

2. **https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/**

3. **http://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02/The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf**

4. **https://scikit-learn.org/stable/modules/svm.html**