DAB 322

Cybercrime In India

SUBMITTED TO: -

Vicky Tao

SUBMITTED BY: -

Manish Kataria (W0865937)

Sahil (W0861930)

Janvi(W0861408)

Parbhjot Kaur (W0859954)

Sandeep Kaur (W0862377)

Table of Contents

1. Introduction

In today's modern world we develop lot of things related to internet and technology. This development of internet and technology also use in the crime world with the name "Cybercrime". First, we have question rise in our mind, "what is Cybercrime?". It is the use of a computer as an instrument to further illegal ends.[1]

If cybercrime were a country, it would have the third-largest economy in the world, behind only the U.S. and China. In 2021, cybercrime was expected to cause $6 trillion in global damages. Experts predict that cybercrime costs will rise by 15% each year, reaching $10.5 trillion annually by 2025—up from $3 trillion in 2015. This makes it the biggest transfer of wealth in history, threatens innovation and investment, causes more damage than natural disasters, and generates more money than the global illegal drug trade.[2]

Cybercrime is becoming a major global issue, and India is facing its own share of challenges. As digital services grow, cybercriminals are finding new ways to exploit individuals and businesses. This project focuses on three main types of cybercrimes in India: malware attacks, credit/debit card fraud, and online banking fraud. By analyzing datasets related to these crimes, we apply machine learning models to detect and predict fraudulent activities. Our goal is to provide insights that can help prevent, detect, and reduce cybercrime in India.

The project we discus four type of analysis:

1. **Descriptive Analysis:** To identify patterns and trends in cybercrime.
2. **Diagnostic Analysis:** To understand the causes of these crimes.
3. **Predictive Analysis:** To predict potential fraud risks.
4. **Prescriptive Analysis:** To suggest ways to prevent these crimes.

We also use machine learning models like Random Forest and Multinomial Naive Bayes to detect malware, credit card fraud, and phishing, taking a data-driven approach to improve cybersecurity in India.

2. Cybercrime

Now, we are going we know about Cybercrime thoroughly.

Again, what is Cybercrime?

The term "cybercrime" came into use with the advancement of computers and networks.

Cybercrimes are a serious threat because they can cause major problems, such as financial losses, data breaches, system failures, and damage to an organization's reputation.[3]

In our words, we like to say that “Cybercrime is the illegal activity that carry out or assist with use of technology.”

Who are The Cybercriminals?

A cybercriminal is someone who uses technology to commit harmful and illegal activities, known as cybercrimes. They can work alone or in groups.

Many cybercriminals operate on the "Dark Web," where they offer illegal services or products.

Here are some examples of cybercriminals:

- Black hat hackers
- Cyberstalkers
- Cyber terrorists
- Scammers

How do Cybercrimes happen?

Cybercriminals find weaknesses in systems and use them to break in and gain control.

These weaknesses can include weak passwords, poor security measures, or a lack of strict security rules and policies.

Why are Cybercrimes Increasing?

As technology keeps advancing, people rely on it more than ever. Most smart devices are connected to the internet, which brings both benefits and risks.

One major risk is the growing number of cybercrimes. There are not enough security measures to fully protect these technologies.

Here are some main reasons for the rise in cybercrime:

- **Weak devices:**
  Many devices do not have strong security, making them easy targets for cybercriminals.
- **Personal reasons:**
  Some cybercriminals attack others out of revenge or personal conflicts.
- **Financial gain:**
  The most common reason—many hackers commit cybercrimes to make money.

2.1 Two Main ways of Cybercrimes

- **Targeting computers:**
  This includes any method that harms computers, such as malware or denial-of-service attacks.
- **Using computers:**
  This involves using computers to carry out various illegal activities.

2.2 Other Ways of Cybercrimes

Cybercrimes can be divided into four main categories:

1. **Individual Cybercrimes:**
   These target individuals and include phishing, spoofing, spam, and cyberstalking.
2. **Organizational Cybercrimes:**
   These attack businesses or institutions, often using malware or denial-of-service attacks.
3. **Property Cybercrimes:**
   These involve crimes like credit card fraud and intellectual property theft.
4. **Society Cybercrimes:**
   The most dangerous type, including cyberterrorism.

2.3 Common Types of Cybercrimes

1. **Phishing & Scams:**
   Cybercriminals trick users with fake emails or messages to steal personal data or install harmful software.
2. **Identity Theft:**
   Criminals steal personal information, like credit card details or photos, to commit fraud.
3. **Ransomware Attacks:**
   A type of malware that locks users out of their files and demands payment to regain access.
4. **Hacking & Network Misuse:**
   Gaining unauthorized access to a computer or network to alter, steal, or damage data.
5. **Internet Fraud:**
   Any fraud that happens online, such as spam, banking frauds, and theft of online services.

More cybercrimes

- Cyber Bullying
- Cyber Stalking
- Software Piracy
- Social Media Frauds
- Online Drug Trafficking
- Electronic Money Laundering
- Cyber Extortion
- Intellectual-Property Infringements
- Online Recruitment Fraud [3]

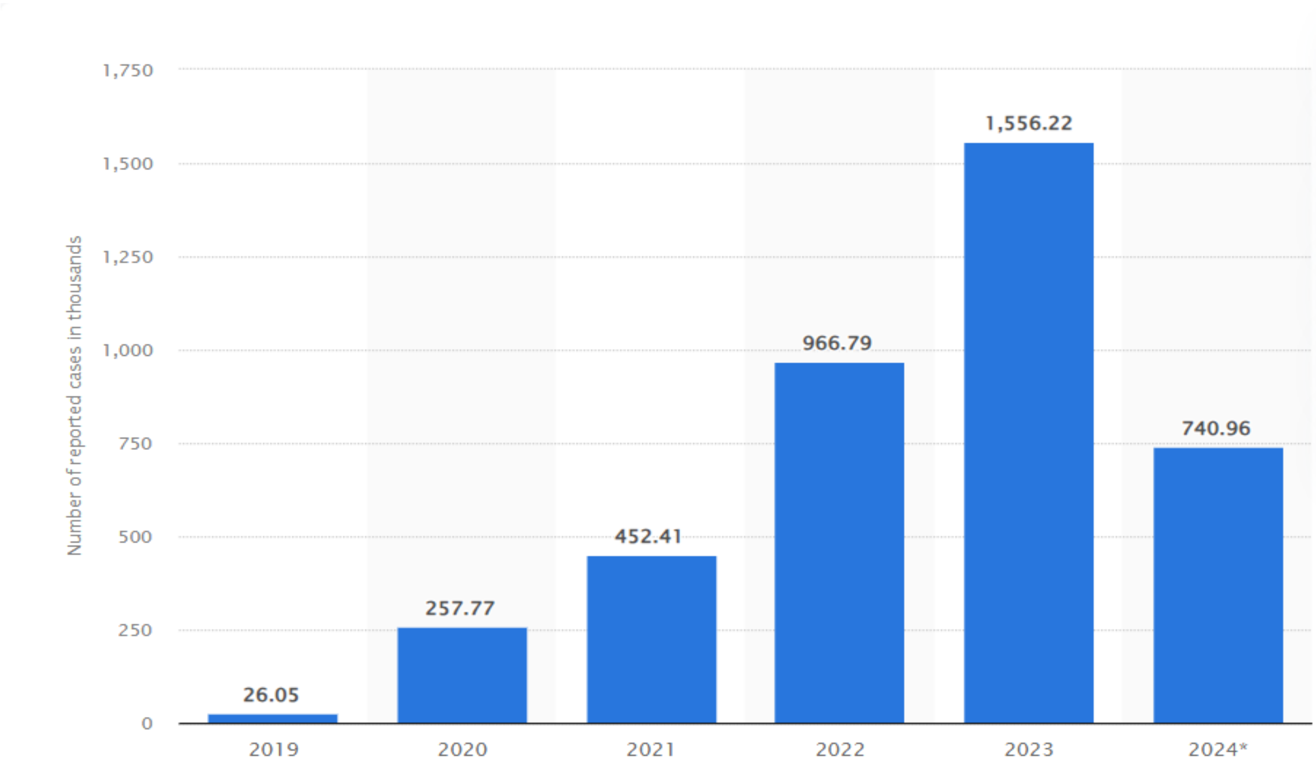
3. Problem Overview

India is on track to become one of the world's top digital hubs. While increased connectivity and a growing digital economy offer significant progress, they also expose the country to new vulnerabilities. India is just starting to build a cybersecurity framework to protect its massive online population. Meanwhile, cybercrimes are borderless and are evolving quickly alongside emerging technologies.

Cyber crime affects a lot of people and businesses each year, with the number of reported cases increasing rapidly. One major issue is cyber fraud, which leads to billions of rupees in financial losses. Industries like IT, healthcare, manufacturing, and finance are especially at risk, and small businesses are also often targeted. In 2024, only 41% of Indian companies had strong cyber security measures in place.

Though the private sector faces most of the online crime, government agencies are also victims of attacks. For example, there was a breach involving India’s Aadhaar system, which exposed personal information, including bank details, addresses, and biometrics, of over a billion people. In 2024, the cost of data breaches in India exceeded two million dollars.[4]

In the first four months of 2024, over 740,000 cyber crime cases were reported to the Indian Cyber Crime Coordination Centre (I4C). The number of cyber crimes in India increased dramatically between 2019 and 2020 and has been rising ever since. Around 85% of the reports in 2024 were about online financial fraud.[5]



Number of cyber crime cases registered by the Indian Cyber Crime Coordination Centre (I4C) in India from January 2019 to April 2024 [5]

The Indian Cyber Crime Coordination Centre (I4C) reported that in May 2024, an average of 7,000 cyber crime complaints were made every day. This was a huge increase of 113.7% compared to the years 2021 to 2023, and a 60.9% rise from 2022 to 2023, as stated in the Economic Times.

Also, 85% of these complaints were about online financial fraud.[6]

In this project we perform four types of analysis on three different topics that are “Malware Attacks”, “Credit/Debit card frauds”, “Online banking Frauds”.

4. Malware Attacks

4.1 Datasets Overview

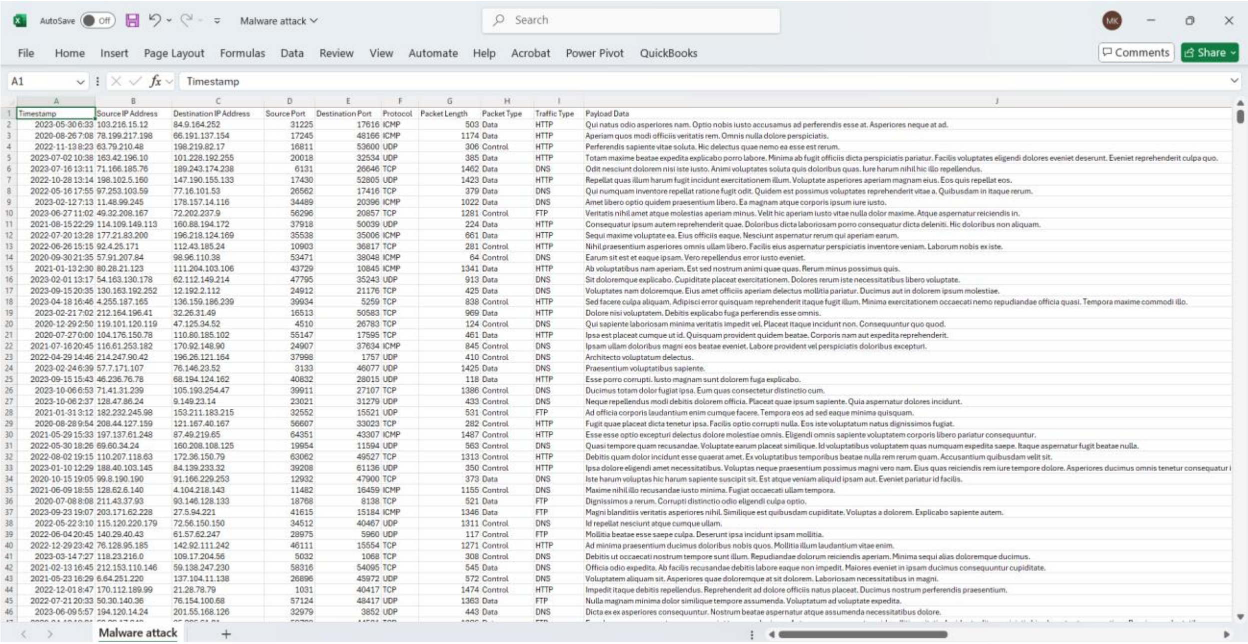
- **Source:** Kaggle Dataset[7]
- **Additional Source for History:** Cyber Security Attacks Dataset[8]
- **Historical Background:** Malware attacks have been increasing with the rise of digital platforms. Various attack types such as ransomware, trojans, and spyware have targeted users in India.
- **Machine Learning Model:** Random Forest Classifier and SVC

4.2 EDA for History of Malware attacks (Diagnostic Analysis)

When we start for history, we have twenty-five following variables:

1. Timestamp
2. Source IP Address
3. Destination IP Address
4. Source Port
5. Destination Port
6. Protocol
7. Packet Length
8. Packet Type
9. Traffic Type
10. Payload Data
11. Malware Indicators
12. Anomaly Scores
13. Alerts/Warnings
14. Attack Type
15. Attack Signature

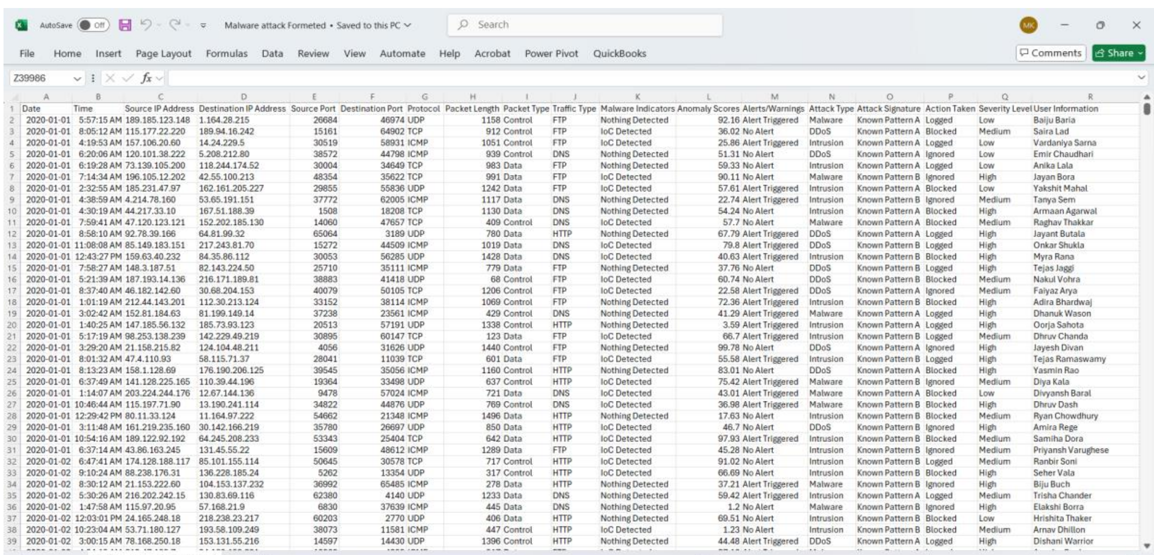
16. Action Taken
17. Severity Level
18. User Information
19. Device Information
20. Network Segment
21. Geo-location Data
22. Proxy Information
23. Firewall Logs
24. IDS/IPS Alerts
25. Log Source



We format our data and remove some variable which we have no need. After that our data have only twenty-three variables which are follow:

1. Date
2. Time
3. Source IP Address
4. Destination IP Address
5. Source Port
6. Destination Port
7. Protocol

8. Packet Length
9. Packet Type
10. Traffic Type
11. Malware Indicators
12. Anomaly Scores
13. Alerts/Warnings
14. Attack Type
15. Attack Signature
16. Action Taken
17. Severity Level
18. User Information
19. Device Information
20. Network Segment
21. City
22. State
23. Log Source



Date	Time	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Packet Type	Traffic Type	Malware Indicators	Anomaly Scores	Alerts/Warnings	Attack Type	Attack Signature	Action Taken	Severity Level	User Information
2020-01-01	5:57:15 AM	189.185.123.148	1.164.28.215	26684	46974	UDP	1158	Control	FTP	Nothing Detected	92.16	Alert Triggered	Malware	Known Pattern A	Logged	Low	Baiju Baria
2020-01-01	8:05:12 AM	115.177.22.220	189.94.16.242	15161	64902	TCP	912	Control	FTP	IoC Detected	36.02	No Alert	DDoS	Known Pattern A	Blocked	Medium	Saira Lad
2020-01-01	4:19:53 AM	157.106.20.60	14.24.229.5	30519	58931	ICMP	1051	Control	FTP	IoC Detected	25.86	Alert Triggered	Intrusion	Known Pattern A	Logged	Low	Vardaniya Sarna
2020-01-01	6:20:06 AM	120.101.38.222	5.208.212.80	38572	44798	ICMP	939	Control	DNS	Nothing Detected	51.31	No Alert	DDoS	Known Pattern A	Ignored	Low	Emir Chaudhari
2020-01-01	6:19:28 AM	73.139.105.200	118.244.174.52	30004	34649	TCP	983	Data	FTP	Nothing Detected	59.33	No Alert	Intrusion	Known Pattern A	Logged	Low	Anika Lala
2020-01-01	7:14:34 AM	196.105.12.202	42.55.100.213	48354	35622	TCP	991	Data	FTP	IoC Detected	90.11	No Alert	Malware	Known Pattern B	Ignored	High	Jayan Bora
2020-01-01	2:32:55 AM	185.231.47.97	162.161.205.227	29855	55836	UDP	1242	Data	FTP	IoC Detected	57.61	Alert Triggered	Intrusion	Known Pattern A	Blocked	Low	Yakshit Mahal
2020-01-01	4:38:59 AM	4.214.78.160	53.65.191.151	37772	62005	ICMP	1117	Data	DNS	Nothing Detected	22.74	Alert Triggered	Intrusion	Known Pattern B	Ignored	Medium	Tanya Sem
2020-01-01	4:30:19 AM	44.217.33.10	167.51.188.39	1508	18208	TCP	1130	Data	DNS	Nothing Detected	54.24	No Alert	Intrusion	Known Pattern A	Blocked	High	Armaan Agarwal
2020-01-01	7:59:41 AM	47.120.123.121	152.202.185.130	14060	47657	TCP	409	Control	DNS	IoC Detected	57.7	No Alert	Malware	Known Pattern A	Blocked	Medium	Raghav Thakkar
2020-01-01	8:58:10 AM	92.78.39.166	64.81.99.32	65064	3189	UDP	780	Data	HTTP	Nothing Detected	67.79	Alert Triggered	DDoS	Known Pattern A	Logged	High	Jayant Butala
2020-01-01	11:08:08 AM	85.149.183.151	217.243.81.70	15272	44509	ICMP	1019	Data	DNS	IoC Detected	79.8	Alert Triggered	DDoS	Known Pattern B	Logged	High	Onkar Shukla
2020-01-01	12:43:27 PM	159.63.40.232	84.35.86.112	30053	56285	UDP	1428	Data	DNS	IoC Detected	40.63	Alert Triggered	Intrusion	Known Pattern B	Blocked	High	Myra Rana
2020-01-01	7:58:27 AM	148.3.187.51	82.143.224.50	25710	35111	ICMP	779	Data	FTP	Nothing Detected	37.76	No Alert	DDoS	Known Pattern B	Logged	High	Tejas Jaggi
2020-01-01	5:21:39 AM	187.193.14.136	216.171.189.81	38883	41418	UDP	68	Control	FTP	IoC Detected	60.74	No Alert	DDoS	Known Pattern B	Blocked	Medium	Nakul Vohra
2020-01-01	8:37:40 AM	46.182.142.60	30.68.204.153	40079	50105	TCP	1206	Control	FTP	IoC Detected	22.58	Alert Triggered	DDoS	Known Pattern A	Ignored	Medium	Faiyaz Arya
2020-01-01	1:01:19 AM	212.44.143.201	112.30.213.124	33152	38114	ICMP	1069	Control	FTP	Nothing Detected	72.36	Alert Triggered	Intrusion	Known Pattern B	Blocked	High	Adira Bhardwaj
2020-01-01	3:02:42 AM	152.81.184.63	81.199.149.14	37238	23561	ICMP	429	Control	DNS	Nothing Detected	41.29	Alert Triggered	Malware	Known Pattern A	Logged	High	Dhanuk Wason
2020-01-01	1:40:25 AM	147.185.56.132	185.73.93.123	20513	57191	UDP	1338	Control	HTTP	Nothing Detected	3.59	Alert Triggered	Intrusion	Known Pattern A	Logged	High	Oorja Sahota
2020-01-01	5:17:19 AM	98.253.138.239	142.229.49.219	30895	60147	TCP	123	Data	FTP	IoC Detected	66.7	Alert Triggered	Intrusion	Known Pattern B	Logged	Medium	Dhruv Chanda
2020-01-01	3:29:20 AM	21.158.215.82	124.104.48.211	4056	31626	UDP	1440	Control	FTP	Nothing Detected	99.78	No Alert	DDoS	Known Pattern A	Ignored	High	Jayesh Divan
2020-01-01	8:01:32 AM	47.4.110.93	58.115.71.37	28041	11039	TCP	601	Data	FTP	IoC Detected	55.58	Alert Triggered	Intrusion	Known Pattern B	Logged	High	Tejas Ramaswamy
2020-01-01	8:13:23 AM	158.1.128.69	176.190.206.125	39545	35056	ICMP	1160	Control	HTTP	Nothing Detected	83.01	No Alert	DDoS	Known Pattern A	Blocked	High	Yasmin Rao
2020-01-01	6:37:49 AM	141.128.225.165	110.39.44.196	19364	33498	UDP	637	Control	HTTP	IoC Detected	75.42	Alert Triggered	Malware	Known Pattern B	Ignored	Medium	Diya Kala
2020-01-01	1:14:07 AM	203.224.244.176	12.67.144.136	9478	57024	ICMP	721	Data	DNS	IoC Detected	43.01	Alert Triggered	Malware	Known Pattern A	Blocked	Low	Divyansh Baral
2020-01-01	10:46:44 AM	115.197.71.90	13.190.241.114	34822	44876	UDP	769	Control	DNS	IoC Detected	36.98	Alert Triggered	Malware	Known Pattern B	Blocked	High	Dhruv Dash
2020-01-01	12:29:42 PM	80.11.33.124	11.164.97.222	54662	21348	ICMP	1496	Data	HTTP	Nothing Detected	17.63	No Alert	Intrusion	Known Pattern B	Blocked	Medium	Ryan Chowdhury
2020-01-01	3:11:48 AM	161.219.235.160	30.142.166.219	35780	26697	UDP	850	Data	HTTP	IoC Detected	46.7	No Alert	DDoS	Known Pattern B	Ignored	High	Amira Rege
2020-01-01	10:54:16 AM	189.122.92.192	64.245.208.233	53343	25404	TCP	642	Data	HTTP	IoC Detected	97.93	Alert Triggered	Intrusion	Known Pattern B	Blocked	Medium	Samiha Dora
2020-01-01	6:37:14 AM	43.86.163.245	131.45.55.22	15609	48612	ICMP	1289	Data	FTP	IoC Detected	45.28	No Alert	Intrusion	Known Pattern B	Ignored	Medium	Priyansh Varughese
2020-01-02	6:47:41 AM	174.128.188.117	85.101.155.114	50645	30578	TCP	717	Control	HTTP	IoC Detected	91.02	No Alert	Intrusion	Known Pattern B	Logged	Medium	Ranbir Soni
2020-01-02	9:10:24 AM	88.238.176.31	136.228.185.24	5262	13354	UDP	317	Control	HTTP	IoC Detected	66.69	No Alert	Intrusion	Known Pattern B	Blocked	High	Seher Vala
2020-01-02	8:30:12 AM	21.153.222.60	104.153.137.232	36992	65485	ICMP	278	Data	HTTP	Nothing Detected	37.21	Alert Triggered	Malware	Known Pattern B	Ignored	High	Biju Buch
2020-01-02	5:30:26 AM	216.202.242.15	130.83.69.116	62380	4140	UDP	1233	Data	DNS	Nothing Detected	59.42	Alert Triggered	Intrusion	Known Pattern A	Logged	Medium	Trisha Chander
2020-01-02	1:47:58 AM	115.97.20.95	57.168.21.9	6830	37639	ICMP	445	Data	DNS	Nothing Detected	1.2	No Alert	Malware	Known Pattern B	Ignored	High	Elakshi Borra
2020-01-02	12:03:01 PM	24.165.248.18	218.238.23.217	60203	2770	UDP	406	Data	HTTP	Nothing Detected	69.51	No Alert	Intrusion	Known Pattern B	Blocked	Low	Hrishita Thaker
2020-01-02	10:23:04 AM	53.71.180.127	193.58.109.249	38073	11581	ICMP	447	Control	HTTP	IoC Detected	1.23	No Alert	Intrusion	Known Pattern B	Blocked	Medium	Arnav Dhillon
2020-01-02	3:00:15 AM	78.168.250.18	153.131.55.216	14597	14430	UDP	1396	Control	HTTP	Nothing Detected	44.48	Alert Triggered	DDoS	Known Pattern A	Logged	High	Dishani Warrior

After formatting our data, we have some empty values we fill that values with opposite of there correspond. Like in column Malware Indicators we have values “IoC Detected” in this column for empty values we fill “Nothing Detected”. For “Alert Triggered”, “No Alert”.
After this all we done our formatting and handle empty values.

Now we are going to work on R studios to find structure of our data we call the dataset and run str() command to find the structure.

````
```{r}
malware_data <- read.csv("C:/Users/manis/Desktop/CAPSTONE/Main/Ransomware Detection/History/Malware attack Formeted.csv",  header = TRUE)
```

```{r}
str(malware_data)
```

'data.frame':   40569 obs. of  24 variables:
 $ Date                  : chr  "2020-01-01" "2020-01-01" "2020-01-01" "2020-01-01" ...
 $ Time                  : chr  "5:57:15 AM" "8:05:12 AM" "4:19:53 AM" "6:20:06 AM" ...
 $ Source.IP.Address     : chr  "189.185.123.148" "115.177.22.220" "157.106.20.60" "120.101.38.222" ...
 $ Destination.IP.Address: chr  "1.164.28.215" "189.94.16.242" "14.24.229.5" "5.208.212.80" ...
 $ Source.Port           : int  26684 15161 30519 38572 30004 48354 29855 37772 1508 14060 ...
 $ Destination.Port      : int  46974 64902 58931 44798 34649 35622 55836 62005 18208 47657 ...
 $ Protocol              : chr  "UDP" "TCP" "ICMP" "ICMP" ...
 $ Packet.Length         : int  1158 912 1051 939 983 991 1242 1117 1130 409 ...
 $ Packet.Type           : chr  "Control" "Control" "Control" "Control" ...
 $ Traffic.Type          : chr  "FTP" "FTP" "FTP" "DNS" ...
 $ Malware.Indicators    : chr  "Nothing Detected" "IoC Detected" "IoC Detected" "Nothing Detected" ...
 $ Anomaly.Scores        : num  92.2 36 25.9 51.3 59.3 ...
 $ Alerts.Warnings       : chr  "Alert Triggered" "No Alert" "Alert Triggered" "No Alert" ...
 $ Attack.Type           : chr  "Malware" "DDoS" "Intrusion" "DDoS" ...
 $ Attack.Signature      : chr  "Known Pattern A" "Known Pattern A" "Known Pattern A" "Known Pattern A" ...
 $ Action.Taken          : chr  "Logged" "Blocked" "Logged" "Ignored" ...
 $ Severity.Level        : chr  "Low" "Medium" "Low" "Low" ...
 $ User.Information      : chr  "Baiju Baria" "Saira Lad" "Vardaniya Sarna" "Emir Chaudhari" ...
 $ Device.Information    : chr  "Mozilla/5.0 (iPod; U; CPU iPhone OS 3_1 like Mac OS X; az-AZ) AppleWebKit/533.34.7 (KHTML, like Gecko) Version/"| __truncated__ "Mozilla/5.0 (iPod; U; CPU iPhone OS 3_0 like Mac OS X; af-ZA) AppleWebKit/532.10.1 (KHTML, like Gecko) Version/"| __truncated__ "Opera/8.46.(Windows NT 5.1; wae-CH) Presto/2.9.169 Version/11.00" "Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)" ...
 $ Network.Segment       : chr  "Segment C" "Segment B" "Segment A" "Segment C" ...
 $ City                  : chr  "Bhilwara" "Narasaraopet" "Ulhasnagar" "Muzaffarnagar" ...
 $ State                 : chr  " Bihar" " Bihar" " Chhattisgarh" " Tripura" ...
 $ Country               : chr  "India" "India" "India" "India" ...
 $ Log.Source            : chr  "Server" "Server" "Server" "Firewall" ...
````

The variables in the data have different types, such as character, integer, and numeric. This is important because it helps us analyze and process the data correctly.

Preview of Data:

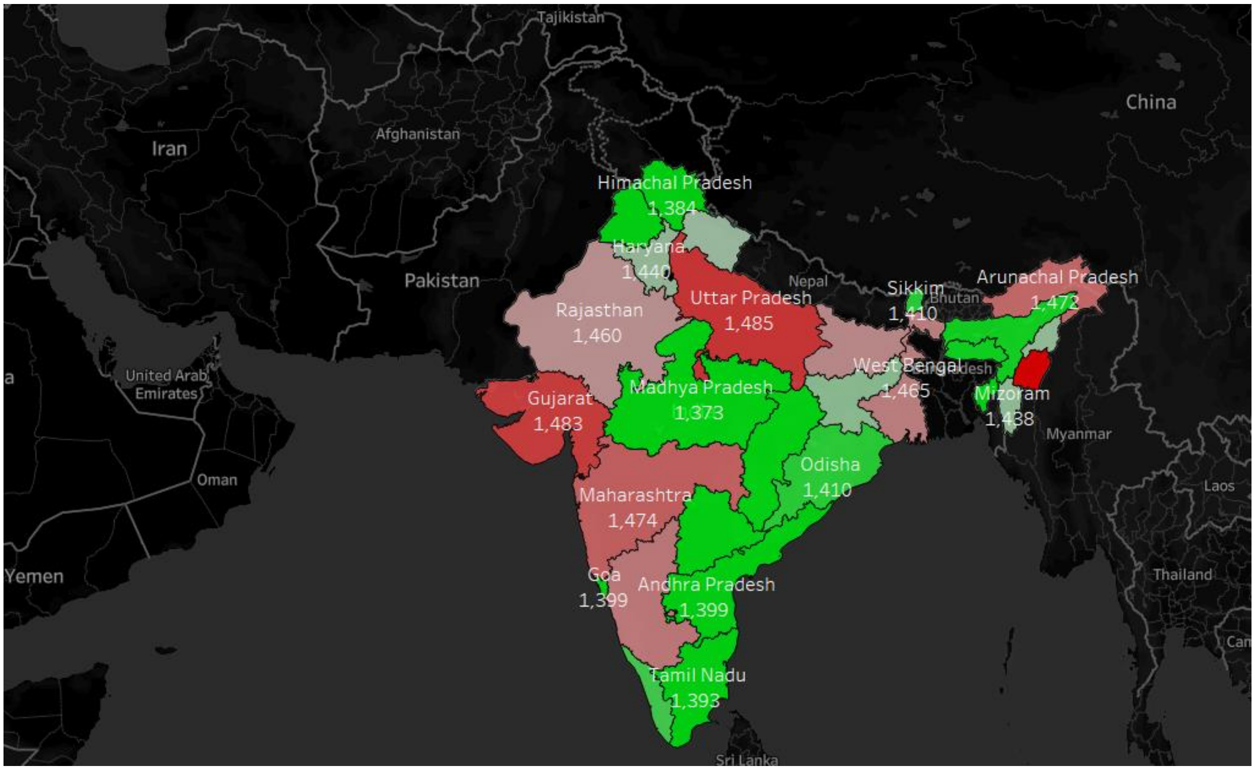
Now, we can see a preview of the first few values in each column, which gives us an idea of what the data looks like and its format.

Moreover, this data likely represents network traffic or security logs. We can use it to detect malware attacks, find anomalies, and identify other security incidents.

4.3 What happened in past understand by visualization

Now we are gone create a graphical representation of our data. Which tells us which state of India have more these attacks in history

4.3.1 Fill Map



Number of Cases of Malware attacks in Different state of India

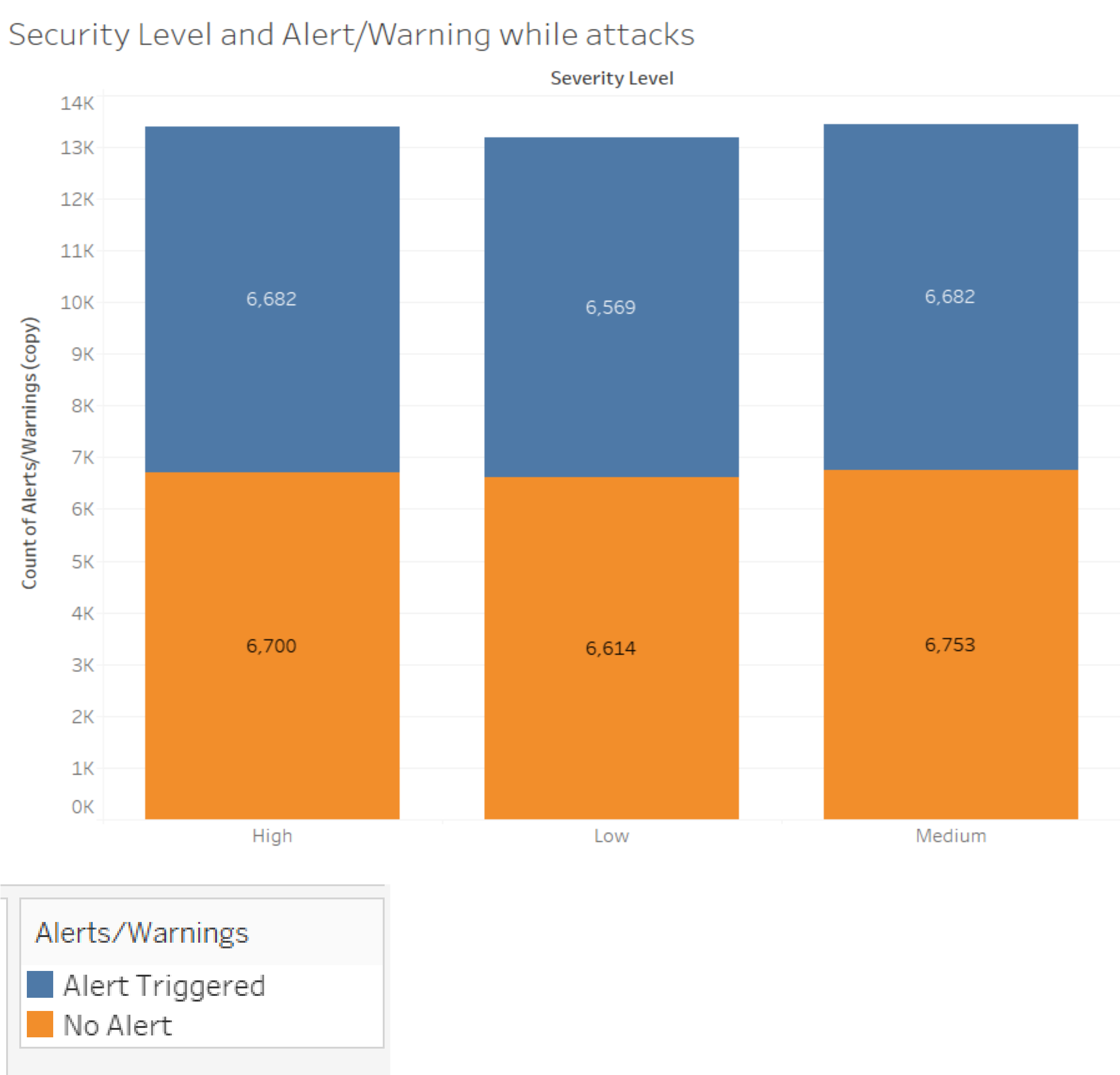
Red States (Higher Values):

- **Uttar Pradesh** appears to be the darkest red, suggesting it has the **highest value** of malware attacks.
- Other states with a distinctly red hue include **Bihar, Jharkhand, Rajasthan, Madhya Pradesh, and Chhattisgarh**.

Green States (Lower Values):

- **Gujarat** is the most prominently green state, indicating it likely has the **lowest** malware attacks.
- Other states with a greenish tint include **Punjab, Maharashtra, Goa, Kerala, Tamil Nadu, and Andhra Pradesh**.

4.3.2 Staked Bar chart



Here, we have a stacked bar chart showing security levels and the number of alerts or warnings during attacks.

Title: *"Security Level and Alert/Warning while Attacks"* – Clearly tells us what the chart is about.

Subtitle: *"Severity Level"* – Highlights that we are analyzing different levels of severity.

Y-axis: *"Count of Alerts/Warnings"* – Shows the number of security alerts or warnings, ranging from 0 to 14,000.

X-axis: *"High", "Low", "Medium"* – Represents three severity levels of attacks.

Bars: Each bar represents a severity level and is divided into two sections (stacked), i.e., different categories within each severity level.

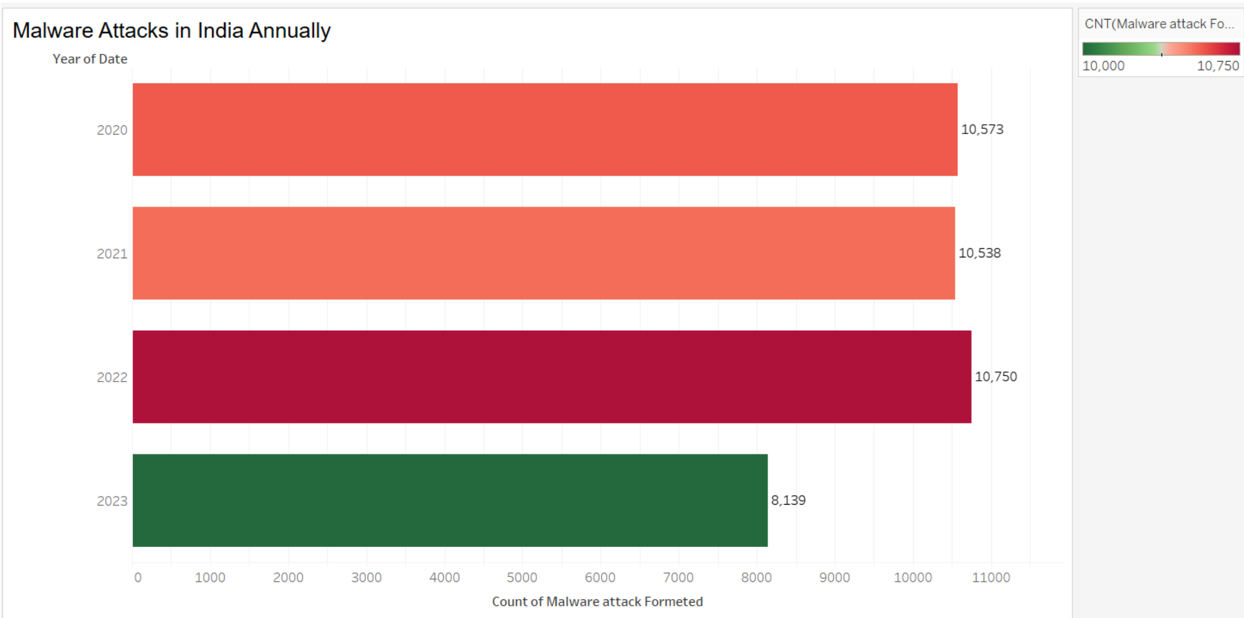
- **Orange Section (Lower):** Represents the total "Count of Alerts/Warnings" for the "Security Level" being analyzed. The exact meaning of "Security Level" refers to the base level of security monitoring or detection.
- **Blue Section (Upper):** Represents the "Count of Alerts/Warnings" specifically linked to attacks.
- **Data Labels:** Each section has a number label showing the exact count of alerts/warnings for that category.

Now, let us analyze the data:

- **High Severity:** 6,700 no alerts/warnings and 6,682 alerts/warnings related to attacks.
- **Low Severity:** 6,614 no alerts/warnings and 6,569 alerts/warnings related to attacks.
- **Medium Severity:** 6,753 no alerts/warnings and 6,682 alerts/warnings related to attacks.

Moreover, this breakdown helps us compare the security monitoring data with attack-related alerts across different severity levels.

4.3.3 Bar chart



Here, we have a line graph showing the yearly number of malware attacks in India from 2020 to 2023.

- **Title:** *"Malware Attacks in India Annually"* – Clearly tells us what the chart is about.
- **X-axis:** *"Count of Malware Attacks Formatted"* – Represents the number of malware attacks, ranging from 0 to 11,000. *"Formatted"* likely means *"Formatted"*, meaning the data has been processed or organized.
- **Y-axis:** *"Year of Date"* – Represents the years: 2020, 2021, 2022, and 2023.
- **Data Points:** Each point on the line shows the number of malware attacks in a specific year, with the exact count labeled above.
- **Line:** Connects the data points, helping us see the trend of malware attacks over the years.

Now, let's analyze the data:

- **2020:** 10,573 malware attacks.

- **2021:** 10,538 malware attacks.
- **2022:** 10,750 malware attacks.
- **2023:** 8,139 malware attacks.

Key Observations:

- **Stable Trend (2020-2022):** The number of malware attacks stayed almost the same from 2020 to 2022, fluctuating slightly between 10,500 and 10,750.
- **Big Drop in 2023:** In 2023, we see a major decline in malware attacks, dropping to 8,139.

Moreover, this trend helps us understand how malware attacks have changed over time and highlights the significant decrease in 2023.

4.4 Some Famous and Major Malware Attacks

WannaCry (2017):
In May 2017, the WannaCry ransomware attack hit many organizations worldwide, including some in India. The Maharashtra Police Department was also affected, as some of its computer systems got infected. Authorities quickly isolated the affected systems to stop the virus from spreading further.[9]

LockBit (2023):
In June 2023, the LockBit ransomware group, linked to Russia, carried out a cyberattack on Granules India, a major pharmaceutical company. The hackers claimed they had stolen sensitive data and even listed Granules India on their dark web leak site. The company confirmed the attack and took steps to secure its IT systems.[10]

Akira Ransomware (2023):
Akira ransomware, discovered in early 2023, now targets both Windows and Linux systems. The attackers use a double-extortion method, i.e., they encrypt

victims' files and also threaten to leak stolen data unless a ransom is paid. While we do not have details of specific Indian companies affected, Akira has already attacked over 250 organizations worldwide, including some in India.[11]

4.5 Common Reasons of Malware Attacks (Descriptive Analysis)

Growing Digital Infrastructure
India is rapidly going digital, with more people using the internet and technology. While this brings many benefits, it also gives cybercriminals more chances to attack. As we move toward digitization, hackers now find more ways to target weak systems, especially those that are less secure.[12]

Weak Cybersecurity Practices
Many Indian businesses, especially small and medium-sized enterprises (SMEs), do not have strong cybersecurity protections. Around 60% of Indian companies fall below the cybersecurity poverty line, i.e., they lack proper defenses against cyber threats. This makes them easy targets for hackers.[13]

Phishing & Social Engineering
Cybercriminals often trick people into sharing private information or downloading harmful software using phishing and social engineering tactics. Now, more than 70% of data breaches start this way. These methods work because they take advantage of human psychology, making them highly effective.[14]

Nation-State Attacks
Rising geopolitical tensions have also led to a sharp increase in cyberattacks backed by foreign governments. From 2021 to September 2023, these attacks grew by 278%, mainly targeting government agencies and startups. Here, the goal is to steal information or disrupt important infrastructure.[15]

4.6 Future Prediction (Predictive Analysis)

India is now a major target for malware attacks, accounting for 28% of global mobile malware incidents—more than the U.S. and Canada.[16]

Predictive Insights:

- **AI-Powered Threats:** Cybercriminals now use AI to create advanced malware that can bypass traditional security systems.
- **Machine Learning Models:** By analyzing patterns, machine learning can improve malware detection and predict attack methods in advance.

4.7 Malware Detection

Now, Let's move to detection of these attacks. We choose another data set with name "Cyber Security Attacks Dataset"[7]. Here we have some information of data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 34 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   millisecond         100000 non-null  int64
 1   classification      100000 non-null  object
 2   state               100000 non-null  int64
 3   usage_counter       100000 non-null  int64
 4   prio                100000 non-null  int64
 5   static_prio         100000 non-null  int64
 6   normal_prio         100000 non-null  int64
 7   policy              100000 non-null  int64
 8   vm_pgoff            100000 non-null  int64
 9   vm_truncate_count   100000 non-null  int64
 10  task_size           100000 non-null  int64
 11  cached_hole_size    100000 non-null  int64
 12  free_area_cache     100000 non-null  int64
 13  mm_users            100000 non-null  int64
 14  map_count           100000 non-null  int64
 15  hiwater_rss         100000 non-null  int64
 16  total_vm            100000 non-null  int64
 17  shared_vm           100000 non-null  int64
 18  exec_vm             100000 non-null  int64
 19  reserved_vm         100000 non-null  int64
 20  nr_ptes             100000 non-null  int64
 21  end_data            100000 non-null  int64
 22  last_interval       100000 non-null  int64
 23  nvcsw               100000 non-null  int64
 24  nivcsw              100000 non-null  int64
 25  min_flt             100000 non-null  int64
 26  maj_flt             100000 non-null  int64
 27  fs_excl_counter     100000 non-null  int64
 28  lock                100000 non-null  int64
 29  utime               100000 non-null  int64
 30  stime               100000 non-null  int64
 31  gtime               100000 non-null  int64
 32  cgtime              100000 non-null  int64
 33  signal_nvcsw        100000 non-null  int64
dtypes: int64(33), object(1)
memory usage: 25.9+ MB
None
```

Here, we have an image showing the output of the info () method from the pandas library in Python. This method gives us a quick summary of a **Data Frame**, which is a key structure in pandas for handling tabular data. Now, let's break down what we see:

- **100000 entries, 0 to 99999**: Here, we see that the Data Frame has **100,000 rows**, with an index ranging from **0 to 99,999**.
- **Data columns (total thirty-four columns):** This confirms that the Data Frame has **thirty-four columns** in total.

Column Details:

The table in the output shows:

- **Column number**: A sequential number for each column.
- **Column name**: The name of the column. Here, some names look split across two lines, which might mean multi-level column names or a display issue.
- **Non-Null Count**: This tells us how many values in each column are **not missing**. Here, all columns have **100,000 non-null values**, i.e., there are **no missing values**.
- **Data Type**: The type of data stored in each column:
    - **int64**: 64-bit integers (most columns).
    - **object**: A general type used for strings or mixed data (e.g., classification column).

Here, we have a **pandas Data Frame with 100,000 rows and thirty-four columns**. Most columns contain numbers, except for **one column (classification)**, which contains text. Also, there are **no missing values**.

Moving Further, Lets perform some machine learning model on given dataset.

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```python
[4] # Load the dataset
    df = pd.read_csv("Malware dataset.csv")  # Update with the correct file path
```

```python
[5] # Display basic info
    print(df.head())
```

```
   millisecond classification  state  usage_counter        prio  static_prio  \
0            0        malware      0              0  3069378560        14274
1            1        malware      0              0  3069378560        14274
2            2        malware      0              0  3069378560        14274
3            3        malware      0              0  3069378560        14274
4            4        malware      0              0  3069378560        14274

   normal_prio  policy  vm_pgoff  vm_truncate_count  ...  nivcsw  min_flt  \
0            0       0         0              13173  ...       0        0
1            0       0         0              13173  ...       0        0
2            0       0         0              13173  ...       0        0
3            0       0         0              13173  ...       0        0
4            0       0         0              13173  ...       0        0

   maj_flt  fs_excl_counter        lock   utime  stime  gtime  cgtime  \
0      120                0  3204448256  380690      4      0       0
1      120                0  3204448256  380690      4      0       0
2      120                0  3204448256  380690      4      0       0
3      120                0  3204448256  380690      4      0       0
4      120                0  3204448256  380690      4      0       0

   signal_nvcsw
0             0
1             0
2             0
3             0
4             0

[5 rows x 34 columns]
```

Here, we are importing essential libraries for **data analysis** and **machine learning**:

- **pandas**: Helps us manipulate and analyze data.
- **NumPy**: Supports numerical operations and arrays.
- **matplotlib.pyplot**: Allows us to create different types of visualizations.
- **seaborn**: Provides statistical data visualization with enhanced plots.
- **train_test_split**: Splits data into **training** and **testing** sets.
- **RandomForestClassifier**: Builds a **Random Forest** model for classification.
- **SVC**: Creates a **Support Vector Machine (SVM)** model.
- **accuracy_score, classification_report, confusion_matrix**: Help evaluate **model performance**.

Then, we **load** the dataset from a **CSV file** into a **pandas Data Frame** called df. Now, we also display the **first five rows**, i.e., we can quickly check the structure and content of the data.

```
[9]  # Assuming 'classification' is the target column (adjust if the column is different)
     X = df.drop(columns=['classification'])  # Features (all columns except the target)
     y = df['classification']  # Target (classification column)
```

```
[10] # Check target value distribution
     print(y.value_counts())

     # Split data into training and testing sets (80% train, 20% test)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     print(f"Training set: {X_train.shape}, Testing set: {X_test.shape}")
```

```
classification
malware    50000
benign     50000
Name: count, dtype: int64
Training set: (80000, 33), Testing set: (20000, 33)
```

```
[11] # Initialize Random Forest Model
     rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

     # Train the model
     rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Double-click (or enter) to edit

```
[12] # Predictions
     y_pred = rf_model.predict(X_test)
     print("Predictions:")
     print(y_pred)
```

```
Predictions:
['malware' 'malware' 'benign' ... 'benign' 'benign' 'benign']
```

```
[13] # Evaluate Model
     accuracy_rf = accuracy_score(y_test, y_pred)
     print(f"Model Accuracy: {accuracy_rf:.4f}")
```

```
Model Accuracy: 1.0000
```

Here, we see the next steps in a **Python machine learning workflow**, building on **data loading** and **initial inspection**. Now, let's break it down step by step:

Defining Features (X) and Target (y)

- We **separate the features (X) and the target (y)** for training the model.
- X = df.drop(columns=['classification']): Features (all columns **except** the target).
- y = df['classification']: Target column (i.e., what we want to predict).
- Here, we **assume** 'classification' is the target column, but we need to adjust if it's different.

Checking Target Distribution & Splitting Data

- print(y.value_counts()) : Shows how many times each class (e.g., malware, benign) appears.
- train_test_split(X, y, test_size=0.2, random_state=42): Splits the data:

    - **80% for training, 20% for testing**.
    - random_state=42 ensures we get **consistent results** every time we run the code.
- print(f"Training set: {X_train.shape}, Testing set: {X_test.shape}"): Displays the number of rows and columns in the training and testing sets.

Initializing and Training the Random Forest Model

- rf_model = RandomForestClassifier(n_estimators=100, random_state=42): Creates a **Random Forest** model with **100 decision trees**.
- rf_model.fit(X_train, y_train): Trains the model on the training data.

Making Predictions

- y_pred = rf_model.predict(X_test): The model **predicts labels** for the test set.
- print(y_pred): Displays predicted class labels (e.g., 'malware', 'benign').

Evaluating the Model

- accuracy_rf = accuracy_score(y_test, y_pred): Calculates **model accuracy**.
- print(f"Model Accuracy: {accuracy_rf:.4f}"): Displays accuracy, formatted to **four decimal places**.

```
[14] # Classification Report
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

      benign       1.00      1.00      1.00     10030
     malware       1.00      1.00      1.00      9970

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000
```

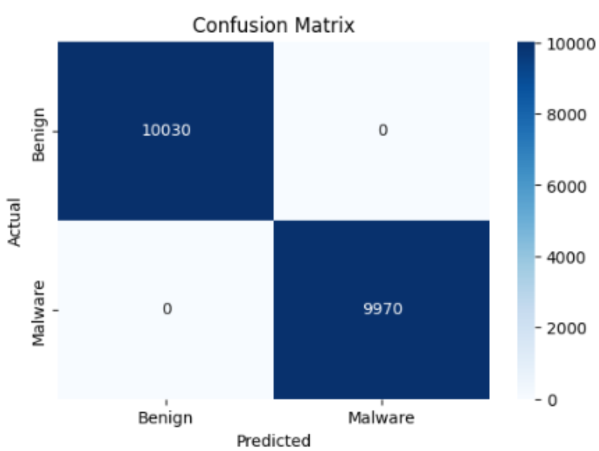
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[10030     0]
 [    0  9970]]
```

Here, we see that both the **classification report** and **confusion matrix** confirm the model's **perfect performance** on the test set.

- **Precision, Recall, and F1-score** are all **1.00** for both **benign** and **malware** classes. This means the model **made no mistakes** in classification.
- The **confusion matrix** has **zeros** in the **off-diagonal elements**, i.e., there are **no false positives** and **no false negatives**.
    - **All benign samples** were correctly classified as **benign**.
    - **All malware samples** were correctly classified as **malware**.

```python
# Visualizing the Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malware'], yticklabels=['Benign', 'Malware'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



Here, we see a **confusion matrix plot**, which helps us understand the model's performance.

Key Elements of the Plot

- **Title:** "Confusion Matrix": Clearly shows what the plot represents.
- **Axes Labels:**
    - **X-axis (Predicted):** Represents the model's predictions (**Benign, Malware**).

- **Y-axis (Actual):** Represents the true labels of the data (**Benign, Malware**).

Understanding the Heatmap Cells

The confusion matrix has **four cells**, each showing classification results:

- **Top-left (Dark Blue, "10030")**: **True Negatives** (Benign correctly predicted as Benign).
- **Top-right (Light Blue, "0")**: **False Positives** (Malware wrongly predicted as Benign: **None here**).
- **Bottom-left (Light Blue, "0")**: **False Negatives** (Benign wrongly predicted as Malware: **None here**).
- **Bottom-right (Dark Blue, "9970")**: **True Positives** (Malware correctly predicted as Malware).

Color Bar Interpretation

- Darker blue: **Higher count of correct classifications**.
- Lighter blue: **Lower count (here, zero for misclassifications)**.

```
[26] from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)

     # Initialize and train SVM model
     svm_model = SVC(kernel='linear', random_state=42)
     svm_model.fit(X_train, y_train)
```

```
SVC
SVC(kernel='linear', random_state=42)
ndarray: y_pred_svm
ndarray with shape (20000,)
```

```
y_pred_svm = svm_model.predict(X_test)
print("SVM Predictions:")
print(y_pred_svm)
```

```
SVM Predictions:
['malware' 'malware' 'benign' ... 'benign' 'benign' 'benign']
```

```
[28] # Evaluate accuracy
     accuracy_svm = accuracy_score(y_test, y_pred_svm)
     print(f"SVM Accuracy: {accuracy_svm:.4f}")
```

```
SVM Accuracy: 0.9478
```

Here, we now see the next steps in the machine learning workflow. We first scale the features using StandardScaler, i.e., we adjust the data to have zero mean and unit variance. This is important for SVM, a distance-based algorithm. Moreover, we then initialize an SVM model with a linear kernel and train it using the scaled training data.

Feature Scaling and SVM Initialization & Training:

- We import StandardScaler and create an instance called scaler.
- We then scale the training data with scaler.fit_transform(X_train) and use the same scaler to transform the test data with scaler.transform(X_test).
- Now, we initialize the SVM model using a linear kernel and set a random state for reproducibility.
- We train the model with the scaled training data.

Making Predictions:

- Next, we use the trained SVM model to predict the labels of the scaled test data.
- We print the predictions to see the output, which shows labels like 'malware' and 'benign'.

Evaluating the SVM Model's Accuracy:

- Finally, we calculate the model's accuracy by comparing the predicted labels with the true labels.
- The accuracy, printed as approximately 94.78%, shows that the SVM performed well.

```python
# Print classification report
print("SVM Classification Report:\n", classification_report(y_test, y_pred_svm))

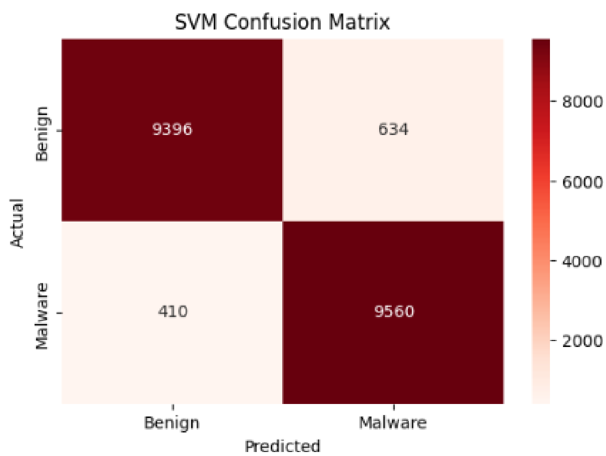
# Confusion matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Reds", xticklabels=["Benign", "Malware"], yticklabels=["Benign", "Malware"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("SVM Confusion Matrix")
plt.show()
```

```
SVM Classification Report:
               precision    recall  f1-score   support

      benign       0.96      0.94      0.95     10030
     malware       0.94      0.96      0.95      9970

    accuracy                           0.95     20000
   macro avg       0.95      0.95      0.95     20000
weighted avg       0.95      0.95      0.95     20000
```



Here, we now see the final steps in evaluating the SVM model, i.e., generating a classification report and visualizing the confusion matrix.

This report shows us detailed metrics for each class. For example, we see that for benign samples, precision is 0.96 (meaning 96% of the predictions for benign are correct), and recall is 0.94 (i.e., 94% of the actual benign samples are correctly identified). For malware, precision is 0.94 and recall is 0.96. Moreover, both classes have an F1-score of 0.95, and the overall accuracy is around 95%.

Here, we also label the axes:

- The **x-axis (Predicted)** shows the predicted labels.
- The **y-axis (Actual)** shows the true labels.

The plot shows a 2x2 matrix:

- **Top-left (dark red, "9396")**: True Negatives, i.e., benign samples correctly predicted as benign.
- **Top-right (lighter red, "634")**: False Positives, i.e., malware samples incorrectly predicted as benign.
- **Bottom-left (lighter red, "410")**: False Negatives, i.e., benign samples incorrectly predicted as malware.
- **Bottom-right (dark red, "9560")**: True Positives, i.e., malware samples correctly predicted as malware.

Moreover, the color intensity shows the number of samples in each cell, with darker red indicating higher counts.

In summary, we now have a detailed evaluation of the SVM model:

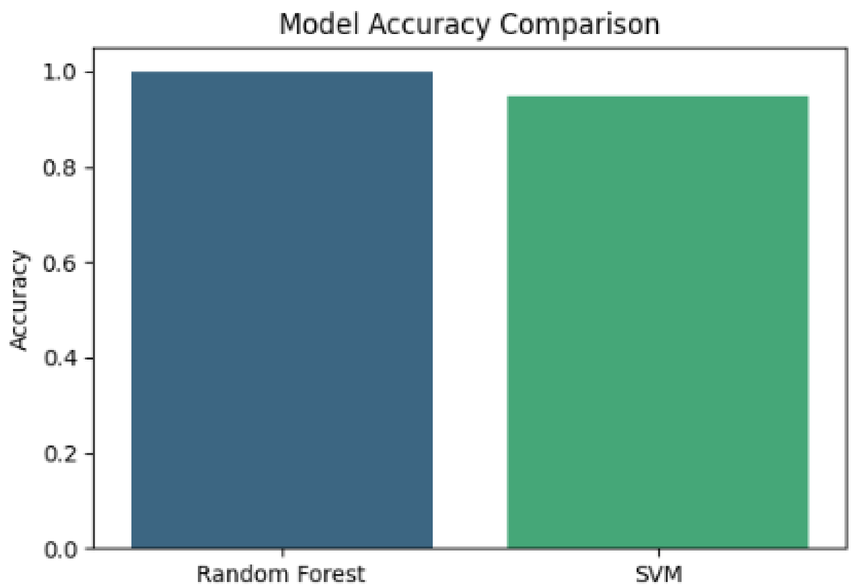
- The classification report gives us a balanced view of precision, recall, and F1-score for both classes.
- The confusion matrix visually summarizes the model's performance, showing that, unlike the perfect performance of the Random Forest model, the SVM made some mistakes (false positives and false negatives).

```
[30] print(f"Random Forest Accuracy: {accuracy_rf:.4f}")
     print(f"SVM Accuracy: {accuracy_svm:.4f}")
```

```
Random Forest Accuracy: 1.0000
SVM Accuracy: 0.9478
```

```
[31] # Bar chart comparison
     models = ["Random Forest", "SVM"]
     accuracies = [accuracy_rf, accuracy_svm]

     plt.figure(figsize=(6,4))
     sns.barplot(x=models, y=accuracies, palette="viridis")
     plt.title("Model Accuracy Comparison")
     plt.ylabel("Accuracy")
     plt.show()
```



Here, we compare the accuracy of the Random Forest and SVM models using a bar chart. Now, we first print the accuracies with:

This shows that the Random Forest achieved 1.0000 (100%) accuracy, while the SVM achieved 0.9478 (94.78%). Moreover, we create a bar chart to visualize this comparison:

Here, the x-axis displays the model names, and the y-axis shows the accuracy scores, ranging from 0.0 to 1.0. The bar for Random Forest is dark blue green, reaching up to 1.0, i.e. 100% accuracy, while the bar for SVM is lighter green, reaching approximately 0.95 (or 94.78% accuracy). This final step clearly compares the performance of the two models.

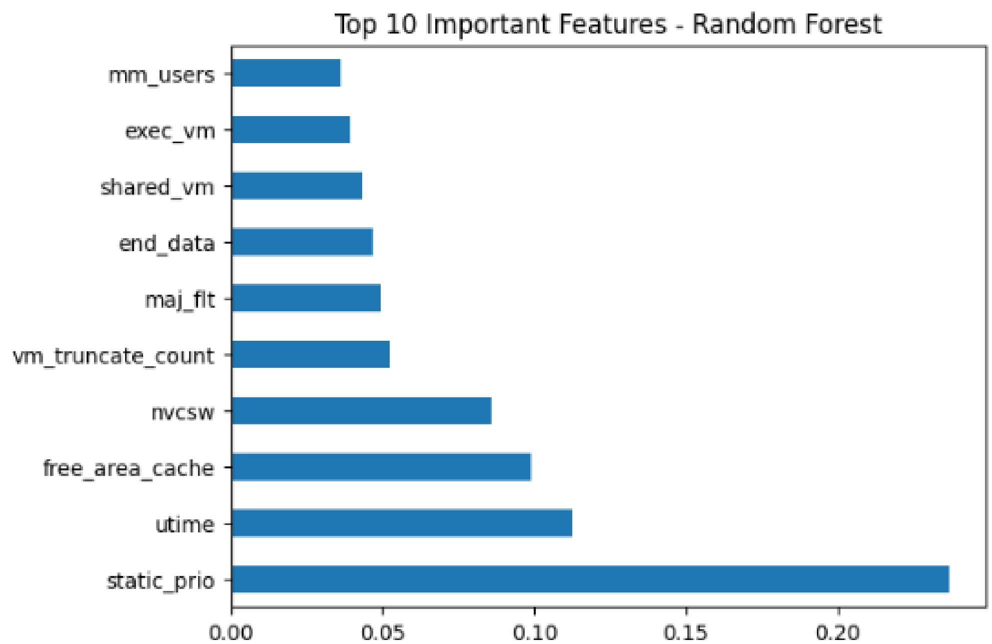
Here, we compare the accuracy of the Random Forest and SVM models using a bar chart. Now, we first print the accuracies with:

This shows that the Random Forest achieved 1.0000 (100%) accuracy, while the SVM achieved 0.9478 (94.78%). Moreover, we create a bar chart to visualize this comparison:

Here, the x-axis displays the model names, and the y-axis shows the accuracy scores, ranging from 0.0 to 1.0. The bar for Random Forest is dark blue green, reaching up to 1.0, i.e. 100% accuracy, while the bar for SVM is lighter green,

reaching approximately 0.95 (or 94.78% accuracy). This final step clearly compares the performance of the two models.

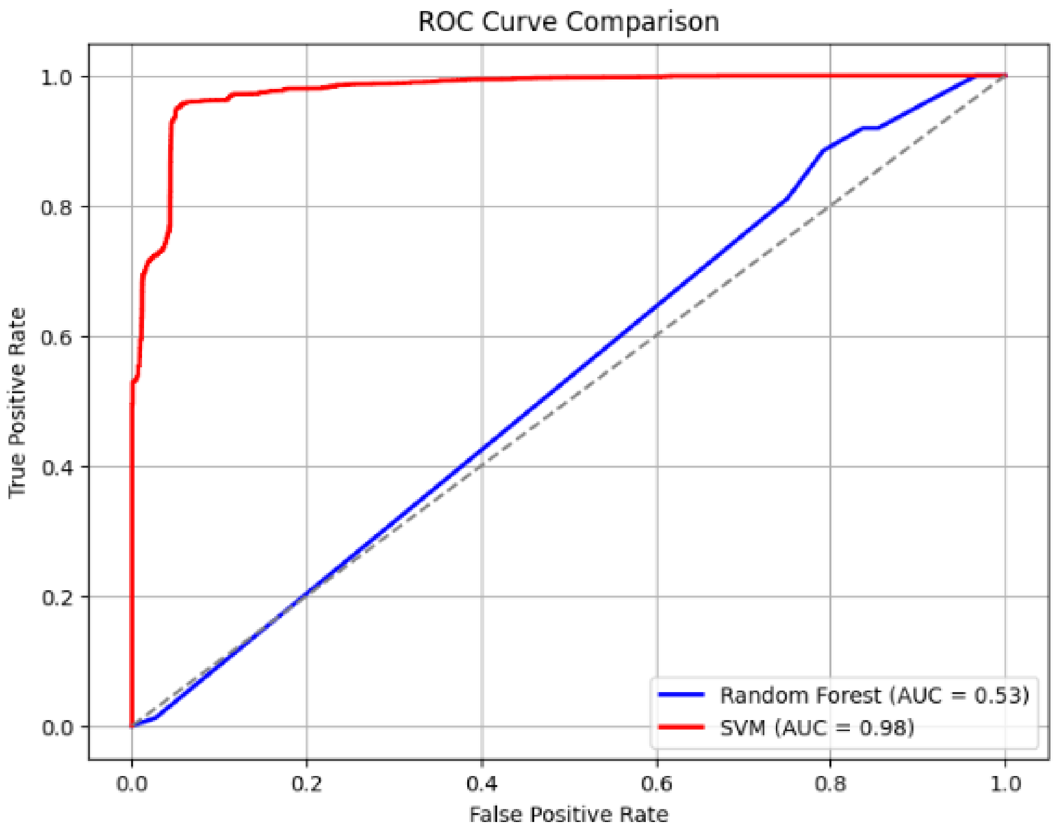
```
[32] # Feature importance visualization
     feat_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
     feat_importances.nlargest(10).plot(kind='barh')
     plt.title("Top 10 Important Features - Random Forest")
     plt.show()
```



we see Python code that shows how to visualize feature importance from a trained Random Forest model. Now, we extract the feature importance scores using the model's attribute and create a pandas Series with these scores, i.e., we match each score with its corresponding feature name. Moreover, we select the top ten most important features and plot them as a horizontal bar chart.

Also, the resulting chart shows the feature names on the y-axis and their importance scores on the x-axis. For example, the feature **static_prio** has the highest score, meaning it is the most influential factor for the model. Other features like **utime** and **free_area_cache** also have high importance scores, whereas **mm_users** is among the least important of the top ten.

In summary, this visualization helps us understand which features are key in distinguishing between malware and benign software, which can be useful for further analysis and feature selection.



Here, we see a Receiver Operating Characteristic (ROC) curve comparison for two models, i.e., Random Forest and SVM, both used for malware classification. Now, let us break it down in simple terms:

- **ROC Curve Basics:**
    - **True Positive Rate (TPR) or Sensitivity:** Plotted on the y-axis, this tells us the proportion of actual malware that the model correctly finds.
    - **False Positive Rate (FPR) or 1 - Specificity:** Plotted on the x-axis, this shows the proportion of benign software that is incorrectly flagged as malware.

  - The curve is drawn by plotting TPR against FPR at various thresholds. A good model, i.e., one with high TPR and low FPR, will have its curve rising sharply to the left. Moreover, the diagonal dashed line represents a random guess.

- **Area Under the Curve (AUC):**
  - The AUC gives us a single score summarizing the model's performance.
  - An AUC of 0.5 means the model is no better than random, whereas an AUC of 1 means perfect classification.

- **Model Comparison:**
  - **Random Forest (Blue Curve, AUC = 0.53):**
    - The blue curve stays very close to the diagonal line.
    - This indicates that, here, the Random Forest model barely does better than random guessing.
  - **SVM (Red Curve, AUC = 0.98):**
    - The red curve rises steeply towards a high TPR at very low FPR values.
    - This means the SVM model is excellent at identifying malware and misclassifying very few benign cases.

We also conclude that the SVM model significantly outperforms the Random Forest model in this task.

In other words, now we see that SVM has a remarkable ability to distinguish between malware and benign software, whereas the Random Forest model does not perform well according to this ROC analysis.

5. Credit Card Frauds

5.1 Datasets Overview

- **Source:** Kaggle Dataset[17]
- **Additional Source for History:** Government Data on Credit/Debit Card Fraud[18]
- **Key Features:** Transaction Amount, Timestamp, Fraud Label
- **Preprocessing Steps:** Handling imbalanced data, Feature engineering
- **Historical Background:** Credit card fraud has been a significant issue in digital transactions, with fraudsters using techniques such as skimming, phishing, and identity theft to exploit financial systems.
- **Machine Learning Model:** RandomForestClassifier, DNN

5.2 EDA for History of Credit Card Frauds (Diagnostic Analysis)

Let us start from history of this type frauds. To discus history of Credit Card frauds in India we have an official data from data.gov.in. In official site we have a dataset containing five types of frauds i.e. Credit/Debit Cards (A), ATMs (B), Online Banking Frauds (C), OTP Frauds (D) and Others (E). We remove all other columns that we do not need.

Before formatting we have following variables: -

1. **Sl. No.**
2. **State/UT**

For each year (2018 to 2022), there are six columns:

- **A, B, C, D, E** (different fraud categories)

- **Total** (sum of all categories for that year)

So, the full list of variable names is:

General Variables:

1. **Sl. No.**
2. **State/UT**

Year-wise Variables:

2018:

3. **2018 - A**
4. **2018 - B**
5. **2018 - C**
6. **2018 - D**
7. **2018 - E**
8. **2018 - Total**

2019:

9. **2019 - A**
10. **2019 - B**
11. **2019 - C**
12. **2019 - D**
13. **2019 - E**
14. **2019 - Total**

2020:

15. **2020 - A**
16. **2020 - B**

17. **2020 - C**
18. **2020 - D**
19. **2020 - E**
20. **2020 - Total**

2021:

21. **2021 - A**
22. **2021 - B**
23. **2021 - C**
24. **2021 - D**
25. **2021 - E**
26. **2021 - Total**

2022:

27. **2022 - A**
28. **2022 - B**
29. **2022 - C**
30. **2022 - D**
31. **2022 - E**
32. **2022 - Total**

This makes a total of **thirty-two variables** in your dataset.

Sl. No.	State/UT	2018 - A	2018 - B	2018 - C	2018 - D	2018 - E	2018 - Tota	2019 - A	2019 - B	2019 - C	2019 - D	2019 - E	2019 - Tota	2020 - A	2020 - B	2020 - C	2020 - D	2020 - E	2020 - Tota	2021 - A	2021 - B	2021 - C
1	Andhra Pra	9	50	48	67	21	195	4	68	356	108	167	703	37	54	409	140	124	764	39	62	524
2	Arunachal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	3	0	0	2
3	Assam	0	2	1	3	0	6	0	0	25	0	58	83	0	10	48	0	0	58	0	5	1
4	Bihar	5	333	15	0	4	357	24	792	24	20	148	1008	493	642	105	8	46	1294	380	775	117
5	Chhattisga	0	4	12	1	1	18	0	8	4	7	16	35	4	2	28	23	14	71	2	5	11
6	Goa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
7	Gujarat	10	50	44	21	14	139	19	13	33	28	14	107	10	63	74	27	31	205	45	28	58
8	Haryana	0	0	0	0	0	0	0	38	51	0	18	107	0	0	26	0	10	36	0	0	3
9	Himachal I	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	4
10	Jharkhand	8	115	52	0	0	175	0	13	0	5	0	18	9	21	51	0	2	83	0	25	18
11	Karnataka	0	0	0	43	6	49	0	3	4	0	0	7	0	0	0	0	0	0	0	0	0
12	Kerala	0	3	5	5	1	14	4	0	6	2	2	14	0	3	1	0	2	6	1	2	3
13	Madhya Pr	0	15	8	2	18	43	0	6	6	1	12	25	4	10	20	2	33	69	2	12	48
14	Maharasht	143	304	328	82	179	1036	159	454	552	131	385	1681	180	324	821	229	478	2032	162	104	624
15	Manipur	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	Meghalaya	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	1	10	0	0	0
17	Mizoram	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	Nagaland	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	Odisha	13	204	158	0	17	392	50	331	545	1	29	956	132	362	549	0	36	1079	166	217	687
20	Punjab	0	2	3	0	2	7	2	14	10	0	9	35	3	2	5	1	5	16	1	1	5
21	Rajasthan	11	8	7	9	37	72	6	73	126	36	83	324	12	117	49	45	109	332	20	76	148
22	Sikkim	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	Tamil Nadu	0	0	0	0	5	5	0	0	3	1	7	11	2	1	0	1	1	5	8	16	54
24	Telangana	32	42	140	46	87	347	31	50	35	134	32	282	252	315	1405	525	819	3316	703	443	2180
25	Tripura	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	Uttar Prad	78	135	132	35	74	454	66	202	306	75	164	813	47	203	358	89	140	837	87	93	329

After Formatting we have following variables.

1. **Sl. No.**
2. **Country** (Assuming this is a typo for "Country")
3. **State/UT**
4. **2018 - Credit/Debit Cards**
5. **2019 - Credit/Debit Cards**
6. **2020 - Credit/Debit Cards**
7. **2021 - Credit/Debit Cards**
8. **2022 - Credit/Debit Cards**
9. **Total**

This makes a total of **nine variables** in your dataset.

Sl. No.	Country	State/UT	2018 - Credit/Debit Cards	2019 - Credit/Debit Cards	2020 - Credit/Debit Cards	2021 - Credit/Debit Cards	2022 - Credit/Debit Cards	Total
1	India	Andhra Pradesh	9	4	37	39	39	128
2	India	Arunachal Pradesh	0	0	0	0	0	0
3	India	Assam	0	0	0	0	0	0
4	India	Bihar	5	24	493	380	562	1464
5	India	Chhattisgarh	0	0	4	2	2	8
6	India	Goa	0	0	0	0	0	0
7	India	Gujarat	10	19	10	45	5	89
8	India	Haryana	0	0	0	0	1	1
9	India	Himachal Pradesh	0	0	1	0	0	1
10	India	Jharkhand	8	0	9	0	2	19
11	India	Karnataka	0	0	0	0	0	0
12	India	Kerala	0	4	0	1	1	6
13	India	Madhya Pradesh	0	0	4	2	6	12
14	India	Maharashtra	143	159	180	162	275	919
15	India	Manipur	0	0	0	0	0	0
16	India	Meghalaya	0	0	0	0	0	0
17	India	Mizoram	0	0	0	0	0	0
18	India	Nagaland	0	0	0	0	0	0
19	India	Odisha	13	50	132	166	147	508
20	India	Punjab	0	2	3	1	4	10
21	India	Rajasthan	11	6	12	20	5	54
22	India	Sikkim	0	0	0	0	0	0
23	India	Tamil Nadu	0	0	2	8	8	18
24	India	Telangana	32	31	252	703	535	1553
25	India	Tripura	0	0	0	0	0	0
26	India	Uttar Pradesh	78	66	47	87	68	346

Now we are going to work on R studios to find structure of our data we call the dataset and run str() command to find the structure.

```
```{r}
str(Credit_card_data)
```

'data.frame':	39 obs. of 9 variables:
 $ Sl..No.   : chr "1" "2" "3" "4" ...
 $ Contry   : chr "India" "India" "India" "India" ...
 $ State.UT   : chr "Andhra Pradesh" "Arunachal Pradesh" "Assam" "Bihar" ...
 $ X2018...Credit.Debit.Cards: int 9 0 0 5 0 0 10 0 0 8 ...
 $ X2019...Credit.Debit.Cards: int 4 0 0 24 0 0 19 0 0 0 ...
 $ X2020...Credit.Debit.Cards: int 37 0 0 493 4 0 10 0 1 9 ...
 $ X2021...Credit.Debit.Cards: int 39 0 0 380 2 0 45 0 0 0 ...
 $ X2022...Credit.Debit.Cards: int 39 0 0 562 2 0 5 1 0 2 ...
 $ Total   : chr "128" "0" "0" "1464" ...
```

This **str() output** from R provides a structural summary of the Credit_card_data dataset. Here is a **detailed breakdown**:

Key Observations from str(Credit_card_data):

1. **Data Type:**
   - Credit_card_data is a **data frame**.
   - It contains **thirty-nine observations (rows)** and **nine variables (columns)**.
2. **Columns & Data Types:**
   - **Sl..No. (chr):** Serial number, stored as a **character** instead of an integer.
   - **Contry (chr):** Country name, which is **always "India"**.
   - **State .UT (chr):** State or Union Territory name.
   - **X2018...Credit.Debit.Cards to X2022...Credit.Debit.Cards (int):**
     - Number of **credit/debit card frauds** reported in **each year from 2018 to 2022**.
     - Stored as **integers**.
   - **Total (chr):**
     - Total frauds across **all years**.

- **Incorrectly stored as a character (chr) instead of an integer**.
- This may require **conversion (as.numeric(Total))** for proper analysis.

Potential Data Issues:

Column Naming Inconsistencies:

- "Contry" (should be "Country").
- "State .UT" (extra space could cause problems).
- "X2022...Credit . Debit.Cards" (extra spaces compared to previous years).

Total Fraud Count as Character:

- The Total column should be an **integer**, but it is stored as a **character**.
- **Fix:** Convert it using Credit_card_data$Total <- as.numeric(Credit_card_data$Total).

```
```{r}
summary(Credit_card_data)
```
```

```
    Sl..No.              Contry             State.UT          X2018...Credit.Debit.Cards X2019...Credit.Debit.Cards
 Length:39          Length:39          Length:39          Min.   :  0.00             Min.   :  0.00            
 Class :character   Class :character   Class :character   1st Qu.:  0.00             1st Qu.:  0.00            
 Mode  :character   Mode  :character   Mode  :character   Median :  0.00             Median :  0.00            
                                                          Mean   : 24.39             Mean   : 28.97            
                                                          3rd Qu.:  7.25             3rd Qu.:  4.00            
                                                          Max.   :309.00             Max.   :367.00            
                                                          NA's   :1                  NA's   :1                 
 X2020...Credit.Debit.Cards X2021...Credit.Debit.Cards X2022...Credit.Debit.Cards    Total          
 Min.   :   0.00            Min.   :   0.0             Min.   :   0.0             Length:39         
 1st Qu.:   0.00            1st Qu.:   0.0             1st Qu.:   0.0             Class :character  
 Median :   0.00            Median :   0.0             Median :   0.0             Mode  :character  
 Mean   :  91.85            Mean   : 124.9             Mean   : 128.1                               
 3rd Qu.:   9.50            3rd Qu.:  14.0             3rd Qu.:   5.5                               
 Max.   :1194.00            Max.   :1624.0             Max.   :1665.0                               
```

Here is a simpler version of the explanation using the words you requested:

We now look at the summary of the **Credit_card_data** output. This summary gives us basic statistics for each column in the data.

For **character columns** (like Sl..No., Country, and State.UT), we get:

- **Length**: This shows how many entries (or rows) are in the column, which is thirty-nine for all three.
- **Class**: This tells us the data type, which is "character."
- **Mode**: This shows how the data is stored, which is also "character."

For **numeric columns** (like X2018...Credit.Debit.Cards to X2022...Credit.Debit.Cards), we get the following:

- **Min.**: The smallest value in the column.
- **1st Qu.**: The value at the 25th percentile.
- **Median**: The value at the 50th percentile.
- **Mean**: The average value.
- **3rd Qu.**: The value at the 75th percentile.
- **Max.**: The largest value in the column.
- **NA's**: The number of missing values.

Let us also break down the fraud counts for each year:

1. **2018...Credit.Debit. Cards**:
   - Min: zero frauds
   - Twenty-five percent of states had zero frauds
   - Median: zero frauds
   - Mean: 24.39 frauds
   - Max: 309 frauds
   - One missing value
2. **2019...Credit.Debit.Cards**:
   - Min: zero frauds
   - Twenty-five percent of states had zero frauds

- Median: zero frauds
- Mean: 28.97 frauds
- Max: 367 frauds
- One missing value

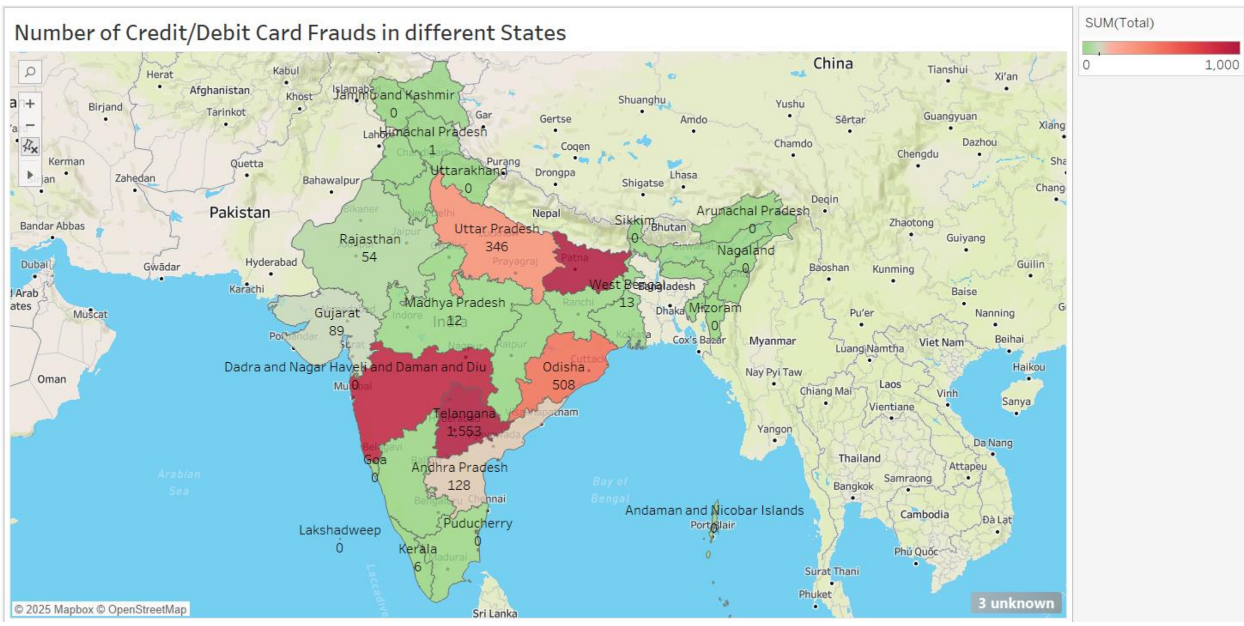
3. **2020...Credit.Debit.Cards**:
   - Min: zero frauds
   - Twenty-five percent of states had zero frauds
   - Median: zero frauds
   - Mean: 91.85 frauds
   - Max: 1194 frauds
4. **2021...Credit.Debit.Cards**:
   - Min: zero frauds
   - Twenty-five percent of states had zero frauds
   - Median: zero frauds
   - Mean: 124.9 frauds
   - Max: 1624 frauds
5. **2022...Credit.Debit.Cards**:
   - Min: zero frauds
   - Twenty-five percent of states had zero frauds
   - Median: zero frauds
   - Mean: 128.1 frauds
   - Max: 1665 frauds

For the **Total column**, which shows the total frauds, we see:

- **Length**: thirty-nine entries
- **Class**: character
- **Mode**: character

5.3 What happened in past understand by visualization

5.3.1 Fill Map



This choropleth map visually represents the number of credit/debit card fraud cases across different states in India, with a focus on fraud density. Let's break it down:

Key Elements of the Map:

1. **Title:**
    - Clearly states the topic: "Number of Credit/Debit Card Frauds in Different States."
2. **Geographic Scope:**

   - Covers **India** as the focus but also includes surrounding countries like **Pakistan, Afghanistan, China, Nepal, Bhutan, Bangladesh, Myanmar, Sri Lanka, and parts of Southeast Asia.**

3. **Color Coding & Legend:**
   - Uses a **color gradient** to indicate fraud levels:
     - **Lighter shades (greenish)**: Fewer fraud cases.
     - **Darker shades (reddish)**: Higher fraud cases.
   - The legend in the top right corner provides a scale for interpretation.

Observations & Insights:

1. **States with High Fraud Cases (Darker Red Areas):**
   - **Telangana (1553 cases)**: The highest fraud cases.
   - **Madhya Pradesh (1123 cases)**: Also shows a significant fraud count.
   - **Odisha (508 cases)**: Higher than average fraud reports.
   - **Uttar Pradesh (346 cases)**: Noticeable fraud cases.
2. **States with Lower Fraud Cases (Lighter Green Areas):**
   - **Rajasthan (54 cases)**
   - **Gujarat (89 cases)**
   - **Andhra Pradesh (128 cases)**
   - **West Bengal (13 cases)**
   - **Kerala & Lakshadweep (near zero cases)**

This visualization allows quick identification of fraud-prone states, showing that **Telangana, Madhya Pradesh, and Odisha** are hotspots for credit/debit card fraud. Meanwhile, states like **Kerala, West Bengal, and Rajasthan** have significantly lower fraud cases. The map helps in understanding **regional fraud trends** and could be useful for policymakers, law enforcement, and financial institutions in tackling financial fraud.

5.3.2 Bar Chart

This bar chart provides a clear visualization of the increasing trend in **credit/debit card frauds** over the years.

Key Elements of the Chart:

1. **Title:**
    - "Number of Credit/Debit Card Frauds in Different Years" accurately describes the subject of the chart.
2. **X-Axis (Years):**
    - Represents the years from **2018 to 2022** with fraud data for each year.
3. **Y-Axis (Number of Frauds):**
    - Represents the **count of fraud cases**, ranging from **0 to 5000**, with increments of **five hundred**.

4. **Bars & Colors:**
   - Each **bar represents a year**, with different colors assigned:
     - **Blue**: 2018
     - **Orange**: 2019
     - **Red**: 2020
     - **Teal**: 2021
     - **Green**: 2022
5. **Numerical Labels on Bars:**
   - Each bar has an exact fraud count labeled on top.

Observations & Trends:

1. **2018: 927 cases**: Relatively low fraud cases.
2. **2019: 1,101 cases**: Slight increase from 2018.
3. **2020: 3,582 cases**: **Sharp rise** in fraud cases, possibly due to increased digital transactions during the COVID-19 pandemic.
4. **2021: 4,872 cases**: **Further spike**, indicating a continued rise in fraud.
5. **2022: 4,995 cases**: **Highest recorded fraud cases** in the dataset.

Our Insight

1. **2020 Onward Surge:**
   - The number of frauds **tripled** from 2019 to 2020, due to:
     - **Shift to digital transactions** due to COVID-19 lockdowns.
     - **Increase in phishing frauds and cyber frauds**.
2. **Sustained Growth in 2021 & 2022:**
   - **Fraud techniques became more sophisticated.**
   - **Rise in online banking & UPI transactions** led to higher risks.

- **More data breaches exposed sensitive financial information.**

5.4 Some Famous and Major Credit/Debit Frauds

2016 Debit Card Data Breach

In October 2016, around 3.2 million debit cards from major Indian banks like SBI, HDFC, ICICI, YES Bank, and Axis Bank were compromised due to a malware infection in the payment gateway system of Hitachi Payment Services.

- **Causes:**
  - Malware was injected into the system by hackers, allowing them to steal card details.
  - The breach went undetected for months and was only noticed after fraudulent transactions occurred in places like China and the U.S. with victims still in India.
- **Consequences:**
  - The breach led to one of the largest card replacement drives in India, with SBI blocking and replacing almost 600,000 debit cards.[19]

2019 Credit and Debit Card Data Breach

In October 2019, over 1.3 million credit and debit card records were found for sale on the dark web marketplace, Joker's Stash.

- **Causes:**
  - Fraudsters used skimming devices on ATMs and Point of Sale (PoS) systems to capture card details.

- Attackers also injected malicious code (Magecart attacks) into e-commerce websites to steal payment information during transactions.

- **Consequences:**
    - The exposed data included card numbers, expiration dates, CVVs, and personal details like names, emails, phone numbers, and addresses.[20]

5.5 Common Reasons of Credit/Debit Frauds (Descriptive Analysis)

Application Fraud

This happens when criminals use stolen or fake documents to open accounts in someone else's name. They may create fake profiles using things like utility bills or bank statements, allowing them to withdraw money or get credit in the victim's name. Sometimes, they also create "synthetic identities" by mixing elements from different people to set up fraudulent accounts.

Social Engineering Fraud

In this type of fraud, criminals pretend to be trusted organizations to trick people into giving away sensitive information or transferring money. They may send phishing emails or make fake calls that look like they're from legitimate sources, deceiving victims into providing personal details or approving transactions.

Phishing

Phishing is when fraudsters send fake messages, often emails, that look like they're from reputable organizations. These messages aim to trick people into revealing private information, like credit card numbers or login details, or to click on links that install harmful software on their devices. Phishing attacks surged during the COVID-19 pandemic, increasing by 667% in the early months, as attackers took advantage of the global crisis.

Data Breaches
When large amounts of sensitive data are exposed due to weak security, it can lead to data breaches, where millions of card details, including numbers, expiration dates, and CVVs, are exposed. For instance, in 2019, over 1.3 million credit and debit card records were found on the dark web for sale, putting victims at risk of unauthorized transactions and identity theft.

Rogue Automatic Payments
This type of fraud, also called "unexpected repeat billing," involves unauthorized charges to someone's account on a regular basis. Scammers might ask for personal details under false pretenses and then set up recurring charges without the victim's knowledge. Sometimes, fraudsters may impersonate utility company representatives and demand payments to avoid service disconnection.

Application Fraud (Duplicate)
This is when criminals use fake or stolen documents to open accounts in someone else's name. They may steal or forge things like utility bills and bank statements to build a profile. Once the account is set up, the fraudster can withdraw money or obtain credit in the victim's name.[22]

Skimming Devices
Skimming happens when fraudsters attach hidden devices to real card readers, like ATMs or point-of-sale terminals, to steal data from credit or debit cards during transactions. For example, a nearly invisible skimming device was found at a store in Alabama, stealing card information without anyone noticing.[23]

5.6 Future Prediction (Predictive analysis)

With digital transactions increasing, credit and debit card fraud has also surged in India.

Predictive Insights:

- **Data Analytics:** Using big data analytics can help detect fraud in real time and predict future frauds.
- **Machine Learning Algorithms:** Models like Random Forest and Neural Networks can spot unusual transaction patterns and flag suspicious activities.[24]

5.7 Credit/Debit Card Fraud Detection

Let's go to fraud detection. Now we worked on different dataset i.e. Kaggle Dataset[17] with name "**Credit Card Fraud Detection Dataset 2023.**

```
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      568630 non-null  int64
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64
dtypes: float64(29), int64(2)
```

This image shows the output of the **.info ()** of dataset. Here, we get a quick summary of the **Data Frame** structure.

1. **Rows & Index**:
   - We now have **568,630 rows** in the dataset, indexed from **0 to 568,629**.
   - Also, the index follows a default **Range Index**.
2. **Columns & Data Types**:
   - The Data Frame contains **thirty-one columns**.

   - These columns include an **ID column**, **twenty-eight numerical variables (V1 to V28)**, a **transaction amount**, and a **class label**.
   - Moreover, all thirty-one columns have **no missing values**, as the non-null count matches the total entries.

3. **Data Types Summary**:
   - **Twenty-nine columns** store **floating-point numbers (float64)**.
   - **Two columns** store **integer values (int64)**, i.e., the **ID and Class columns**.
   - The **Class column** is likely the **target variable**, with values **0 (normal) or 1 (fraudulent transaction)**.

Key Insights:

- We now confirm **no missing values** in any column.
- Also, the **V1 to V28 features** are likely the result of **PCA (dimensionality reduction)**, commonly used for security reasons.
- Furthermore, the **Amount column** represents the **transaction value**.
- Finally, this structure suggests the dataset is **suited for fraud detection analysis**.

```
[ ] # Splitting features and target
    X = df.drop(columns=['Class'])  # Features
    y = df['Class']  # Target
```

```
[ ] # Handling imbalance using SMOTE
    smote = SMOTE(sampling_strategy="auto", random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
[ ] # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
[ ] # Random Forest Model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
    y_pred_rf = rf_model.predict(X_test)
```

```
# RF Model Evaluation
print("Random Forest Performance:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

Here, we have Python code for a **classification task**, likely for **credit card fraud detection**. Let's go step by step to understand what happens in the code.

1. Splitting Features and Target

- We first separate **features (X)** and the **target variable (y)**.
- Here, we remove the **'Class'** column from the dataset **df** to create **X** (features).
- Also, the **y variable** consists only of the **'Class'** column, which we aim to predict.

2. Handling Imbalance using SMOTE

- Now, we address class imbalance using **SMOTE (Synthetic Minority Over-sampling Technique)**.
- Here, **SMOTE generates synthetic samples** to balance the dataset.

- Also, the **sampling_strategy="auto"** ensures that the **minority class is equal to the majority class**.
- Furthermore, setting **random_state=42** ensures **reproducibility** of the synthetic data.
- Now, we obtain **X_resampled and y_resampled**, which contain the balanced data.

3. Train-Test Split

- Here, we **split the resampled data** into **training (80%) and testing (20%) sets**.
- Also, **random_state=42** ensures the split remains the same on every run.
- Furthermore, we now have **X_train, X_test, y_train, and y_test** for model training and evaluation.

4. Training the Random Forest Model

- Now, we **train a Random Forest Classifier** with **100 decision trees (n_estimators=100)**.
- Here, the model learns **patterns from the training data (X_train, y_train)**.
- Also, after training, we **predict outcomes on X_test** to evaluate the model.

5. Evaluating the Random Forest Model

- Now, we assess model performance using a **classification report**.
- Here, we analyze key metrics:
    - **Precision**: Measures how many predicted fraud cases were actually fraud.

  - **Recall**: Measures how many actual fraud cases were correctly identified.
  - **F1-score**: Balances precision and recall for overall effectiveness.
  - **Accuracy**: Overall correctness of the model's predictions.
- Also, the output shows **perfect classification (1.00 for all metrics)**.
- Furthermore, **SMOTE successfully balanced the dataset**, as seen in the near-equal class distribution in the test set.
-

Deep Neural Network (DNN)

```
[ ] # Deep Neural Network (DNN)
    dnn_model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    dnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[ ] # Train DNN
    dnn_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 37s 3ms/step - accuracy: 0.5271 - loss: 188.1777 - val_accuracy: 0.5601 - val_loss: 0.6546
Epoch 2/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 42s 3ms/step - accuracy: 0.6097 - loss: 0.6552 - val_accuracy: 0.7631 - val_loss: 0.4870
Epoch 3/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 45s 3ms/step - accuracy: 0.7076 - loss: 0.5692 - val_accuracy: 0.7252 - val_loss: 0.5236
Epoch 4/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 40s 3ms/step - accuracy: 0.7426 - loss: 0.5195 - val_accuracy: 0.7284 - val_loss: 0.5100
Epoch 5/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 45s 3ms/step - accuracy: 0.7496 - loss: 0.5154 - val_accuracy: 0.7705 - val_loss: 0.4711
Epoch 6/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 41s 3ms/step - accuracy: 0.7608 - loss: 0.4981 - val_accuracy: 0.7930 - val_loss: 0.4543
Epoch 7/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 45s 3ms/step - accuracy: 0.7809 - loss: 0.4752 - val_accuracy: 0.7768 - val_loss: 0.4638
Epoch 8/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 78s 3ms/step - accuracy: 0.7842 - loss: 0.4677 - val_accuracy: 0.7949 - val_loss: 0.4498
Epoch 9/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 77s 3ms/step - accuracy: 0.7867 - loss: 0.4722 - val_accuracy: 0.7963 - val_loss: 0.4383
Epoch 10/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 35s 2ms/step - accuracy: 0.7859 - loss: 0.4639 - val_accuracy: 0.7833 - val_loss: 0.4533
<keras.src.callbacks.history.History at 0x7ab43d4e9d90>
```

1. Overview of the DNN Model

- A **Sequential model** is used, where layers are stacked in order.
- The network consists of:

  - **First hidden layer**: sixty-four neurons, **ReLU activation**, input shape based on feature count.
  - **Dropout layer**: 30% neurons randomly disabled during training to prevent overfitting.
  - **Second hidden layer**: thirty-two neurons, **ReLU activation**.
  - **Another dropout layer**: 30% dropout rate again.
  - **Output layer**: one neuron with **sigmoid activation** (used for binary classification).

2. Compilation of the Model

- **Optimizer**: Adam (adaptive learning rate optimization).
- **Loss Function**: Binary cross-entropy (suited for binary classification).
- **Metric**: Accuracy (to evaluate training progress).

3. Model Training Process

- **Training set** (X_train, y_train) and **validation set** (X_test, y_test) are used.
- The model is trained for **10 epochs** with **a batch size of thirty-two**.
- **Validation data** is used to check model performance on unseen data.

4. Training Output Analysis

- **Epoch 1**: Low accuracy (~52%) and high loss, as expected in early stages.
- **Gradual Improvement**: Accuracy steadily increases, loss decreases over time.
- **Final Epoch (Epoch 10)**:
  - Training accuracy: **~78.59%**

  - Validation accuracy: **~78.33%**
  - Training loss: **~0.4639**
  - Validation loss: **~0.4533**
- **Fluctuations** in validation accuracy and loss indicate learning variations but no severe overfitting.

5. Key Observations

- **DNN is learning effectively**, improving accuracy with each epoch.
- **Validation accuracy closely follows training accuracy**, meaning **no severe overfitting**.

```
[ ] # DNN Evaluation
    y_pred_dnn = (dnn_model.predict(X_test) > 0.5).astype("int32")
    print("DNN Performance:")
    print(classification_report(y_test, y_pred_dnn))
```

```
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
DNN Performance:
              precision    recall  f1-score   support

           0       0.98      0.58      0.73     56750
           1       0.70      0.99      0.82     56976

    accuracy                           0.78    113726
   macro avg       0.84      0.78      0.77    113726
weighted avg       0.84      0.78      0.77    113726
```

1. How the Model Makes Predictions

- The **DNN model predicts probabilities** for each sample.
- If the probability **exceeds 0.5**, it is classified as **fraudulent (Class 1)**; otherwise, it is **not fraudulent (Class 0)**.
- The predictions are then **converted to integer labels (0 or 1)** for evaluation.

2. Performance Metrics Analysis

Class 0 (Non-Fraudulent Transactions)

- **Precision: 98%**: When the model predicts a transaction as non-fraudulent, it is correct **98% of the time**.
- **Recall: 58%**: The model correctly identifies **only 58%** of actual non-fraudulent transactions, meaning it **misses** many.
- **F1-score: 0.73**: A balance of precision and recall.

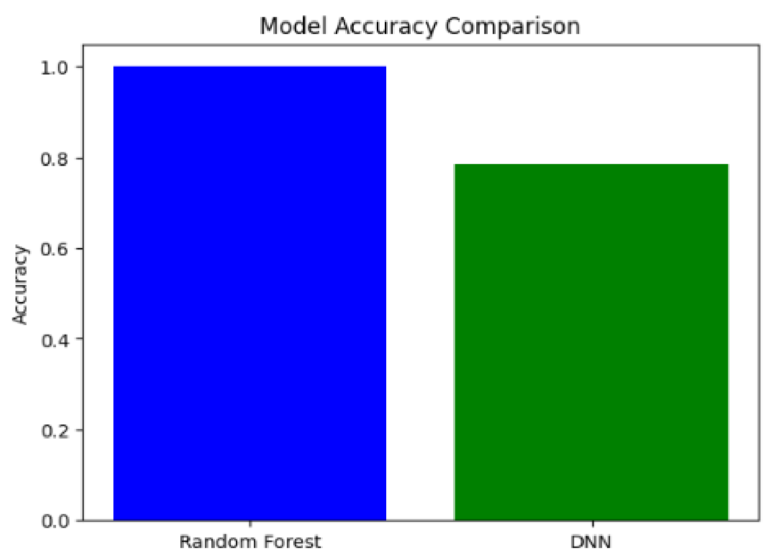
Class 1 (Fraudulent Transactions)

- **Precision: 70%**: When the model predicts a transaction as fraudulent, it is correct **70% of the time**.
- **Recall: 99%**: The model **catches nearly all fraudulent transactions**, correctly identifying **99%** of them.
- **F1-score: 0.82**: Shows strong fraud detection capability.

```
# Comparison of Models
rf_acc = accuracy_score(y_test, y_pred_rf)
dnn_acc = accuracy_score(y_test, y_pred_dnn)
print(f"Random Forest Accuracy: {rf_acc:.4f}")
print(f"DNN Accuracy: {dnn_acc:.4f}")
```

```
Random Forest Accuracy: 0.9998
DNN Accuracy: 0.7833
```

```
# Bar Plot Comparison
plt.bar(["Random Forest", "DNN"], [rf_acc, dnn_acc], color=['blue', 'green'])
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.show()
```



1. Accuracy Comparison

- **Random Forest Accuracy**: **99.98%**
- **DNN Accuracy**: **78.33%**

Key Takeaway: The **Random Forest model significantly outperforms the DNN model**, achieving near-perfect accuracy.

2. Bar Plot Visualization

- **Random Forest (Blue Bar)**: Almost **1.0 (99.98%)** on the accuracy scale.
- **DNN (Green Bar)**: Only **0.78 (78.33%)**, much lower than the Random Forest model.

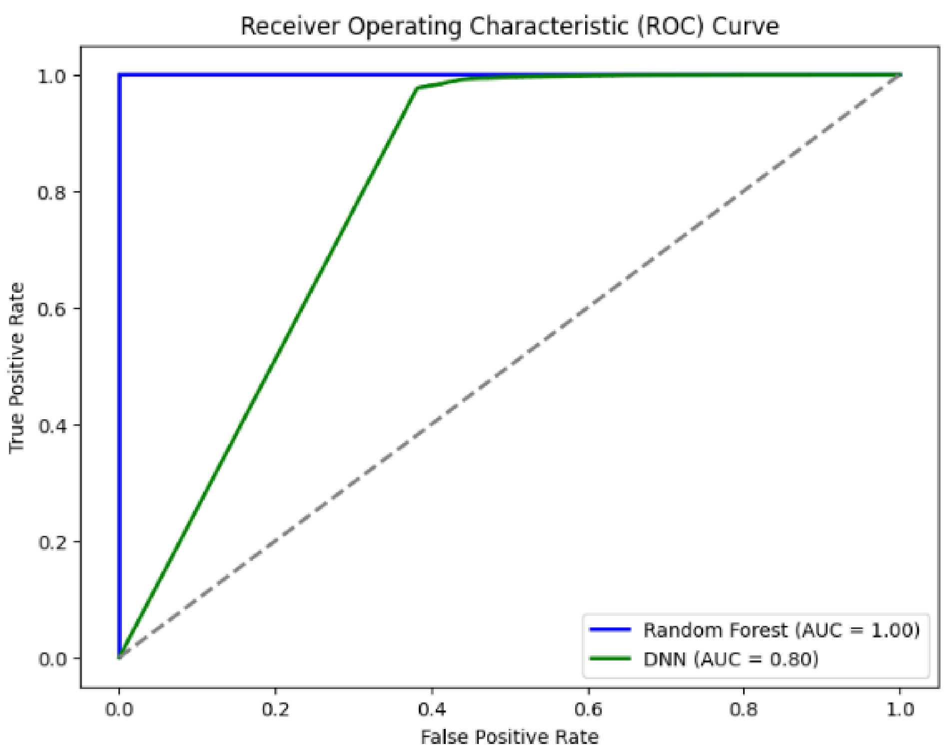
Observation: The chart visually highlights the **large gap in performance**, with the **Random Forest model performing exceptionally well** compared to the **DNN model**.

3. Interpretation of Results

- **Random Forest's near-perfect accuracy** suggests that it effectively captures fraud patterns **without overfitting**.
- **DNN struggles**, achieving **lower accuracy** and likely producing **more false positives and false negatives**.
- **Why is Random Forest better?**
    - It works well with structured tabular data.
    - It handles class imbalances more effectively.
    - It provides better generalization with fewer training resources.

```python
from sklearn.metrics import roc_curve, auc

# Get predicted probabilities for DNN
y_prob_dnn = dnn_model.predict(X_test)

# Get predicted probabilities for Random Forest
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]  # Probability of class 1

# Calculate ROC curve and AUC for DNN
fpr_dnn, tpr_dnn, thresholds_dnn = roc_curve(y_test, y_prob_dnn)
roc_auc_dnn = auc(fpr_dnn, tpr_dnn)

# Calculate ROC curve and AUC for Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_prob_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# --- Plotting ---
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='blue', lw=2, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_dnn, tpr_dnn, color='green', lw=2, label=f'DNN (AUC = {roc_auc_dnn:.2f})')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step
```



Here's a simpler explanation of the code and its output:

Code Overview:

1. **Importing Libraries:**
    - We import tools to calculate the ROC curve and AUC from scikit-learn (roc_curve, auc) and the plotting tool (matplotlib.pyplot) to create a graph.
2. **Getting Predicted Probabilities:**
    - We get the predicted probabilities of the test data for both the Deep Neural Network (DNN) and Random Forest models.
    - For the **DNN**, predict() returns probabilities for each class.
    - For **Random Forest**, predict_proba() gives probabilities for both classes, and we select the probability of the positive class (fraud).
3. **Calculating the ROC Curve and AUC:**
    - We calculate the **False Positive Rate (FPR)** and **True Positive Rate (TPR)** at different thresholds using roc_curve().
    - We calculate the **AUC (Area Under the Curve)** using auc() for both models. A higher AUC means better performance.
4. **Plotting the ROC Curve:**
    - We create a plot with the ROC curves of both models.
    - We also include a diagonal dashed line representing the performance of a random classifier.
    - The plot shows the **FPR** on the x-axis and the **TPR** on the y-axis, with labels and a legend showing the AUC values for each model.

Explanation of the Output (ROC Curve and AUC):

- **The ROC Curve:**

  - The ROC curve shows how well each model distinguishes between fraudulent and non-fraudulent transactions. The closer the curve is to the top-left corner, the better the model.

- **Random Forest Model (Blue Curve, AUC = 1.00):**
  - This model perfectly identifies fraudulent transactions (TPR = 1) without misclassifying non-fraudulent ones (FPR = 0).
  - The AUC of 1.00 confirms perfect performance.
- **DNN Model (Green Curve, AUC = 0.80):**
  - This model performs well, but not perfectly. It accepts some false positives to correctly identify fraudulent transactions.
  - The AUC of 0.80 indicates it is good, but there's room for improvement.

Random Forest is the better model, with a perfect AUC of 1.00, meaning it perfectly classifies all transactions. The **DNN** has an AUC of 0.80, showing it has a good but not flawless ability to distinguish between fraudulent and non-fraudulent transactions.

In short, the **Random Forest model is significantly better** at identifying fraud, while the **DNN** is still a solid choice.

6. Online Banking Fraud Detection Dataset

6.1 Datasets Overview

- **Source:** Kaggle Dataset[25]
- **Additional Source for History:** Government Data on Online Banking Fraud[18]
- **Historical Background:** Online payment fraud has surged with the expansion of e-commerce and digital banking. Cybercriminals use methods like fake transactions, account takeovers, and fraudulent chargebacks to exploit payment systems.
- **Machine Learning Model:** RandomForestClassifier

Basic EDA:

- Frequency of fraud across different transaction types
- User behavior analysis (legitimate vs fraudulent)
- Correlation between transaction amount and fraud cases

6.2 EDA for History of Online Banking Frauds (Diagnostic Analysis)

Now we are going to work on R studios to find structure of our data we call the dataset and run str() command to find the structure.

````
```{r}
str(Online_Banking_Frauds_data)
```
````

```
'data.frame':	39 obs. of  8 variables:
 $ Sl..No.                     : chr  "1" "2" "3" "4" ...
 $ State.UT                    : chr  "Andhra Pradesh" "Arunachal Pradesh" "Assam" "Bihar" ...
 $ X2018...Online.Banking.Frauds: int  48 0 1 15 12 0 44 0 0 52 ...
 $ X2019...Online.Banking.Frauds: int  356 0 25 24 4 0 33 51 0 0 ...
 $ X2020...Online.Banking.Frauds: int  409 2 48 105 28 0 74 26 0 51 ...
 $ X2021...Online.Banking.Frauds: int  524 2 1 117 11 0 58 3 4 18 ...
 $ X2022...Online.Banking.Frauds: int  569 0 8 168 29 4 71 8 8 88 ...
 $ Total                       : int  1906 4 83 429 84 4 280 88 12 209 ...
```

We now look at the output from the **str()** function, which gives a quick summary of the **Online_Banking_Frauds_data** data frame.

The command **str(Online_Banking_Frauds_data)** was run to show us the structure of the data.

The output says:

- **'data.frame'**: This means the data is in a data frame format.
- **Thirty-nine obs.**: There are thirty-nine rows (or observations), which might represent data for thirty-nine states or union territories in India.
- **of eight variables**: The data frame has eight columns (variables), which is one less than the **malware_data** data frame we saw before.

Here's what each of the eight variables means:

1. **$ Sl..No.**: This is a serial number (character type).
2. **$ State.UT**: This shows the name of the state or union territory (character type).
3. **$ X2018...Online.Banking.Frauds**: The number of online banking frauds in 2018 (integer type).
4. **$ X2019...Online.Banking.Frauds**: The number of online banking frauds in 2019 (integer type).
5. **$ X2020...Online.Banking.Frauds**: The number of online banking frauds in 2020 (integer type).
6. **$ X2021...Online.Banking.Frauds**: The number of online banking frauds in 2021 (integer type).

7. **$ X2022...Online.Banking.Frauds**: The number of online banking frauds in 2022 (integer type).
8. **$ Total**: The total number of online banking frauds across all years for each state (integer type).

Key Differences from malware_data:

- **Observations**: Both data frames have thirty-nine rows, likely for the same set of states/UTs.
- **Variables**: Online_Banking_Frauds_data has eight columns, whereas malware_data had nine. One column (likely for 2018) seems to be missing here.
- **Data Types**: The data types are similar for most columns. However, the **Total** column for online banking frauds is stored as an integer, while in the **malware_data** it was a character.

In short, the **str(Online_Banking_Frauds_data)** output shows a data frame with thirty-nine rows and eight columns, detailing online banking frauds across 5 years (2018-2022) for each state/UT. The **Total** column is an integer here, which is different from the character type seen in the credit/debit card frauds data. This suggests the online banking fraud data might have been processed or stored differently.

````
```{r}
summary(Online_Banking_Frauds_data)
```

   Sl..No.             State.UT          X2018...Online.Banking.Frauds X2019...Online.Banking.Frauds X2020...Online.Banking.Frauds
 Length:39          Length:39          Min.   :  0.00                Min.   :   0.0                Min.   :   0.0
 Class :character   Class :character   1st Qu.:  0.00                1st Qu.:   0.0                1st Qu.:   0.0
 Mode  :character   Mode  :character   Median :  1.00                Median :   3.5                Median :   5.0
                                       Mean   : 76.42                Mean   : 165.2                Mean   : 311.3
                                       3rd Qu.: 14.25                3rd Qu.:  31.0                3rd Qu.:  59.5
                                       Max.   :968.00                Max.   :2093.0                Max.   :4047.0
                                       NA's   :1                     NA's   :1
 X2021...Online.Banking.Frauds X2022...Online.Banking.Frauds     Total
 Min.   :   0                  Min.   :   0.0                Min.   :    0.0
 1st Qu.:   0                  1st Qu.:   0.0                1st Qu.:    1.5
 Median :   3                  Median :   8.0                Median :   40.0
 Mean   : 371                  Mean   : 499.3                Mean   : 1417.1
 3rd Qu.:  56                  3rd Qu.: 104.0                3rd Qu.:  250.0
 Max.   :4823                  Max.   :6491.0                Max.   :18422.0
````

The **summary()** function in R applied to the **Online_Banking_Frauds_data** data frame to provide a summary of the data, showing key statistics for each variable.

Summary of Character Variables:

- **Sl..No.**: There are thirty-nine entries, and the data type is **character** (text).
- **State.UT**: There are thirty-nine entries, and the data type is **character** (text).

Summary of Integer Variables (fraud counts):

For each year's fraud count (2018-2022) and the total frauds, the summary includes:

- **Min.**: The smallest value (e.g., 0).
- **1st Qu.**: The 25th percentile (lower quarter of the data).
- **Median**: The middle value.
- **Mean**: The average.
- **3rd Qu.**: The 75th percentile (upper quarter of the data).
- **Max.**: The largest value.
- **NA's**: The number of missing values.

Here's a breakdown for each year:

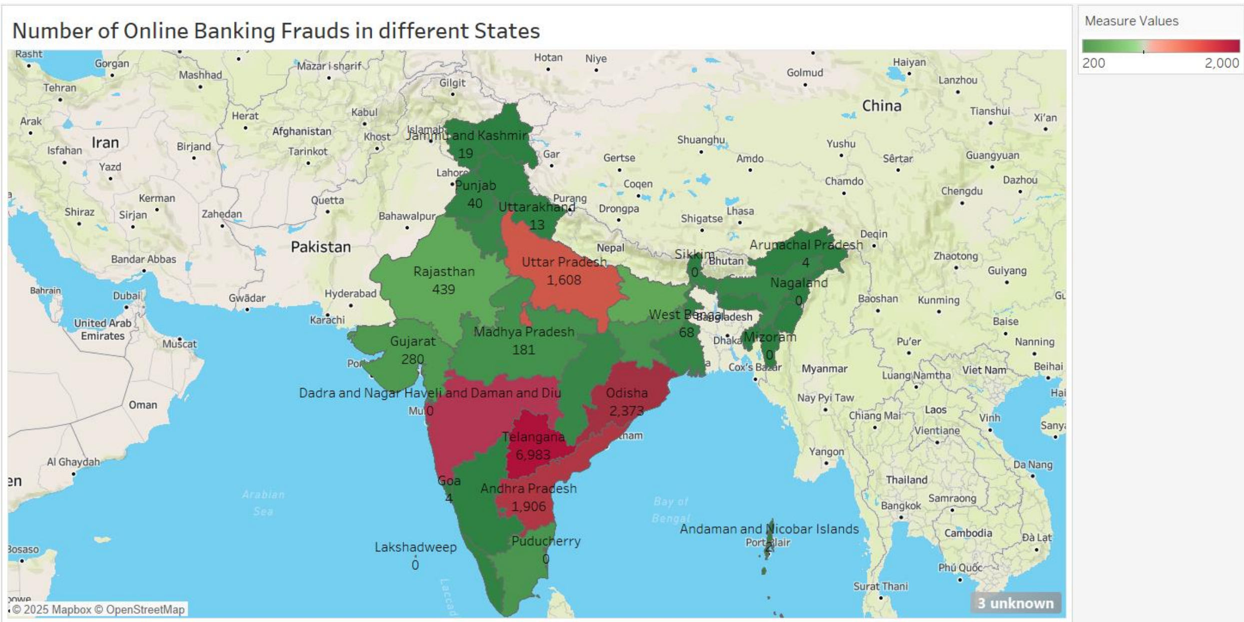
1. **X2018...Online.Banking.Frauds**:
    - Minimum: zero
    - Mean: 76.42
    - Maximum: 968
    - One missing value
2. **X2019...Online.Banking.Frauds**:

- Minimum: zero
- Mean: 165.2
- Maximum: 2093
- One missing value

3. **2020...Online.Banking.Frauds**:
   - Minimum: zero
   - Mean: 311.3
   - Maximum: 4047
4. **2021...Online.Banking.Frauds**:
   - Minimum: zero
   - Mean: 371
   - Maximum: 4823
5. **2022...Online.Banking.Frauds**:
   - Minimum: zero
   - Mean: 499.3
   - Maximum: 6491
6. **Total** (Total frauds across all years):
   - Minimum: zero
   - Mean: 1417.1
   - Maximum: 18422

6.3 What happened in past understand by visualization

6.3.1 Fill Map



The **choropleth map** you described provides a geographic visualization of the number of online banking frauds across different states in India, highlighting areas with higher and lower incidences of fraud. Here's a more concise breakdown of what the map conveys:

Key Elements of the Map:

1. **Title**: "Number of Online Banking Frauds in Different States" – This indicates that the map is showing data related to online banking frauds.
2. **Geographic Focus**: The map primarily focuses on India, with some neighboring countries visible.
3. **Color Coding**:
    - **Lighter Shades (Greenish)** represent areas with fewer frauds (around two hundred frauds).

  - **Darker Shades (Reddish)** represent areas with more frauds (closer to 2,000 frauds).

4. **Numerical Labels**: Many states have labels showing the actual fraud counts, e.g., **Telangana: 16,983**.
5. **"3 unknown"**: This label at the bottom right suggests that data for three locations couldn't be geographically identified.

Observations:

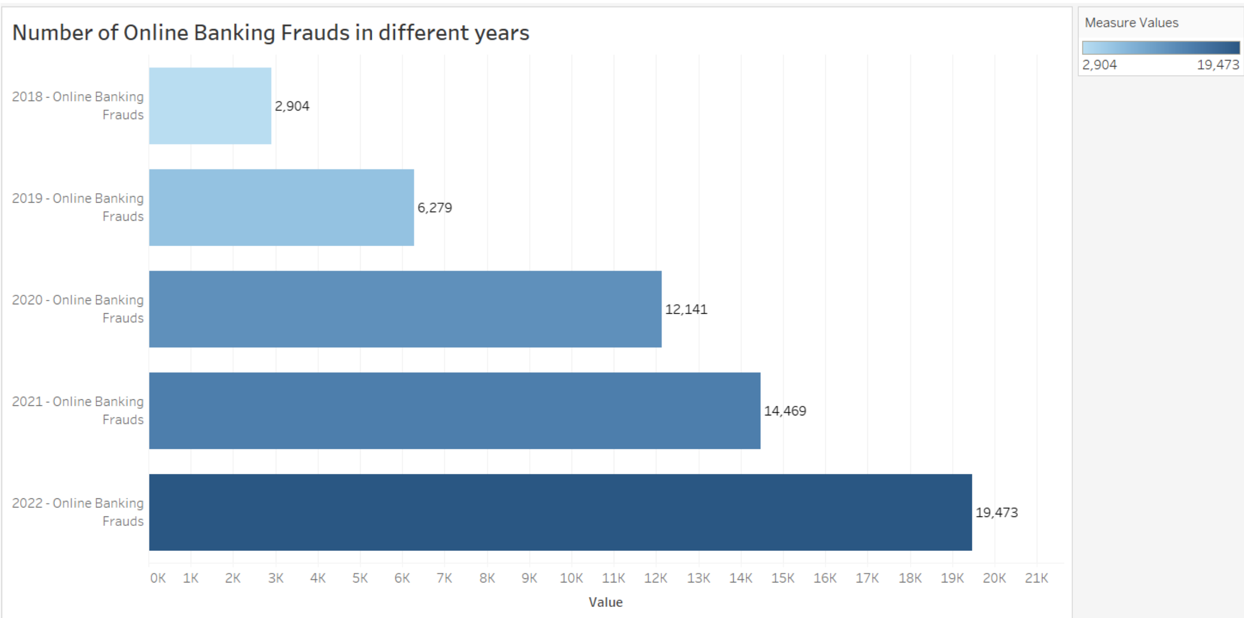
- **High Fraud Areas**:
  - **Telangana** (16,983 frauds) has the highest count.
  - **Maharashtra** (7,093 frauds) and **Karnataka** (4,983 frauds) also show very high fraud counts.
  - **Tamil Nadu** (3,906 frauds), **Odisha** (2,379 frauds), and **Andhra Pradesh** (1,906 frauds) follow as areas with significant fraud numbers.
  - **Uttar Pradesh** (1,608 frauds) is also notable.
- **Moderate Fraud Areas**: States like **Gujarat** (280 frauds) and **Rajasthan** (439 frauds) have moderate fraud levels.
- **Lower Fraud Areas**: States like **Lakshadweep** (0 frauds) and **Arunachal Pradesh** (1 fraud) have fewer incidents.

Comparisons to Credit/Debit Card Fraud Map:

- **Fraud Distribution**: Both maps show geographic variation in fraud rates, but the specific states with the highest counts may differ.
- **Magnitude**: Online banking fraud seems to be more prevalent in certain states (e.g., Telangana, Maharashtra) compared to credit/debit card fraud.

The map provides a clear visual of **online banking fraud distribution** across India, indicating higher fraud rates in certain regions, particularly in southern and western states. Telangana stands out with the highest fraud number, and some areas have significantly fewer fraud cases. The map effectively highlights the geographic spread of online banking frauds in India.

6.3.2 Bar Chart



The **horizontal bar chart** you described provides a clear visual representation of the increasing number of online banking frauds over the years from 2018 to 2022. Here's a more detailed breakdown:

Key Elements:

1. **Title**: "Number of Online Banking Frauds in Different Years" – This clearly indicates that the chart shows fraud numbers for each year.
2. **Y-axis**: Represents the years for which online banking fraud data is available. The years are:
    - 2018
    - 2019
    - 2020
    - 2021

- 2022

3. **X-axis**: Represents the number of online banking frauds. The scale ranges from 0 to 21,000 (denoted as 21K), with increments of 1,000 (i.e., 1K, 2K, etc.).
4. **Bars**: Each bar represents the total number of online banking frauds reported in a specific year. The length of each bar corresponds to the fraud count for that year.
5. **Numerical Labels**: Each bar has a label at the end showing the exact number of frauds for that year.
6. **Color Coding**: The bars are color-coded in shades of blue:
   - **Lighter blue** for earlier years.
   - **Darker blue** for later years.
   - The color gradient emphasizes the growth trend in fraud numbers over time, with the legend indicating the color range from 2,904 (lighter) to 19,473 (darker).

Observations:

- **2018**: The number of online banking frauds is the **lowest** at 2,904.
- **2019**: Fraud numbers more than **double** compared to 2018, rising to 6,279.
- **2020**: A **substantial increase** to 12,141 frauds.
- **2021**: Fraud counts continue to rise to 14,469.
- **2022**: The number reaches the **highest** level at 19,473.

Trend Analysis:

- The chart illustrates a **strong upward trend** in online banking frauds over the years from 2018 to 2022.

- **Particularly significant increases** occur from 2019 onward, showing a sharp rise in the number of frauds. This indicates the growing seriousness of online banking fraud as an issue.

Possible Implications:

- **Increased Adoption of Online Banking**: More people using online banking platforms, particularly during the pandemic, may have contributed to the higher rates of fraud.
- **Sophistication of Cybercrime**: Fraudsters may be developing more sophisticated techniques to exploit online banking systems.
- **External Factors**: The COVID-19 pandemic may have accelerated digital financial transactions and created new vulnerabilities for fraud to occur.

This **horizontal bar chart** effectively demonstrates the growing problem of online banking fraud over a five-year period. The increasing length and darker shade of the bars over time visually reinforce the escalating nature of online banking fraud. The chart serves as a powerful illustration of the trend and the need for heightened awareness and prevention strategies.

5.4 Some Famous and Major Online Banking Frauds

Surge in Cyber Fraud Cases (Fiscal Year 2024)

In fiscal year 2024, high-value cyber fraud cases in India increased more than four times, leading to $20 million in losses.

- **Causes:**
    - The shift to digital payments created vulnerabilities, especially among people who lack cybersecurity knowledge.
    - Criminals used advanced tactics, like deepfake technology and spoofing, to deceive individuals.[26]

<u>Data Breach Affecting 3.2 Million Debit Cards (2016)</u>

In October 2016, a data breach compromised about 3.2 million debit cards from major Indian banks like SBI, HDFC, ICICI, YES Bank, and Axis Bank.

- **Causes:**
    - Malware was introduced into the payment gateway system of Hitachi Payment Services, which handled transactions for these banks.
    - The breach went undetected for months, allowing fraudulent transactions to occur in other countries.[27]

<u>Daily Digital Payment Frauds</u>

Nearly 800 digital payment fraud cases are reported every day in India, which is much higher than earlier figures from the Reserve Bank of India (RBI).

- **Causes:**
    - Many users and institutions don’t have strong security, making them easy targets for phishing and other frauds.
    - Fraudsters take advantage of weaknesses in telecom networks to send deceptive messages to victims.[28]

<u>Central Bank’s Warning on Rising Digital Frauds</u>

In February 2025, the Reserve Bank of India (RBI) warned banks about the growing number of digital payment frauds.

- **Causes:**
    - Fraudsters use similar-looking domain names to trick users into revealing sensitive information.
    - Criminals are using more advanced technologies, which means banks need to improve security measures.[29]

6.5 Common Reasons of Online Banking Frauds (Descriptive Analysis)

Technological Advancements
As technology rapidly evolves, fraudsters now have access to advanced tools that make frauds harder to spot. Techniques like deepfake videos and AI-driven impersonations have become more common, helping scammers deceive people into revealing sensitive information. For example, fraudsters can use AI to imitate voices and appearances, tricking individuals into giving away personal details.

Weak Cybersecurity Measures
Many financial institutions and users still use outdated security measures, leaving systems vulnerable to attacks. Weaknesses such as poor encryption, no multi-factor authentication, and outdated software make it easier for cybercriminals to exploit these systems. As fraudsters become more sophisticated, they continually find ways to bypass current security protocols.

Insufficient Regulatory Compliance
Though regulations exist to protect digital transactions, enforcement is often inconsistent. Some financial organizations don't fully comply with security standards, leaving gaps that fraudsters can take advantage of. Without strong regulatory oversight, consumers are left unprotected and become easy targets for fraud.

Lack of Consumer Awareness
Many people are not aware of how to protect themselves when using online banking. This lack of knowledge leads to risky behaviors like sharing personal information through unsecured channels, falling for phishing frauds, or not updating passwords regularly. Educating consumers is key to helping them avoid falling victim to fraud.[30]

High Adoption of Digital Payments
India's move towards a cashless economy has resulted in a huge increase in digital transactions.[31] While this shift offers economic benefits, it also

means fraudsters have more people to target. The rise in digital payments has led to an increase in digital payment fraud, showing that stronger security measures are needed.[32]

6.7 Future Prediction (Predictive analysis)

Online banking fraud is a major issue, with account takeover attacks making up 55% of all fraud cases in India.[33]

Predictive Insights:

- **Behavioral Analytics:** Tracking user behavior can help detect suspicious activities and prevent fraud before it happens.
- **Predictive Modeling:** Banks can use predictive analytics to assess risk and stop fraud before it occurs by identifying high-risk transactions.[34]

6.8 Online banking Fraud Detection

Now we are going to use some models to detect these types of frauds. We chose Kaggle Dataset[25] with name **Online Payments Fraud Detection Dataset for detection.**

```
RangeIndex: 736848 entries, 0 to 736847
Data columns (total 11 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            736848 non-null  int64
 1   type            736848 non-null  object
 2   amount          736848 non-null  float64
 3   nameOrig        736847 non-null  object
 4   oldbalanceOrg   736847 non-null  float64
 5   newbalanceOrig  736847 non-null  float64
 6   nameDest        736847 non-null  object
 7   oldbalanceDest  736847 non-null  float64
 8   newbalanceDest  736847 non-null  float64
 9   isFraud         736847 non-null  float64
 10  isFlaggedFraud  736847 non-null  float64
dtypes: float64(7), int64(1), object(3)
memory usage: 61.8+ MB
```

The Data Frame has a total of 736,848 rows, with indexes ranging from 0 to 736,847. The "Range Index" is the default index assigned to the Data Frame.

- **Data columns (total eleven columns):**
  - The Data Frame contains **eleven columns** in total.

Column Breakdown:

Each row in the summary describes one column in the Data Frame:

1. **Step (int64):**
   - 736,848 non-null values, meaning no missing data. It is an integer column.
2. **Type (object):**
   - 736,848 non-null values, meaning no missing data. This column likely contains text or categorical data (strings or mixed types).
3. **Amount (float64):**
   - 736,848 non-null values, meaning no missing data. This column contains decimal numbers (floats).
4. **NameOrig (object):**
   - 736,847 non-null values. There is one missing value. It is a text column (object type).
5. **OldbalanceOrg (float64):**
   - 736,847 non-null values. There is one missing value. This column contains decimal numbers (floats).
6. **NewbalanceOrig (float64):**
   - 736,847 non-null values. There is one missing value. This column also contains decimal numbers (floats).
7. **NameDest (object):**

- 736,847 non-null values. There is one missing value. It is a text column.

8. **OldbalanceDest (float64):**
    - 736,847 non-null values. There is one missing value. This column contains decimal numbers (floats).
9. **NewbalanceDest (float64):**
    - 736,847 non-null values. There is one missing value. This column contains decimal numbers (floats).
10. **isFraud (float64):**
    - 736,847 non-null values. There is one missing value. This column contains decimal numbers (floats).
11. **isFlaggedFraud (float64):**
    - 736,847 non-null values. There is one missing value. This column contains decimal numbers (floats).

Data Types Summary:

- **Seven columns** have **float64** data type, meaning they contain decimal values.
- **One column** has **int64** data type, meaning it contains integer values.
- **Three columns** have **object** data type, meaning they contain text or mixed types (such as names).

We choose **logistic regression** model and **decision tree classifier** for given dataset. Because Both models are use for the categorical and binary (0,1) (Yes, No) analysis.

Here are our models: -

```
[ ] # Logistic Regression Model
    log_model = LogisticRegression()
    log_model.fit(X_train_scaled, y_train)

    y_pred_log = log_model.predict(X_test_scaled)

    # Evaluate Logistic Regression
    print(" 🔹 Logistic Regression Evaluation:")
    print("Accuracy:", accuracy_score(y_test, y_pred_log))
    print("Classification Report:\n", classification_report(y_test, y_pred_log))
```

```
🔹 Logistic Regression Evaluation:
Accuracy: 0.9994367917486598
Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    147284
         1.0       1.00      0.03      0.07        86

    accuracy                           1.00    147370
   macro avg       1.00      0.52      0.53    147370
weighted avg       1.00      1.00      1.00    147370
```

```
[ ] # Decision Tree Classifier
    tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
    tree_model.fit(X_train, y_train)

    y_pred_tree = tree_model.predict(X_test)

    # Evaluate Decision Tree
    print(" 🔹 Decision Tree Evaluation:")
    print("Accuracy:", accuracy_score(y_test, y_pred_tree))
    print("Classification Report:\n", classification_report(y_test, y_pred_tree))
```

```
🔹 Decision Tree Evaluation:
Accuracy: 0.9994367917486598
Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    147284
         1.0       0.62      0.09      0.16        86

    accuracy                           1.00    147370
   macro avg       0.81      0.55      0.58    147370
weighted avg       1.00      1.00      1.00    147370
```

The provided text shows the classification reports for two machine learning models: Logistic Regression and a Decision Tree Classifier. Each report evaluates the model's performance on a binary classification task (likely identifying fraudulent vs. non-fraudulent transactions).

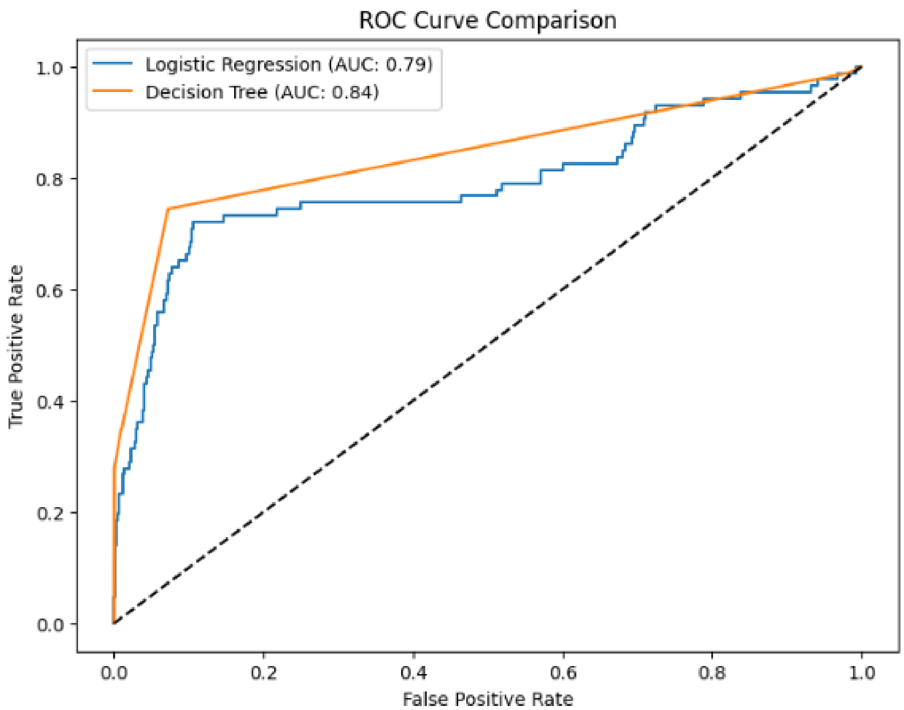
Logistic Regression Report:

- Precision: For class 0.0, it's 1.00; for class 1.0, it's 1.00.
- Recall: For class 0.0, it's 1.00; for class 1.0, it's 0.03.
- F1-score: For class 0.0, it's 1.00; for class 1.0, it's 0.07.
- Support: Class 0.0 has 147284 instances; class 1.0 has eighty-six instances.
- Accuracy: 1.00
- Macro avg: Precision 1.00, Recall 0.52, F1-score 0.53.
- Weighted avg: Precision 1.00, Recall 1.00, F1-score 1.00.

Decision Tree Classifier Report:

- Precision: For class 0.0, it's 1.00; for class 1.0, it's 0.62.
- Recall: For class 0.0, it's 1.00; for class 1.0, it's 0.09.
- F1-score: For class 0.0, it's 1.00; for class 1.0, it's 0.16.
- Support: Class 0.0 has 147284 instances; class 1.0 has eighty-six instances.
- Accuracy: 1.00
- Macro avg: Precision 0.81, Recall 0.55, F1-score 0.58.
- Weighted avg: Precision 1.00, Recall 1.00, F1-score 1.00.

```python
# ROC Curve & Model Comparison
fpr_log, tpr_log, _ = roc_curve(y_test, log_model.predict_proba(X_test_scaled)[:,1])
fpr_tree, tpr_tree, _ = roc_curve(y_test, tree_model.predict_proba(X_test)[:,1])

plt.figure(figsize=(8,6))
plt.plot(fpr_log, tpr_log, label="Logistic Regression (AUC: {:.2f})".format(auc(fpr_log, tpr_log)))
plt.plot(fpr_tree, tpr_tree, label="Decision Tree (AUC: {:.2f})".format(auc(fpr_tree, tpr_tree)))
plt.plot([0,1], [0,1], 'k--')  # Random classifier line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.show()
```



ROC Curve Plot Explanation:

The resulting plot includes two ROC curves:

- **Blue Curve (Logistic Regression):** The AUC for this curve is approximately 0.79. It shows the trade-off between false positives and true positives for Logistic Regression at different thresholds.
- **Dark Orange Curve (Decision Tree):** The AUC for this curve is approximately 0.84. It illustrates the Decision Tree model's performance across different thresholds.

- **Dashed Black Line:** Represents a random classifier (AUC = 0.5), indicating random performance.

Interpretation:

The ROC curves provide a visual comparison of the models' ability to distinguish between positive and negative classes. A curve closer to the top-left corner indicates better performance. The AUC value quantifies this ability, with a higher AUC indicating better model performance.

In this case:

- The **Decision Tree model** (AUC ≈ 0.84) demonstrates slightly superior discrimination between the classes compared to **Logistic Regression** (AUC ≈ 0.79), indicating that the Decision Tree maintains a better balance between the true positive and false positive rates.

7. Prescriptive Analysis for Preventive Measures and Solutions

Prescriptive analysis focuses on recommending specific actions to mitigate cybercrime risks, based on insights derived from descriptive, diagnostic, and predictive analyses. Given the increasing threat of malware attacks, credit/debit card fraud, and online banking fraud in India, the following preventive measures and solutions are proposed:

1. Enhancing User Awareness and Education

- **Training and Awareness Programs:** A crucial measure is educating individuals about cybersecurity best practices. Users must be trained to recognize phishing emails, avoid suspicious links, and understand the importance of strong, unique passwords. Continuous awareness campaigns can significantly reduce the likelihood of cyberattacks.

- **Promoting Multi-Factor Authentication (MFA):** Encouraging users to enable MFA for online banking and financial accounts strengthens security by adding another layer of protection, making unauthorized access more difficult.

2. Strengthening Financial Institution Security

- **Advanced Fraud Detection Systems:** Banks and financial institutions should implement machine learning-based fraud detection systems that analyze user behavior and flag any unusual activities in real-time, such as abnormal spending patterns or location anomalies.
- **Encryption of Transactions:** Ensuring that all financial transactions are encrypted using up-to-date cryptographic standards (e.g., SSL/TLS) is crucial for preventing data interception by cybercriminals. Secure communication channels should be a norm for all financial institutions.
- **Real-Time Alerts and Monitoring:** Implementing real-time alerts for users when transactions, particularly high-value or international transfers, occur can provide immediate notifications of suspicious activities. Alerts could include confirmation messages or prompts for immediate action if the transaction appears unusual.

3. Improved Malware Protection

- **Regular Software Updates and Patches:** Regular updates for operating systems, software, and applications are vital in closing security vulnerabilities that malware can exploit. Timely software updates should be prioritized by both organizations and individuals.
- **Anti-Malware Software:** Installing and maintaining robust anti-malware solutions on all devices helps prevent, detect, and remove malware. These tools should be frequently updated to combat new malware strains effectively.

- **Firewalls and Intrusion Detection Systems (IDS):** Firewalls and IDS should be used to monitor and block malicious network traffic that could carry malware. Organizations must deploy these as part of a comprehensive cybersecurity defense.

4. Strengthening Regulatory and Legal Measures

- **Strict Enforcement of Cybersecurity Laws:** Governments should enforce stricter cybersecurity regulations to ensure businesses comply with data protection, fraud prevention, and incident reporting standards. Penalizing organizations that fail to meet security requirements can motivate better cybersecurity practices.
- **Collaboration Between Stakeholders:** Encouraging collaboration among government agencies, financial institutions, cybersecurity firms, and the public is crucial to forming a unified defense against cybercrime. Sharing threat intelligence can help in quicker identification of emerging risks.

5. Adoption of Secure Payment Technologies

- **Tokenization and Encryption for Card Transactions:** Tokenization replaces sensitive card data with a unique token, reducing the exposure of real card information. Payment systems should adopt this technology to protect consumers from fraud in case of data breaches.
- **Biometric Authentication:** Financial institutions should adopt biometric authentication methods (e.g., fingerprint scanning, facial recognition) to verify users during financial transactions. This makes it harder for attackers to impersonate legitimate users.

6. Collaboration with Cybersecurity Professionals

- **Cybersecurity Audits and Assessments:** Regular security audits and vulnerability assessments by third-party cybersecurity professionals can help detect weaknesses before cybercriminals exploit them. These audits should include both internal infrastructure and third-party services.
- **Incident Response Plans:** Developing and maintaining an effective incident response plan is essential to respond quickly and efficiently when cyberattacks occur. The plan should include steps for mitigating damage, notifying affected parties, and recovering lost data.

7. Government Initiatives for Cybercrime Prevention

- **Public Awareness Campaigns on Cybercrime:** The government should run campaigns to educate the public about common cyber threats and best practices for online security. Awareness programs for individuals and businesses can significantly reduce the chances of cybercrime.
- **Cybersecurity Research and Development (R&D):** Investment in cybersecurity R&D is crucial to developing innovative solutions to counter evolving cybercrime techniques. Collaboration between the private sector, academia, and the government will foster the creation of effective tools to combat cybercrime.

8. Continuous Monitoring and Feedback Loop

- **Continuous Monitoring Systems:** Businesses, especially in sectors like banking and payments, should implement 24/7 monitoring of critical infrastructures. Real-time data feeds and automated responses can identify and neutralize threats before they cause significant harm.
- **Data Analytics for Threat Intelligence:** Predictive analytics and machine learning models should be employed to analyze cybercrime

trends and anticipate future attacks. Cybersecurity teams should use threat intelligence sources to predict and proactively block cybercriminal activities.

8. Conclusion

This project highlights the increasing threat of cybercrime in India, especially in areas like malware attacks, credit/debit card fraud, and online banking fraud. By analyzing data and using machine learning models, we have successfully detected patterns and predicted fraud risks. The findings show the importance of data-driven decisions in fighting cybercrime and offer useful insights for individuals, businesses, and authorities to improve cybersecurity.

The results of this study can help shape policies, enhance security systems, and raise awareness about online risks. As the digital world continues to grow, it's important to keep monitoring and updating machine learning models to stay ahead of new threats and reduce the impact of cybercrime on society.

9. References

[1] https://www.britannica.com/topic/cybercrime

[2] ***Steve Morgan**, "Special Report: Cyberwarfare In The C-Suite."13th November 2020*.

[3] https://cybertalents.com/blog/what-is-cyber-crime-types-examples-and-prevention

[4] https://www.statista.com/topics/5054/cyber-crime-in-india/#topicOverview

[5] https://www.statista.com/statistics/1499739/india-cyber-crime-cases-reported-to-i4c/#:~:text=Over%20740%2C000%20cases%20of%20cyber,related%20to%20online%20financial%20fraud.

[6] https://www.business-standard.com/india-news/here-is-how-much-indians-lost-to-cyber-frauds-between-jan-and-apr-of-2024-124052700151_1.html

[7] https://www.kaggle.com/datasets/sateeshkumar6289/cicids-2017-dataset

[8] https://www.kaggle.com/datasets/teamincribo/cyber-security-attacks

[9] https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

[10] https://techcrunch.com/2023/06/15/lockbit-ransomware-granules-india/

[11] https://www.fortinet.com/blog/threat-research/ransomware-roundup-akira

[12] https://cio.economictimes.indiatimes.com/news/digital-security/combating-cyber-fraud-threats-for-success-of-the-india-digital-growth-story/116238461

[13] https://www.financialexpress.com/business/digital-transformation-nearly-60-indian-enterprises-fall-below-cybersecurity-poverty-line-study-3660693/

[14] https://www.proofpoint.com/us/threat-reference/social-engineering

[15] https://kratikal.com/blog/278-percent-rise-of-state-sponsored-cyberattacks-in-india/

[16] https://economictimes.indiatimes.com/tech/technology/india-tops-list-of-global-malware-attacks-zscaler-report/articleshow/115935130.cms

[17] https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023

[18] https://www.data.gov.in/resource/stateut-wise-number-cases-registered-under-fraud-cyber-crimes-such-creditdebit-cards-atms

[19] https://en.wikipedia.org/wiki/2016_Indian_bank_data_breach

[20] https://en.wikipedia.org/wiki/Data_breaches_in_India

[21] https://www.reuters.com/world/india/india-says-cyber-fraud-cases-jumped-over-four-fold-fy2024-caused-20-mln-losses-2025-03-11/

[22] https://en.wikipedia.org/wiki/Credit_card_fraud

[23] https://www.the-sun.com/news/13818001/info-stealing-skimmer-piggly-wiggly-store-device-found/

[24] https://www.financialexpress.com/business/banking-finance-role-of-data-analytics-in-fraud-prevention-for-fintech-3600113/

[25] https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

[26] https://www.reuters.com/world/india/india-says-cyber-fraud-cases-jumped-over-four-fold-fy2024-caused-20-mln-losses-2025-03-11/

[27] https://en.wikipedia.org/wiki/2016_Indian_bank_data_breach

[28] https://timesofindia.indiatimes.com/business/india-business/beware-800-online-financial-frauds-a-day/articleshow/111241765.cms

[29] https://www.reuters.com/technology/india-central-bank-governor-cautions-lenders-against-rising-digital-frauds-2025-02-07/

[30] https://www.fraud.com/post/increase-in-digital-banking-fraud

[31] https://timesofindia.indiatimes.com/technology/tech-news/rbi-data-suggests-rise-in-online-payment-frauds-in-india-reasons-and-what-users-should-do/articleshow/110627011.cms

[32] https://thesecretariat.in/article/rise-in-online-fraud-a-challenge-for-india-s-digital-economy

[33] https://www.biocatch.com/press-release/biocatch-releases-2024-digital-banking-fraud-trends-report-in-india

[34] https://www.crif.in/blog/predictive-analytics/the-power-of-predictive-analytics-in-banking