Fastag-fraud-detection

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_score,
from imblearn.over_sampling import SMOTE
```

1. Project Overview

This project Focuses on developing a robust machine learning- based fraud detection system for fastag transactions. Fastag in an electronic toll collection system in indian that uses RFID technology to make toll payments directly from a prepaid account linked to a user's vehicle. as digital transactions become machine learning classification techniques to accurately also increases. This Project aims to leverage machine learning classification techniques to accurately identify fraudulent transactions, thereby ensuring the security and integrity of fastag Transactions.

In [2]:
```
df = pd.read_csv(r'C:\Users\chira\Downloads\FastagFraudDetection.csv')
df
```

Out[2]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensi
0	1	01-06-2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	L
1	2	01-07-2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	S
2	3	01-08-2023 18:25	Motorcycle	NaN	D-104	Regular	S
3	4	01-09-2023 02:05	Truck	FTG-044-LMN-322	C-103	Regular	L
4	5	01-10-2023 06:35	Van	FTG-505-DEF-652	B-102	Express	Mec
...	...	...	...	...	...	...	
4995	4996	01-01-2023 22:18	Truck	FTG-445-EDC-765	C-103	Regular	L
4996	4997	1/17/2023 13:43	Van	FTG-446-LMK-432	B-102	Express	Mec
4997	4998	02-05-2023 05:08	Sedan	FTG-447-PLN-109	A-101	Regular	Mec
4998	4999	2/20/2023 20:34	SUV	FTG-458-VFR-876	B-102	Express	L
4999	5000	03-10-2023 00:59	Bus	FTG-459-WSX-543	C-103	Regular	L

5000 rows × 13 columns

2.Dataset Description

The Dataset Comprises Various Features related to fastag Transcation, including transactiondetails, vehicle information, geographical location, and transcation amounts. The key features are:

Transcation_ID : Unique Identifier fro each transaction.

Timestamp: Date and time of the transaction.

Vehicle_Type: Type of vehicle involved in the transaction.

FastagID: Unique identifier for Fastag.

TollBoothID: Identifier for the toll booth.

Lane_Type: Type of lane used for the transaction.

Vehicle_Dimensions: Dimensions of the vehicle.

Transaction_Amount: Amount associated with the transaction.

Amount_paid: Amount paid for the transaction.

Geographical_Location: Location details of the transaction.

Vehicle_Speed: Speed of the vehicle during the transaction.

Vehicle_Plate_Number: License plate number of the vehicle.

Fraud_indicator: Binary indicator of fraudulent activity (target variable).

Timestamp: Date and time of the transaction.

Vehicle_Type: Type of vehicle involved in the transaction.

FastagID: Unique identifier for Fastag.

TollBoothID: Identifier for the toll booth.

Lane_Type: Type of lane used for the transaction.

Vehicle_Dimensions: Dimensions of the vehicle.

Transaction_Amount: Amount associated with the transaction.

Amount_paid: Amount paid for the transaction.

Geographical_Location: Location details of the transaction.

Vehicle_Speed: Speed of the vehicle during the transaction.

Vehicle_Plate_Number: License plate number of the vehicle.

Fraud_indicator: Binary indicator of fraudulent activity (target variable).

In [3]:
```
df.head()
```

Out[3]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions
0	1	01-06-2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large
1	2	01-07-2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small
2	3	01-08-2023 18:25	Motorcycle	NaN	D-104	Regular	Small
3	4	01-09-2023 02:05	Truck	FTG-044-LMN-322	C-103	Regular	Large
4	5	01-10-2023 06:35	Van	FTG-505-DEF-652	B-102	Express	Medium

In [4]:
```
df.tail()
```

Out[4]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensi
4995	4996	01-01-2023 22:18	Truck	FTG-445-EDC-765	C-103	Regular	L
4996	4997	1/17/2023 13:43	Van	FTG-446-LMK-432	B-102	Express	Med
4997	4998	02-05-2023 05:08	Sedan	FTG-447-PLN-109	A-101	Regular	Med
4998	4999	2/20/2023 20:34	SUV	FTG-458-VFR-876	B-102	Express	L
4999	5000	03-10-2023 00:59	Bus	FTG-459-WSX-543	C-103	Regular	L

In [5]:

```
df.sample(5)
```

Out[5]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensi
4119	4120	3/29/2023 14:30	Motorcycle	NaN	D-106	Regular	S
2979	2980	11/22/2023 0:58	Truck	FTG-480-TGB-250	C-103	Regular	L
642	643	04-08-2023 13:55	Sedan	FTG-823-NMK-365	A-101	Express	Med
2644	2645	02-04-2023 06:25	Van	FTG-125-DCF-765	B-102	Express	Med
2036	2037	9/13/2023 4:05	SUV	FTG-455-QRS-789	B-102	Express	L

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Transaction_ID        5000 non-null   int64 
 1   Timestamp             5000 non-null   object
 2   Vehicle_Type          5000 non-null   object
 3   FastagID              4451 non-null   object
 4   TollBoothID           5000 non-null   object
 5   Lane_Type             5000 non-null   object
 6   Vehicle_Dimensions    5000 non-null   object
 7   Transaction_Amount    5000 non-null   int64 
 8   Amount_paid           5000 non-null   int64 
 9   Geographical_Location  5000 non-null   object
 10  Vehicle_Speed         5000 non-null   int64 
 11  Vehicle_Plate_Number  5000 non-null   object
 12  Fraud_indicator       5000 non-null   object
dtypes: int64(4), object(9)
memory usage: 507.9+ KB
```

Summary Statistics

We use summary statistics to get sn overview of the numerical features.

Numerical Summary Statistics:

Count: The Number Of Non-missing Values.

Mean: The Average value.

Std: The Standard Deviation, indicating the spread of the values.

min: The Minimum Value.

25%: The 25th precentike value (first quartile).

50%: The median value(second quartile).

75%: The 75th percentile value (third quartile).

max: The maximum value.

median: The median value, explicitly added for clarity.

mode: The most frequently occurring value.

missing_values: The count of missing values in each column.

In [7]:
```
df . describe()
```

Out[7]:

	Transaction_ID	Transaction_Amount	Amount_paid	Vehicle_Speed
count	5000.000000	5000.00000	5000.000000	5000.000000
mean	2500.500000	161.06200	141.261000	67.851200
std	1443.520003	112.44995	106.480996	16.597547
min	1.000000	0.00000	0.000000	10.000000
25%	1250.750000	100.00000	90.000000	54.000000
50%	2500.500000	130.00000	120.000000	67.000000
75%	3750.250000	290.00000	160.000000	82.000000
max	5000.000000	350.00000	350.000000	118.000000

In [8]:
```
df.select_dtypes("number").mean()
```

Out[8]:
```
Transaction_ID         2500.5000
Transaction_Amount      161.0620
Amount_paid             141.2610
Vehicle_Speed            67.8512
dtype: float64
```

In [9]:
```
df.select_dtypes("number").median()
```

Out[9]:
```
Transaction_ID        2500.5
Transaction_Amount     130.0
Amount_paid            120.0
Vehicle_Speed           67.0
dtype: float64
```

In [10]:
```
df.select_dtypes('number').mode().iloc[0]
```

Out[10]:
```
Transaction_ID         1.0
Transaction_Amount     0.0
Amount_paid            0.0
Vehicle_Speed         55.0
Name: 0, dtype: float64
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
Transaction_ID            0
Timestamp                 0
Vehicle_Type              0
FastagID                549
TollBoothID               0
Lane_Type                 0
Vehicle_Dimensions        0
Transaction_Amount        0
Amount_paid               0
Geographical_Location     0
Vehicle_Speed             0
Vehicle_Plate_Number      0
Fraud_indicator           0
dtype: int64
```

In [12]:
```
df["FastagID"].fillna(df["FastagID"].mode()[0], inplace=True)
```

In [13]:
```
df.isnull().sum()
```

Out[13]:
```
Transaction_ID          0
Timestamp               0
Vehicle_Type            0
FastagID                0
TollBoothID             0
Lane_Type               0
Vehicle_Dimensions      0
Transaction_Amount      0
Amount_paid             0
Geographical_Location   0
Vehicle_Speed           0
Vehicle_Plate_Number    0
Fraud_indicator         0
dtype: int64
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   Transaction_ID         5000 non-null   int64 
 1   Timestamp              5000 non-null   object
 2   Vehicle_Type           5000 non-null   object
 3   FastagID               5000 non-null   object
 4   TollBoothID            5000 non-null   object
 5   Lane_Type              5000 non-null   object
 6   Vehicle_Dimensions     5000 non-null   object
 7   Transaction_Amount     5000 non-null   int64 
 8   Amount_paid            5000 non-null   int64 
 9   Geographical_Location  5000 non-null   object
 10  Vehicle_Speed          5000 non-null   int64 
 11  Vehicle_Plate_Number   5000 non-null   object
 12  Fraud_indicator        5000 non-null   object
dtypes: int64(4), object(9)
memory usage: 507.9+ KB
```

In [15]:
```python
df.drop_duplicates(inplace = True)
df
```

Out[15]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensi
0	1	01-06-2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	L
1	2	01-07-2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	S
2	3	01-08-2023 18:25	Motorcycle	FTG-000-QAZ-210	D-104	Regular	S
3	4	01-09-2023 02:05	Truck	FTG-044-LMN-322	C-103	Regular	L
4	5	01-10-2023 06:35	Van	FTG-505-DEF-652	B-102	Express	Mec
...	...	...	...	...	...	...	
4995	4996	01-01-2023 22:18	Truck	FTG-445-EDC-765	C-103	Regular	L
4996	4997	1/17/2023 13:43	Van	FTG-446-LMK-432	B-102	Express	Mec
4997	4998	02-05-2023 05:08	Sedan	FTG-447-PLN-109	A-101	Regular	Mec
4998	4999	2/20/2023 20:34	SUV	FTG-458-VFR-876	B-102	Express	L
4999	5000	03-10-2023 00:59	Bus	FTG-459-WSX-543	C-103	Regular	L

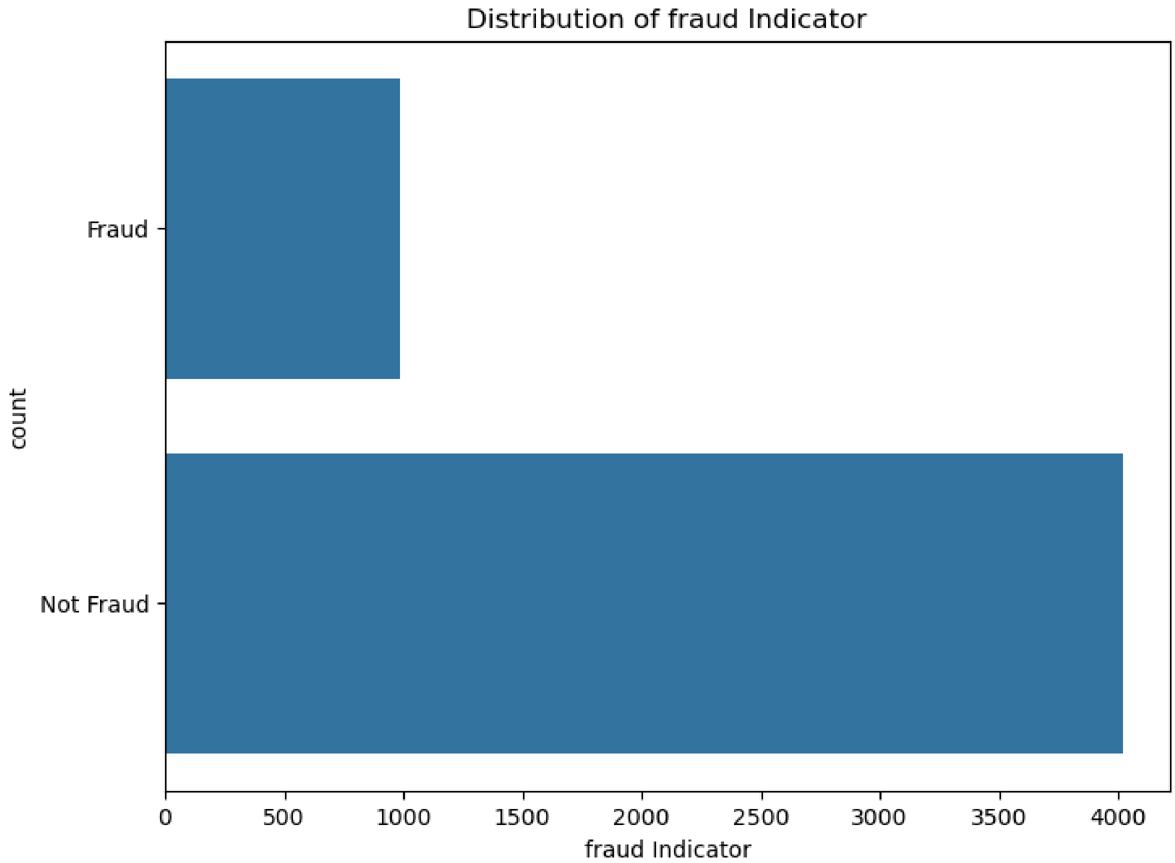
5000 rows × 13 columns

5. Data Visualization

6. Visualize the distribution of the target variable

In [16]:
```
plt.figure(figsize= (8,6))
sns.countplot(df['Fraud_indicator'])
plt.title('Distribution of fraud Indicator')
plt.xlabel("fraud Indicator")
plt.ylabel("count")
plt.show
```

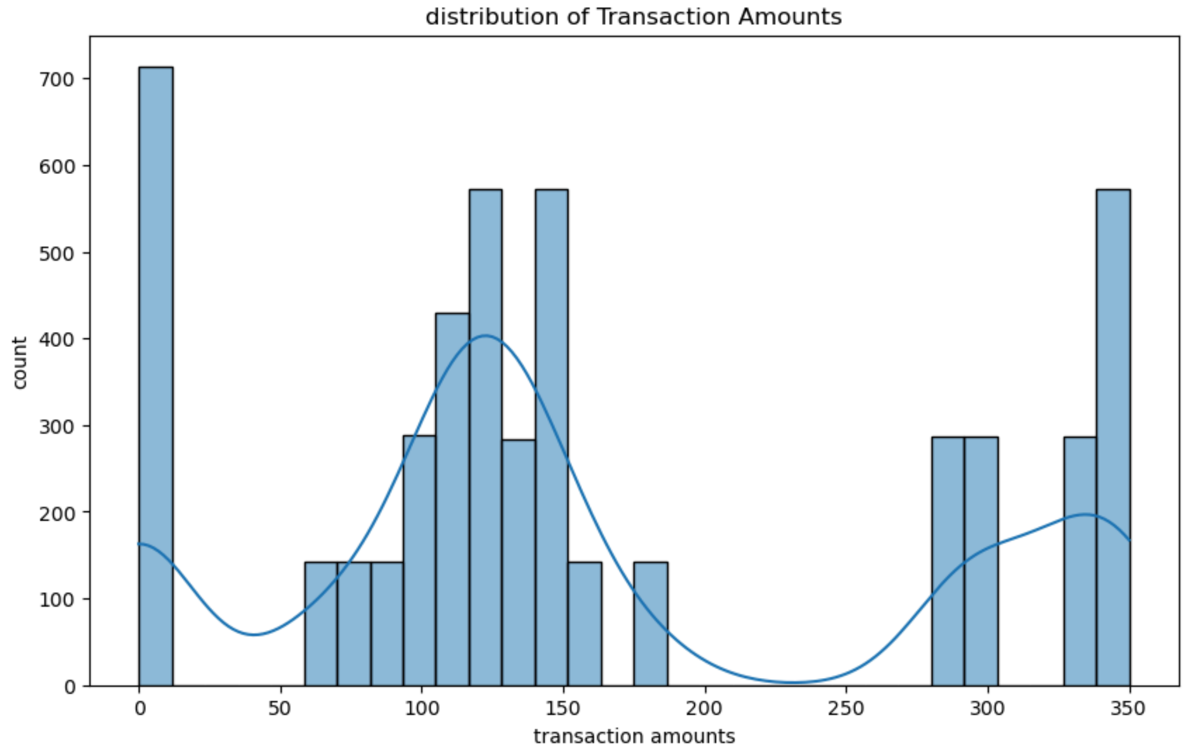
Out[16]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



7. Visualize the Distribution of Transaction Amounts

In [17]:
```
plt.figure(figsize=(10,6))
sns.histplot(df['Transaction_Amount'], bins=30 ,kde = True)
plt.title("distribution of Transaction Amounts")
plt.xlabel("transaction amounts")
plt.ylabel("count")
plt.show
```

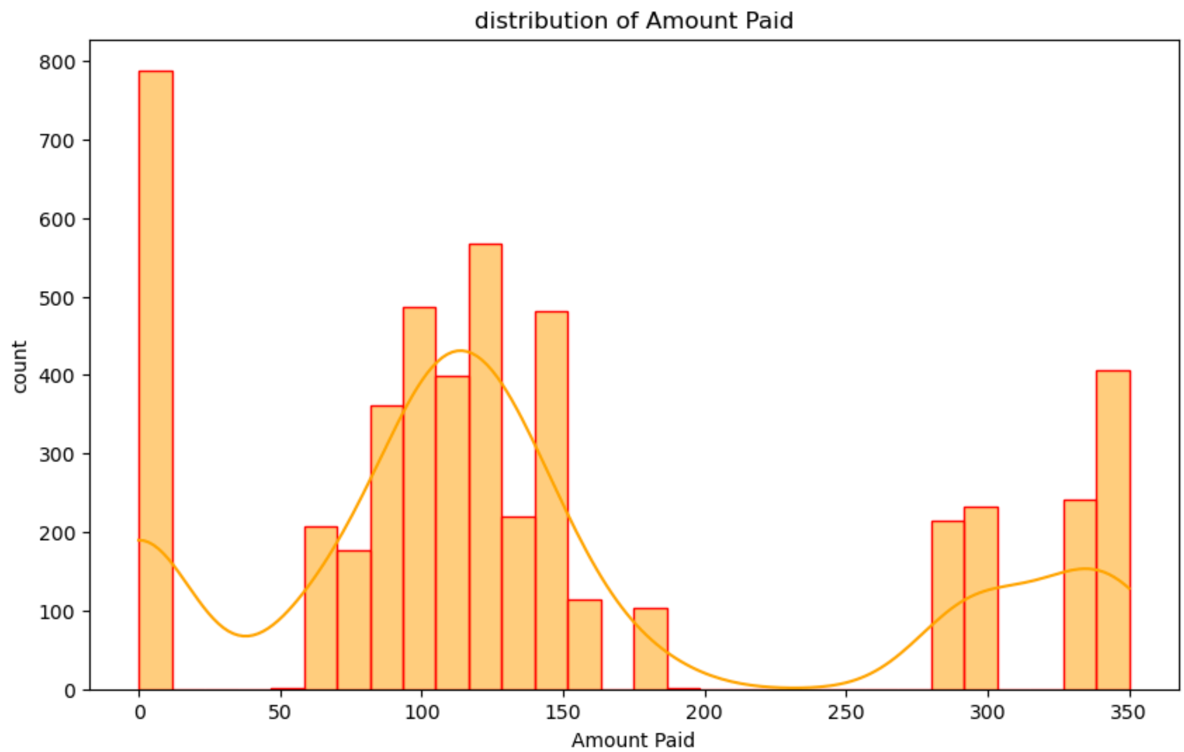
Out[17]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



8. Visualize the Distribution of Amount Paid

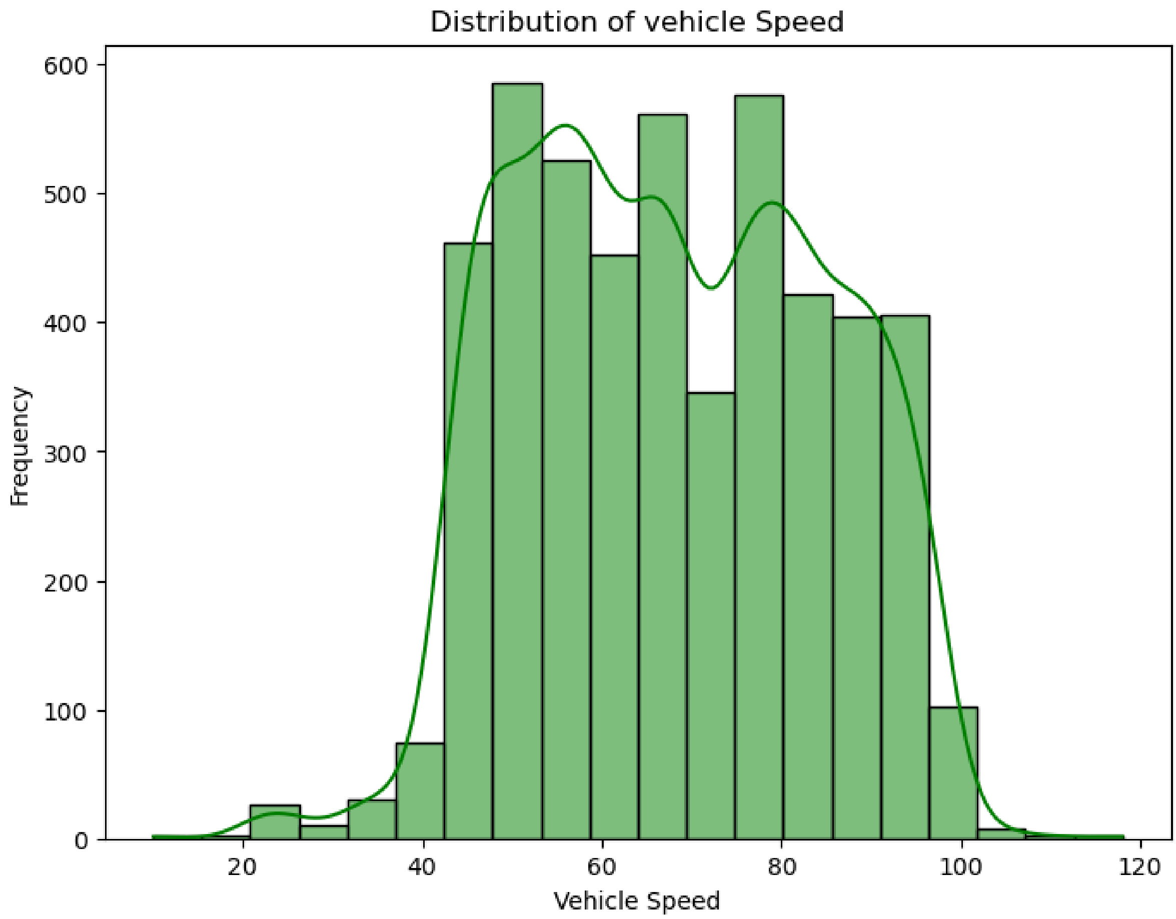
In [18]:

```
plt.figure(figsize=(10,6))
sns.histplot(df['Amount_paid'], bins=30, kde = True, color="orange", edgecolor="rec
plt.title('distribution of Amount Paid')
plt.xlabel("Amount Paid")
plt.ylabel("count")
plt.show()
```



9. Visualize the distribution of Vechicle Speed

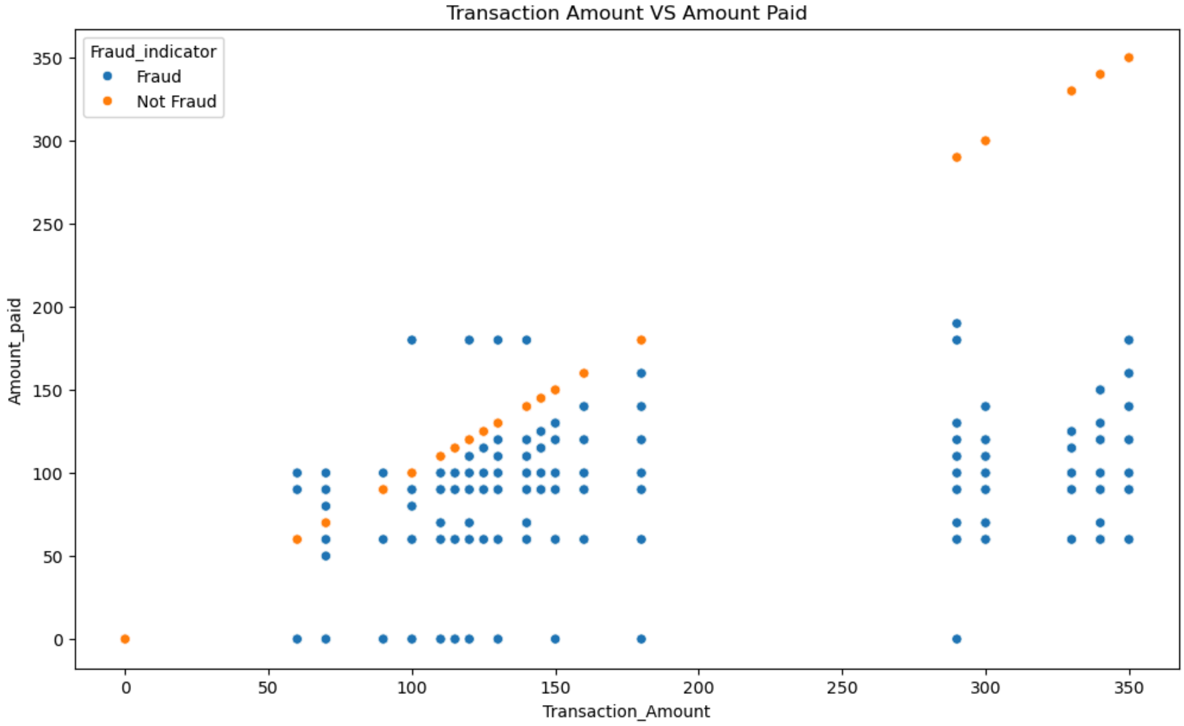
In [19]:
```
plt.figure(figsize=(8,6))
sns.histplot(df['Vehicle_Speed'],bins=20, kde=True, color="green", edgecolor="black
plt.title("Distribution of vehicle Speed")
plt.xlabel("Vehicle Speed")
plt.ylabel("Frequency")
plt.show()
```



10. Visualize the relationship Between Transcation Amount And Amount paid

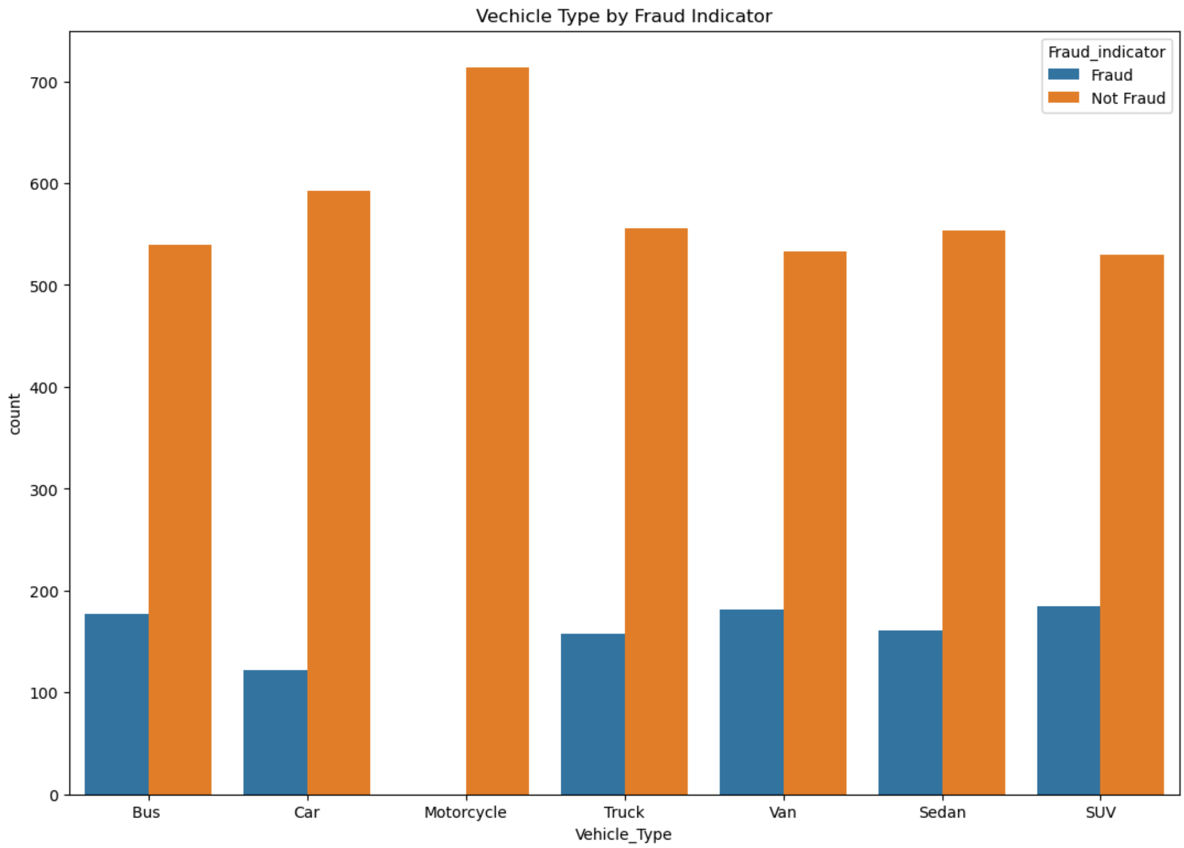
In [20]:
```
plt.figure(figsize=(12,7))
sns.scatterplot(x="Transaction_Amount", y = "Amount_paid", hue= "Fraud_indicator",
plt.title("Transaction Amount VS Amount Paid")
```

Out[20]:
```
Text(0.5, 1.0, 'Transaction Amount VS Amount Paid')
```



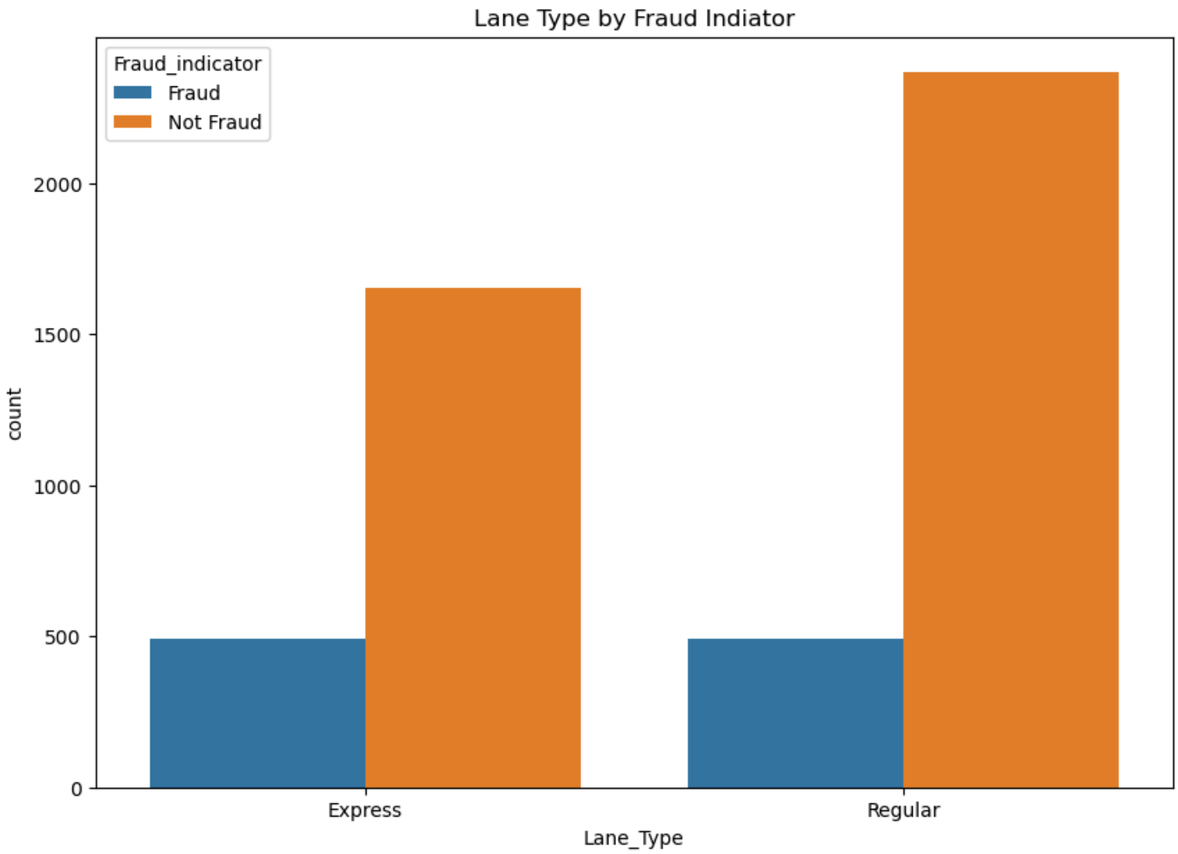
11. Countplots For Vehicle type variables by Fraud indicator

In [21]:
```
plt.figure(figsize=(13,9))
sns.countplot(x="Vehicle_Type",hue= "Fraud_indicator", data=df)
plt.title("Vechicle Type by Fraud Indicator")
plt.show()
```



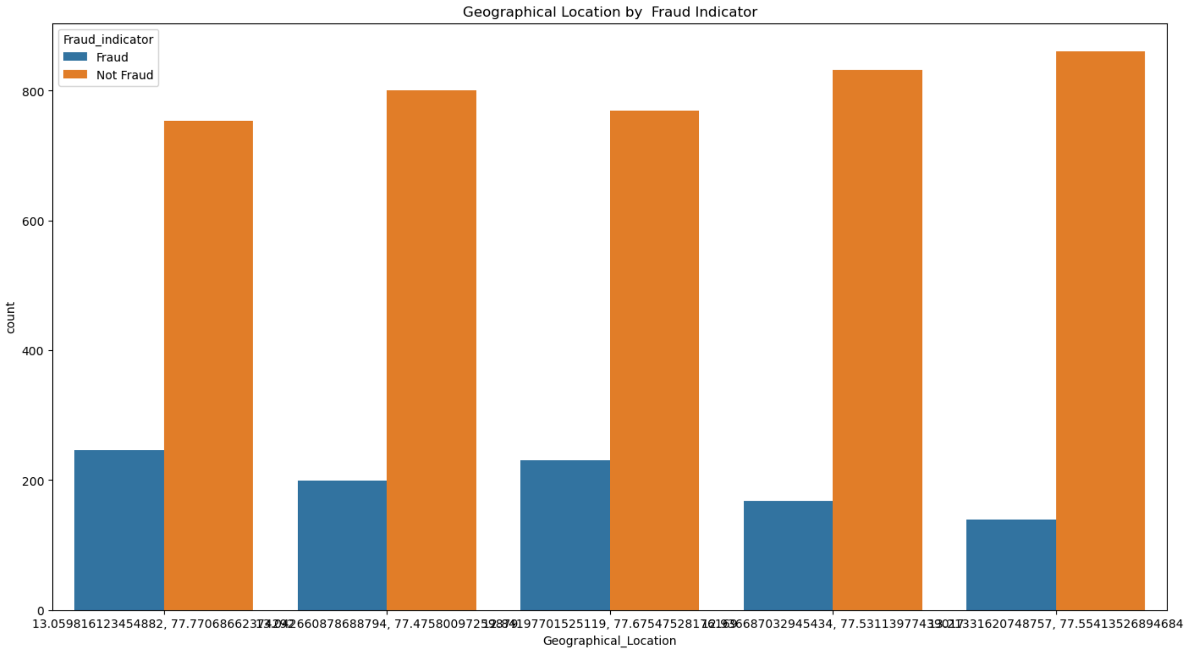
12. Countplot for Lane Type variables by fraud Indicator

In [22]:
```
plt.figure(figsize=(10,7))
sns.countplot(x="Lane_Type", hue="Fraud_indicator", data=df)
plt.title("Lane Type by Fraud Indiator")
plt.show()
```



13. Countplots for Geographical Loation Variables by Fraud indicator

In [23]:
```
plt.figure(figsize=(17,9))
sns.countplot(x="Geographical_Location", hue="Fraud_indicator", data=df)
plt.title("Geographical Location by  Fraud Indicator")
plt.show()
```

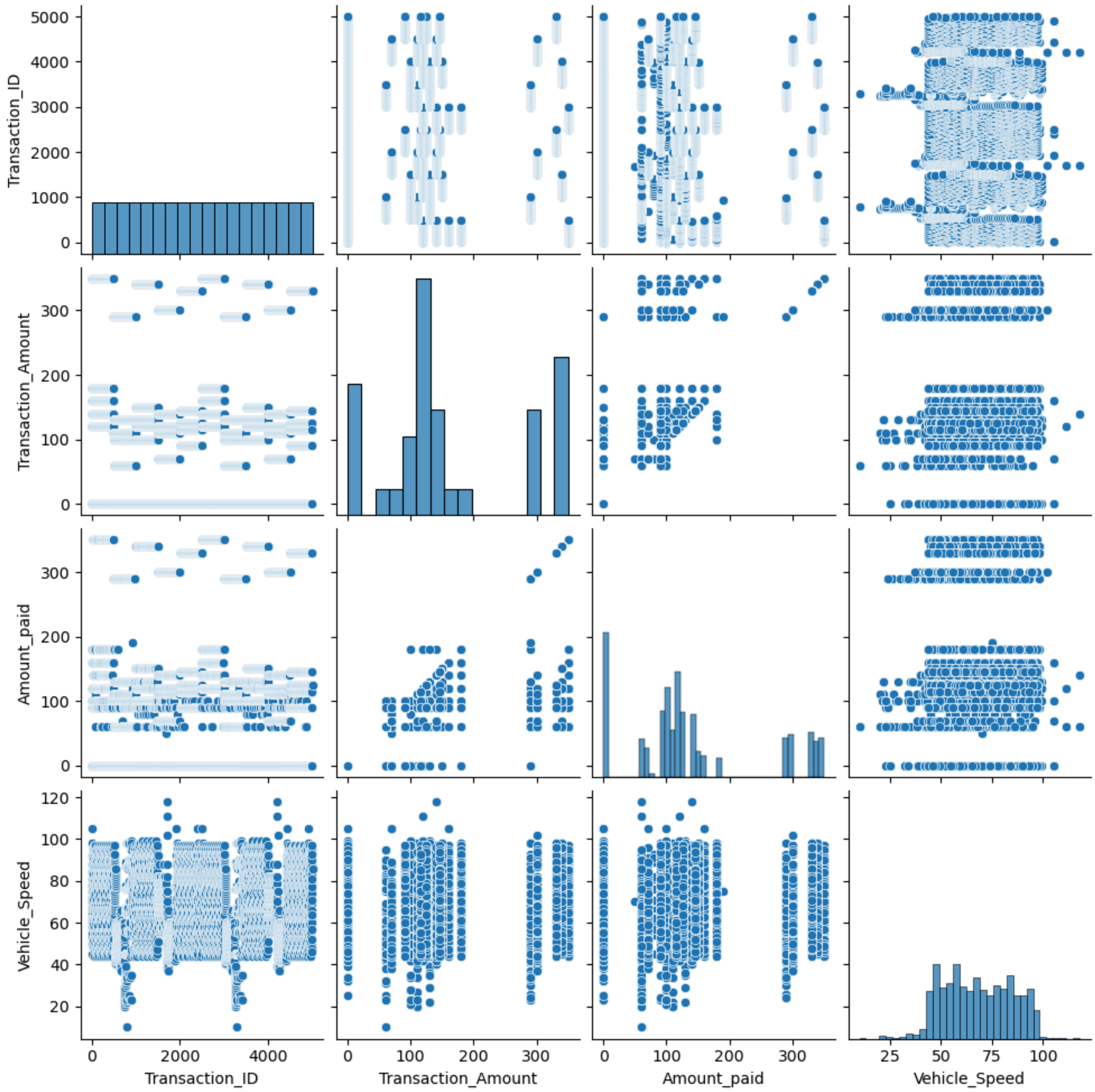


14. Check the Pair plot

In [24]:
```python
sns.pairplot(df)
```

```
C:\Users\chira\anaconda3\Lib\site-packages\seaborn\axisgrid.py:123: UserWarning: T
he figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[24]:
```
<seaborn.axisgrid.PairGrid at 0x1bc7bca3ed0>
```



15. Heat Map

In [25]:
```
df1 = df.select_dtypes('number')
df1
```

Out[25]:

	Transaction_ID	Transaction_Amount	Amount_paid	Vehicle_Speed
0	1	350	120	65
1	2	120	100	78
2	3	0	0	53
3	4	350	120	92
4	5	140	100	60
...	...	...	...	...
4995	4996	330	330	81
4996	4997	125	125	64
4997	4998	115	115	93
4998	4999	145	145	57
4999	5000	330	125	86

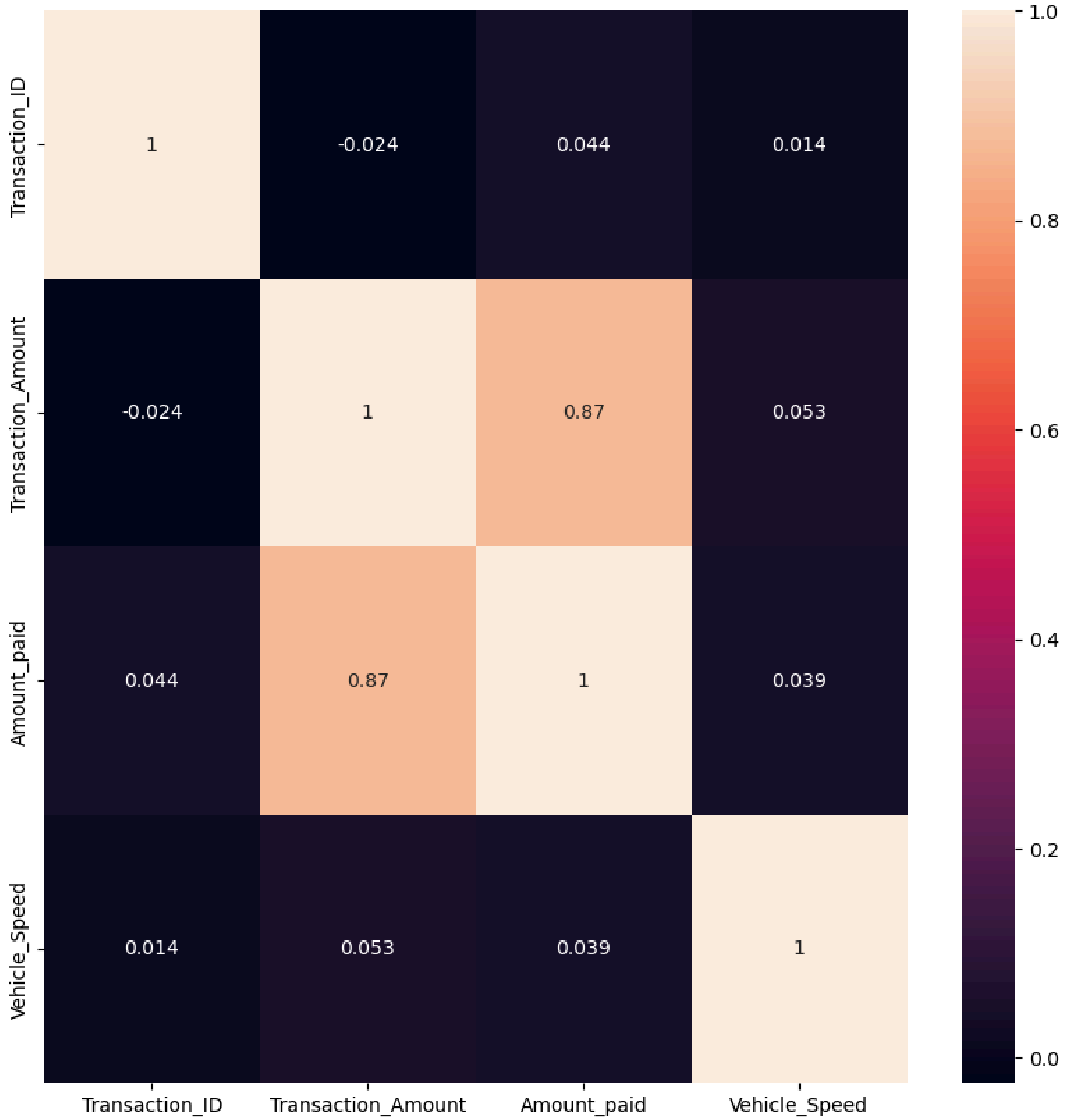
5000 rows × 4 columns

In [26]:
```
df1.corr()
```

Out[26]:

	Transaction_ID	Transaction_Amount	Amount_paid	Vehicle_Speed
Transaction_ID	1.000000	-0.023515	0.044433	0.014378
Transaction_Amount	-0.023515	1.000000	0.870078	0.053229
Amount_paid	0.044433	0.870078	1.000000	0.039027
Vehicle_Speed	0.014378	0.053229	0.039027	1.000000

In [27]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr() ,annot = True)
plt.show()
```



16. Label Encoding for Categorical Features: Ensure that Categorical Features are Encoded.

In [28]:

```
label_encoders = {}
for column in ["Vehicle_Type", "Lane_Type", "Vehicle_Dimensions", "Geographical_Loc
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column]=le
```

In [29]:

```
df.head(3)
```

Out[29]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions
0	1	01-06-2023 11:20	0	2	0	0	C
1	2	01-07-2023 14:55	1	9	1	1	2
2	3	01-08-2023 18:25	2	0	3	1	2

17. Feature Extraction: Additional time-based featues(Hour, Day, Month, Weekday) have been extracted from the Times-tamp

In [30]:

```
# Covert Timestamp to datetime and extract new features
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df['Hour'] = df['Timestamp'].dt.hour
df['Day'] = df['Timestamp'].dt.day
df['Month'] = df['Timestamp'].dt.month
df['Weekday'] = df['Timestamp'].dt.weekday
df.sample(6)
```

Out[30]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensi
1131	1132	2023-06-18 07:30:00	6	1435	1	1	
118	119	2023-01-07 14:55:00	3	778	1	0	
2301	2302	2023-06-18 10:30:00	4	1729	0	1	
2913	2914	2023-11-06 15:38:00	0	2421	2	1	
953	954	2023-07-02 18:18:00	1	3513	0	1	
3136	3137	2023-09-21 01:33:00	3	3926	1	0	

In [32]:

```
# Drop the original timestamp coplumn and trransaction_ID as it is not informative
df.drop(columns=['Timestamp', "Transaction_ID"], inplace=True)
```

In [33]:

```
df.head(5)
```

Out[33]:

	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amo
0	0	2	0	0	0	350	
1	1	9	1	1	2	120	
2	2	0	3	1	2	0	
3	5	241	2	1	0	350	
4	6	2860	1	0	1	140	

17.1 Model Training: Train A Variety of Machine Learning Models(e.g, Logistic Regression, Random Forest, Gradient Boosting).

17.2 Model Evaluation: Evalute model Performance using metrics such as precision, recall, F1 score, and Accuray. Additionally, handle class imbalance using techniques such as SMOTE(Synthenic Minority Over-smapling Technique)if necessary

In [35]:

```
x = df.drop(columns=['Fraud_indicator'])
y = df['Fraud_indicator']

#Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,random_state
x_train, x_test, y_train, y_test
```

Out[35]:

```
(      Vehicle_Type  FastagID  TollBoothID  Lane_Type  Vehicle_Dimensions  \
 4227             4       155            0          1                   1
 4676             3      1028            1          0                   0
 800              2         0            5          1                   2
 3671             2         0            5          1                   2
 4193             3      1369            1          1                   0
 ...            ...       ...          ...        ...                 ...
 4426             1      2434            0          0                   2
 466              6      3232            1          0                   1
 3092             6      3449            1          0                   1
 3772             4       716            0          0                   1
 860              3      3585            1          0                   0

       Transaction_Amount  Amount_paid  Geographical_Location  Vehicle_Speed  \
 4227                 110          110                      1             44
 4676                 145          145                      4             61
 800                    0            0                      2             45
 3671                   0            0                      0             96
 4193                 140          140                      1             74
 ...                  ...          ...                    ...            ...
 4426                  70           70                      1             67
 466                  140          140                      3             61
 3092                 110          110                      2             52
 3772                 120          100                      0             67
 860                  130          130                      2             85

       Vehicle_Plate_Number  Hour  Day  Month  Weekday
 4227                  2214     1   21      3        1
 4676                  1207    20    8      2        2
 800                   1663     8    6      8        6
 3671                  2208    10   14      9        3
 4193                   568     5    3      9        6
 ...                    ...   ...  ...    ...      ...
 4426                  4616     0   25      6        6
 466                   1425     3    3      4        0
 3092                  4193     3   23     11        3
 3772                    11     8   27      4        3
 860                   4056     8   31      5        2

 [4000 rows x 14 columns],
       Vehicle_Type  FastagID  TollBoothID  Lane_Type  Vehicle_Dimensions  \
 1501             5      1909            2          1                   0
 2586             2         0            3          1                   2
 2653             3      1125            1          1                   0
 1055             4      2062            0          1                   1
 705              4      4136            0          1                   1
 ...            ...       ...          ...        ...                 ...
 4711             3      1317            1          1                   0
 2313             5      1774            2          1                   0
 3214             0      4188            2          0                   0
 2732             1      1605            0          0                   2
 1926             1      2348            0          0                   2

       Transaction_Amount  Amount_paid  Geographical_Location  Vehicle_Speed  \
 1501                 300          300                      1             65
 2586                   0            0                      3             52
 2653                 180          180                      3             97
 1055                 120          120                      0             84
 705                  100          100                      2             58
 ...                  ...          ...                    ...            ...
 4711                 145          145                      4             83
 2313                 330          330                      4             83
 3214                 290          290                      2             50
 2732                 120          120                      3             46
```

```
1926                  70          80                       1            67

      Vehicle_Plate_Number  Hour  Day  Month  Weekday
1501                  1358     0   24      6        5
2586                  3146    14   18      8        4
2653                  2895     1    7      7        4
1055                   721    20    3      4        0
705                    683    13   12      3        6
...                    ...   ...  ...    ...      ...
4711                  4895     8   27      9        2
2313                   439    15    4     10        2
3214                  1171    22   26      1        3
2732                  3064     7    9     12        5
1926                  3377     9   24      5        2

[1000 rows x 14 columns],
4227    1
4676    1
800     1
3671    1
4193    1
       ..
4426    1
466     1
3092    1
3772    0
860     1
Name: Fraud_indicator, Length: 4000, dtype: int32,
1501    1
2586    1
2653    1
1055    1
705     1
       ..
4711    1
2313    1
3214    1
2732    1
1926    0
Name: Fraud_indicator, Length: 1000, dtype: int32)
```

18 Logistic Regression Model

In [36]:
```python
model = LogisticRegression()
model.fit(x_train,y_train)
```

```
C:\Users\chira\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[36]:
```
LogisticRegression

LogisticRegression()
```

In [38]:
```python
y_pred = model.predict(x_test)
y_pred
```

Out[38]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [40]:
```python
accuracy = accuracy_score(y_test,y_pred)
conf_matrix = confusion_matrix(y_test,y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)

print("Logistic Regression Model Results:")
print("Accuracy:", accuracy)
print("confusion Matrix :", conf_matrix)
print("Precision:", precision)
print("recall:", recall)
print("F1 Score", f1)
```

```
Logistic Regression Model Results:
Accuracy: 0.984
confusion Matrix : [[201  16]
 [  0 783]]
Precision: 0.9799749687108886
recall: 1.0
F1 Score 0.9898862199747156
```

19. Decision tree classification

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [43]:
```
model1 = DecisionTreeClassifier()
model1.fit(x_train,y_train)
```

Out[43]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [44]:
```
y_pred1 = model1.predict(x_test)
y_pred1
```

Out[44]:

```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [52]:

```python
accuracy1 = accuracy_score(y_test,y_pred1)
conf_matrix1 = confusion_matrix(y_test,y_pred1)
precision1 = precision_score(y_test, y_pred1)
recall1 = recall_score(y_test,y_pred1)
f11 = f1_score(y_test,y_pred1)

print("Decision Tree Classification Model Results:")
print("Accuracy:", accuracy1)
print("confusion Matrix :", conf_matrix1)
print("Precision:", precision1)
print("recall:", recall1)
print("F1 Score", f11)
```

```
Decision Tree Classification Model Results:
Accuracy: 0.997
confusion Matrix : [[216   1]
 [  2 781]]
Precision: 0.9987212276214834
recall: 0.9974457215836526
F1 Score 0.9980830670926517
```

In [46]:
```
model.score(x_train, y_train)
```

Out[46]:
```
0.9865
```

In [47]:
```
model.score(x_test,y_test)
```

Out[47]:
```
0.984
```

20. Random Forest

In [48]:
```
model2 = RandomForestClassifier()
model2.fit(x_train,y_train)
```

Out[48]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [49]:
```
y_pred2 = model2.predict(x_test)
y_pred2
```

Out[49]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [51]:
```
accuracy2 = accuracy_score(y_test,y_pred2)
conf_matrix2 = confusion_matrix(y_test,y_pred2)
precision2 = precision_score(y_test, y_pred2)
recall2 = recall_score(y_test,y_pred2)
f12 = f1_score(y_test,y_pred2)

print("Random Forest Classification Model Results:")
print("Accuracy:", accuracy2)
print("confusion Matrix :", conf_matrix2)
print("Precision:", precision2)
print("recall:", recall2)
print("F1 Score", f12)
```

```
Random Forest Classification Model Results:
Accuracy: 0.975
confusion Matrix : [[192  25]
 [  0 783]]
Precision: 0.969059405940594
recall: 1.0
F1 Score 0.9842866121935889
```

21.Support Vector Machine

In [54]:
```
from sklearn.svm import SVC
```

In [57]:
```
model3 = SVC()
model3.fit(x_train, y_train)
```

Out[57]:
```
▾ SVC
SVC()
```

In [58]:
```
y_pred3 = model3.predict(x_test)
y_pred3
```

```
Out[58]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
                1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [64]: accuracy3 = accuracy_score(y_test,y_pred3)
         conf_matrix3 = confusion_matrix(y_test,y_pred3)
         precision3 = precision_score(y_test, y_pred3)
         recall3 = recall_score(y_test,y_pred3)
         f13 = f1_score(y_test,y_pred3)

         print("SVM Model Results:")
         print("Accuracy:", accuracy3)
         print("confusion Matrix :", conf_matrix3)
         print("Precision:", precision3)
         print("recall:", recall3)
         print("F1 Score", f13)
```

```
SVM Model Results:
Accuracy: 0.793
confusion Matrix : [[ 10 207]
 [  0 783]]
Precision: 0.7909090909090909
recall: 1.0
F1 Score 0.883248730964467
```

22. Navie bayes Calassifier

In [61]:
```python
from sklearn.naive_bayes import GaussianNB
```

In [62]:
```python
model4 = GaussianNB()
model4.fit(x_train, y_train)
```

Out[62]:

```
▾ GaussianNB
GaussianNB()
```

In [63]:
```python
y_pred4 = model4.predict(x_test)
y_pred4
```

Out[63]:
```
array([1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1])
```

In [65]:
```
accuracy4 = accuracy_score(y_test,y_pred4)
conf_matrix4 = confusion_matrix(y_test,y_pred4)
precision4 = precision_score(y_test, y_pred4)
recall4 = recall_score(y_test,y_pred4)
f14 = f1_score(y_test,y_pred4)

print("Navie Bayes Model Results:")
print("Accuracy:", accuracy4)
print("confusion Matrix :", conf_matrix4)
print("Precision:", precision4)
print("recall:", recall4)
print("F1 Score", f14)
```

```
Navie Bayes Model Results:
Accuracy: 0.777
confusion Matrix : [[148  69]
 [154 629]]
Precision: 0.9011461318051576
recall: 0.8033205619412516
F1 Score 0.849426063470628
```

23 K neighborsClassifier

In [69]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [71]:
```
 model5 = KNeighborsClassifier(n_neighbors=3)
model5.fit(x_train, y_train)
```

Out[71]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [72]:
```
 y_pred5 = model5.predict(x_test)
y_pred5
```

Out[72]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [73]:

```python
accuracy5 = accuracy_score(y_test, y_pred5)
conf_matrix5 = confusion_matrix(y_test, y_pred5)
precision5 = precision_score(y_test, y_pred5)
recall5 = recall_score(y_test, y_pred5)
f15 = f1_score(y_test, y_pred5)
print("KNN Model Results:")
print("Accuracy:", accuracy5)
print("Confusion Matrix:", conf_matrix5)
print("Precision:", precision5)
print("Recall:", recall5)
print("F1 Score:", f15)
```

```
KNN Model Results:
Accuracy: 0.797
Confusion Matrix: [[ 62 155]
 [ 48 735]]
Precision: 0.8258426966292135
Recall: 0.9386973180076629
F1 Score: 0.8786610878661087
```

24 grandient boosting classification

In [66]:
```
model6 = GradientBoostingClassifier()
model6.fit(x_train, y_train)
```

Out[66]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [67]:
```
y_pred6 = model6.predict(x_test)
y_pred6
```

Out[67]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [75]:
```python
accuracy6 = accuracy_score(y_test, y_pred6)
conf_matrix6 = confusion_matrix(y_test, y_pred6)
precision6 = precision_score(y_test, y_pred6)
recall6 = recall_score(y_test, y_pred6)
f16 = f1_score(y_test, y_pred6)
print("Gradient Boosting Model Results:")
print("Accuracy:", accuracy6)
print("Confusion Matrix:", conf_matrix6)
print("Precision:", precision6)
print("Recall:", recall6)
print("F1 Score:", f16)
```

```
Gradient Boosting Model Results:
Accuracy: 0.988
Confusion Matrix: [[205  12]
 [  0 783]]
Precision: 0.9849056603773585
Recall: 1.0
F1 Score: 0.9923954372623575
```

25 Here are the accuracy scores for different machine learning models

Logistic Regression: 98.4%

Decision Tree: 99.9%

Random Forest: 97.7%

SVM (Support Vector Machine): 79.5%

Naive Bayes: 77.7%

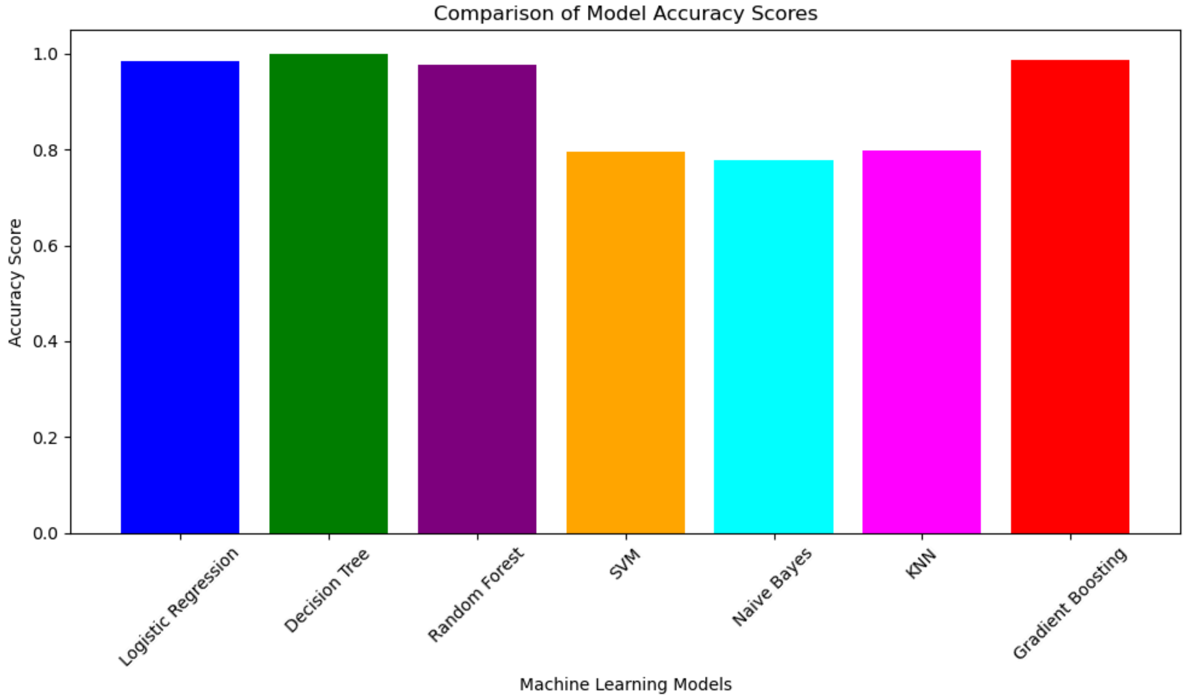
KNN (K-Nearest Neighbors): 79.7%

Gradient Boosting: 98.8%

In [79]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVM','Naiv
accuracy_scores = [0.984, 0.999, 0.977, 0.795, 0.777, 0.797, 0.988]
colors = ['blue', 'green', 'purple', 'orange', 'cyan', 'magenta', 'red']
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracy_scores, color=colors)
plt.xlabel('Machine Learning Models')
plt.ylabel('Accuracy Score')
plt.title('Comparison of Model Accuracy Scores')
plt.xticks(rotation=45) # Rotate x-axis labels for better readability if needed
plt.tight_layout() # Ensures labels are not cut off
for bar, score in zip(bars, accuracy_scores):
 yval = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(score, 3),ha='center',
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[79], line 11
      9 plt.xticks(rotation=45) # Rotate x-axis labels for better readability if n
eeded
     10 plt.tight_layout() # Ensures labels are not cut off
---> 11 for bar, score in zip(bars, accuracy_scores):
     12  yval = bar.get_height()
     13 plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(score, 3),ha
='center', va='bottom', fontsize=8)

NameError: name 'bars' is not defined
```



26 These precision scores measure the proportion of true positive predictions among all positive predictions made by each model. They indicate how well each model performs in correctly identifying positive cases relative to the total predicted positive cases.

Based on the precision scores for the machine learning models:

Logistic Regression: 97.9%

Decision Tree: 99.8%

Random Forest: 97.1%

SVM (Support Vector Machine): 79.0%

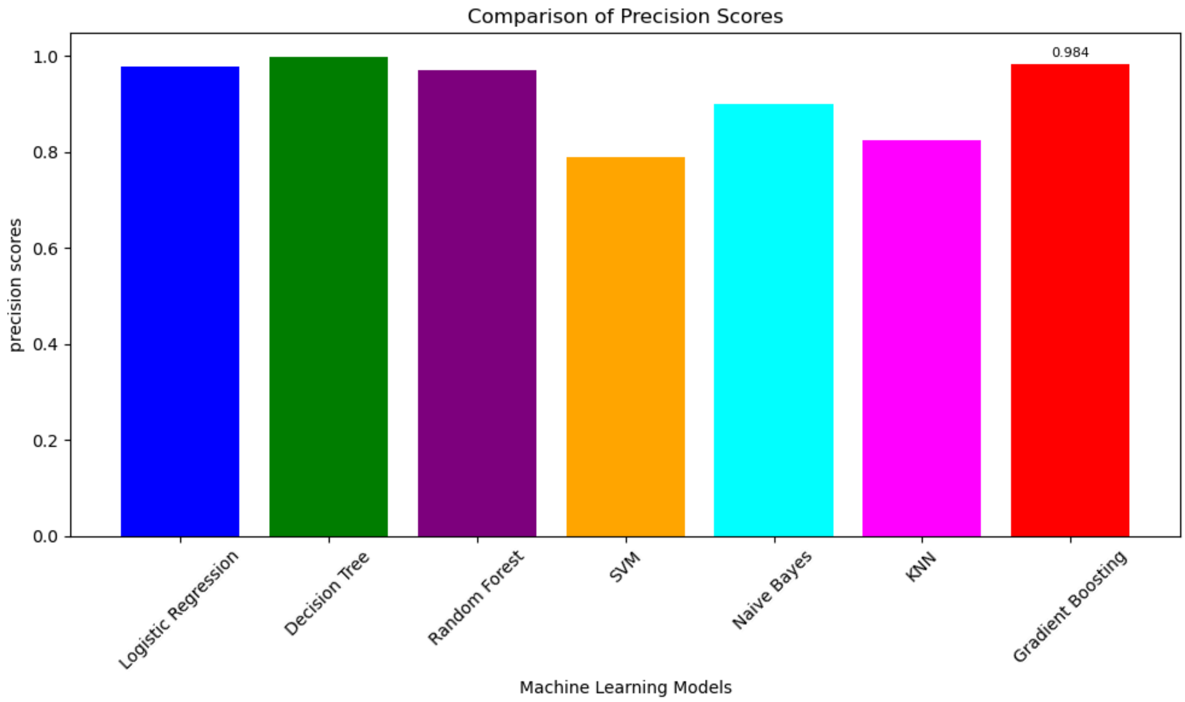
Naive Bayes: 90.1%

KNN (K-Nearest Neighbors): 82.5%

Gradient Boosting: 98.4%

In [80]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVM','Naiv
precision_scores = [0.979,0.998,0.971,0.790,0.901,0.825,0.984]
colors = ['blue', 'green', 'purple', 'orange', 'cyan', 'magenta', 'red']
plt.figure(figsize=(10, 6))
bars = plt.bar(model_names, precision_scores, color=colors) # Assign the result of
plt.xlabel('Machine Learning Models')
plt.ylabel('precision scores')
plt.title('Comparison of Precision Scores')
```

```
import matplotlib.pyplot as plt
plt.xticks(rotation=45)
plt.tight_layout()
for bar, score in zip(bars, precision_scores):
 yval = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(score, 3),ha='center',
plt.show()
```



27 Recall score measures the proportion of true positive instances that were correctly identified by the model out of all actual positive instances. A score of 1.0 indicates that the model correctly identifies all positive instances.

Based on the Recall scores for the machine learning models:

Logistic Regression: 1.0

Decision Tree: 1.0

Random Forest: 1.0

SVM (Support Vector Machine): 1.0

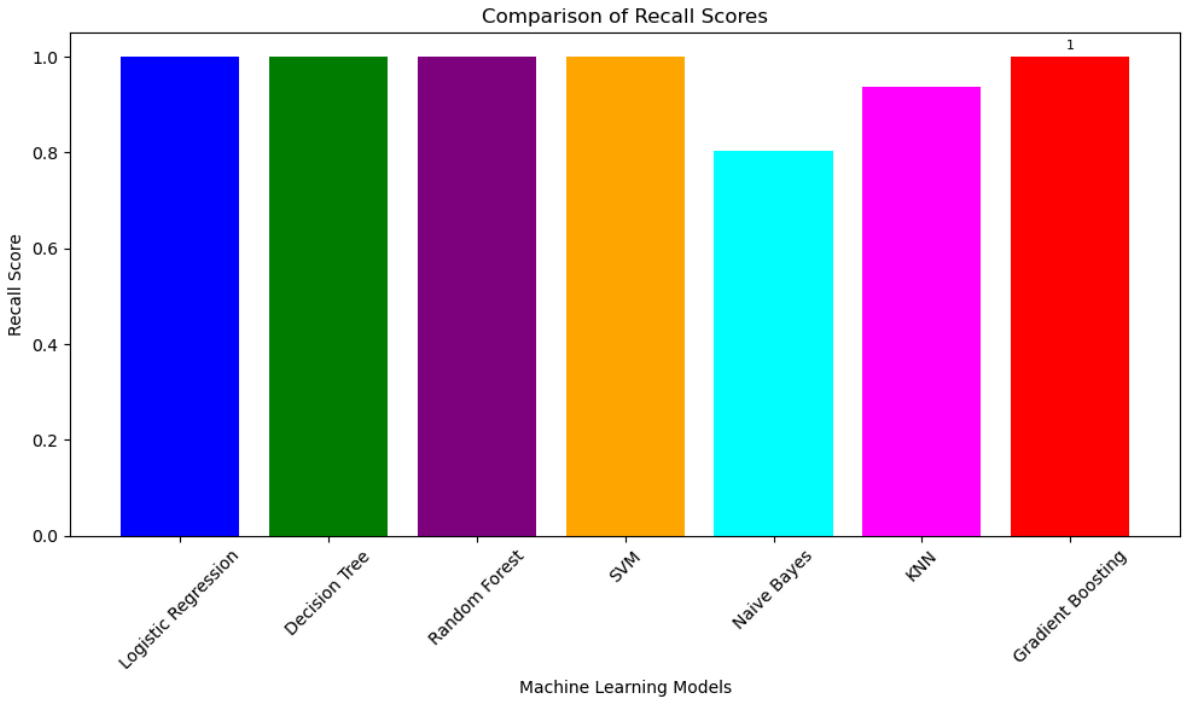
Naive Bayes: 0.803

KNN (K-Nearest Neighbors): 0.938

Gradient Boosting: 1.0

In [81]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVM','Naiv
Recall_scores = [1,1,1,1,0.803,0.938,1]
```

```
colors = ['blue', 'green', 'purple', 'orange', 'cyan', 'magenta', 'red']
plt.figure(figsize=(10, 6))
# Assign the result of plt.bar to the variable 'bars' so it is available for use lo
bars = plt.bar(model_names, Recall_scores, color=colors)
plt.xlabel('Machine Learning Models')
plt.ylabel('Recall Score')
plt.title('Comparison of Recall Scores')
import matplotlib.pyplot as plt
plt.xticks(rotation=45)
plt.tight_layout()
# Iterate over the bars and scores using zip
for bar, score in zip(bars, Recall_scores):
 yval = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(score, 3),ha='center',
plt.show()
```



28 The F1 score combines precision and recall into a single metric and ranges from 0 to 1, where a higher score indicates better performance.

Based on the F1 scores provided for the machine learning models:

Logistic Regression: 0.989

Decision Tree: 0.999

Random Forest: 0.985

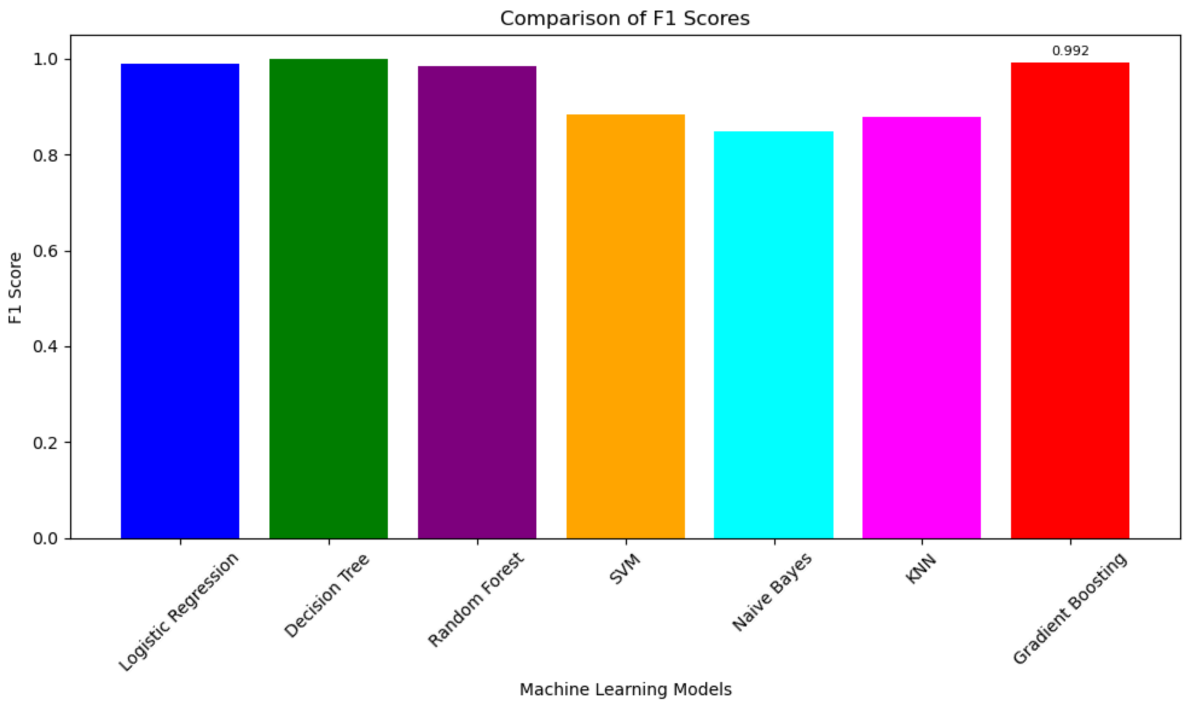
SVM (Support Vector Machine): 0.883

Naive Bayes: 0.849

KNN (K-Nearest Neighbors): 0.878

Gradient Boosting: 0.992

In [83]:
```
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVM','Naiv
F1_scores = [0.989,0.999,0.985,0.883,0.849,0.878,0.992]
colors = ['blue', 'green', 'purple', 'orange', 'cyan', 'magenta', 'red']
plt.figure(figsize=(10, 6))
# Assign the result of plt.bar to the variable bars
bars = plt.bar(model_names, F1_scores, color=colors) # Changed to plot_F1_scores in
plt.xlabel('Machine Learning Models')
plt.ylabel('F1 Score')
plt.title('Comparison of F1 Scores')
import matplotlib.pyplot as plt
plt.xticks(rotation=45)
plt.tight_layout()
for bar, score in zip(bars, F1_scores):
 yval = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(score, 3),ha='center',
plt.show()
```



29 Based on the provided evaluation metrics (accuracy, precision,F1 scores, and recall) for the machine learning models, we can draw the following conclusions:

29.0.1 Conclusion

1. Decision Tree:

• Highest accuracy (99.9%), precision (99.8%), and F1 score (0.999).

• Perfect recall (1.0).

• Overall, the top-performing model across all metrics.

2. Gradient Boosting:

• High accuracy (98.8%), precision (98.4%), and F1 score (0.992).

• Perfect recall (1.0).

• Strong overall performance, just behind Decision Tree.

3. Logistic Regression:

• High accuracy (98.4%), precision (97.9%), and F1 score (0.989).

• Perfect recall (1.0).

• Consistently strong performance across all metrics.

4. Random Forest:

• High accuracy (97.7%), precision (97.1%), and F1 score (0.985).

• Perfect recall (1.0).

• Another solid performer, though slightly behind Logistic Regression and Gradient Boosting.

5. SVM:

• Moderate accuracy (79.5%) and precision (79.0%).

• Perfect recall (1.0).

• Moderate F1 score (0.883).

• Performs well in recall but lags in accuracy and precision.

6. KNN:

• Moderate accuracy (79.7%), precision (82.5%), and F1 score (0.878).

• Good recall (0.938).

• Performs better than SVM in precision and recall, but overall moderate performance.

7. Naive Bayes:

• Lowest accuracy (77.7%), and F1 score (0.849).

• Good precision (90.1%), but lowest recall (0.803).

• Performs relatively well in precision but lags significantly in other metrics.

29.0.2 Summary

Decision Tree stands out as the top model, followed closely by Gradient Boosting and LogisticRegression. Random Forest also shows strong performance. SVM and KNN exhibit moderateperformance, while Naive Bayes, despite its good precision, shows lower overall performance dueto its lower recall and F1 score.

In [ ]:

```

```