Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Report on Mini Project
Machine Learning -I (DJ19DSC402)
AY: 2021-22

FRAUD DETECTION

NAME: Dev Patel **SAP ID: 60009200016**
NAME: Prachet Shah **SAP ID: 60009200029**

Guided By
Dr. Kriti Srivastava

CHAPTER 1: INTRODUCTION

Fraud Detection using Machine Learning deploys a machine learning (ML) model and an example dataset of credit card transactions to train the model to recognize fraud patterns. In Machine Learning terminology, Fraud Detection problem may be framed as a classification problem, of which the goal is to predict the discrete label 0 or 1 where 0 generally suggests that a transaction is non-fraudulent and 1 suggest that the transaction seems to be fraudulent. The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect maximum number of the fraudulent transactions while minimizing the incorrect fraud classifications using the following features (columns) in our dataset:

CHAPTER 2: DATA DESCRIPTION

The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect maximum number of the fraudulent transactions while minimizing the incorrect fraud classifications using the following features (columns) in our dataset:

- step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).
- type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
- amount - amount of the transaction in local currency.
- nameOrig - customer who started the transaction
- oldbalanceOrg - initial balance before the transaction
- newbalanceOrig - new balance after the transaction
- nameDest - customer who is the recipient of the transaction
- oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
- newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).
- isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers' accounts and try to empty the funds by transferring to another account and then cashing out of the system.
- isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200,000 in a single transaction.

CHAPTER 3: DATA ANALYSIS

After importing the dataset, shape of the dataset and the features in it were checked. After checking the information present in the columns, there was a check for null values and the description of statistics of the dataset for checking outliers. Value counts of the target column was checked for imbalanced data. As the dataset was imbalanced, the categories having the most fraud cases were found and reshaped randomly to balance the dataset for reducing overfitting. Concatenation of these data-frames to get our dataset and converting categorical columns to numerical columns using Label Encoding was done. Feature selection to drop the insignificant features was done and the final dataset was saved for future reference.

CHAPTER 4: REASONS FOR SELECTING MACHINE LEARNING MODELS

As this is a classification problem, several algorithms suitable for this dataset were used. The one which gave the best overall performance were selected. The average of all scores were compared while comparing models during evaluation trained on different training sets.

1. Logistic Regression: As the target feature is binary, logistic regression was implemented on the dataset and its accuracy was 90.33%
2. Linear Support Vector Machine (SVM): As the dimensionality in data was high, used Linear SVM on it to check whether a hyperplane is able to segregate it or not.
3. RBF Kernel SVM: As the dimensionality in data was high, tried kernel function SVM on it to check whether transformed data can create a hyperplane to segregate our features or not.
4. Naïve Bayes: Naive Bayes although works exceptionally well on small data but as the data size here was enormous and due to presence of randomness throughout the feature space it became computationally expensive to train it and worked terribly.
5. Decision Tree: Branch method was applied to check whether a single tree is able to classify this binary classification data based on noise and data containing class imbalance.
6. Random Forest: Ensemble method was applied to check whether various weak learners are able to find suitable conditionals and gain huge confidence in classifying the data.

CHAPTER 5: ALGORITHM

Logistic Regression builds a classifier that divides the space in two parts which is suitable only for binary target variable whereas Linear SVM also provides the same outcome but instead of considering all the points on the dataset, it only uses the points on the edge of the margin of the gutter and RBF Kernel builds a non-linear classifier. Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. Decision Tree is an algorithm where the data is continuously split according to a certain parameter. It is drawn in a top-down approach with the root at the top and the leaf nodes having as much purity as possible. As Decision Trees are prone to overfitting the hyper parameters need to be adjusted to get a more generalised model. Random Forest is an ensemble of Decision Trees.

CHAPTER 6: RESULT ANALYSIS

Decision Tree Classifier showed the highest recall, accuracy and F1 score when compared to others:
Recall - 99.1799 %, Accuracy - 99.124 %, f1 score - 99.108 %
Rejecting other networks because: Neural Networks are rejected because it obviously takes a lot of training time & computation power.
Performance of other models that were tested are:

Model	Accuracy (%)	Recall (%)	F1 Score (%)
Decision Tree	99.17	99.32	99.16
Random Forest	98.94	99.42	98.92
Logistic Regression	90.33	85.06	89.61
Linear SVM	90.16	84.71	89.41
RBF Kernel SVM	88.48	81.38	87.39
Naïve Bayes	72.59	45.92	62.16

CHAPTER 7: CONCLUSION AND FUTURE SCOPE

On performing model selection, we were able to conclude that Decision Tree was best suited for this problem as it was seen from data itself that it had a lot of class imbalance. It was best suited for Decision Tree Classification. One Point to note here is that, the Ensemble method of Bagging, i.e., Random Forest Classification is close in performance compared to its parent Decision Tree. However, weak learners are not able to surpass a single Decision Tree for better output. The metrics fluctuated on different training sets for both classifications but average of all came in favour of Decision Tree Classification.

CHAPTER 8: PYTHON NOTEBOOK

Pre-processing and EDA

In [ ]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
plt.style.use('dark_background')
```

In [ ]:

```
# Importing the dataset
dataset = pd.read_csv('AIML Dataset.csv')
dataset.head()
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0

In [ ]:

```
dataset.shape
```

Out[ ]:

```
(6362620, 11)
```

In [ ]:

```
dataset.columns
```

Out[ ]:

```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

In [ ]:

```
# Used to find if dataset has any missing values
dataset.isna().sum().any()
```

Out[ ]:

```
False
```

In [ ]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
```

```
---  ------         -----
 0   step           int64
 1   type           object
 2   amount         float64
 3   nameOrig       object
 4   oldbalanceOrg  float64
 5   newbalanceOrig float64
 6   nameDest       object
 7   oldbalanceDest float64
 8   newbalanceDest float64
 9   isFraud        int64
 10  isFlaggedFraud int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

In [ ]:

```
dataset.describe()
```

Out[ ]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlagge
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.3626
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2.5146
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.5857
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0.0000
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0.0000
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1.0000

In [ ]:

```
dataset.isFraud.value_counts()
```

Out[ ]:

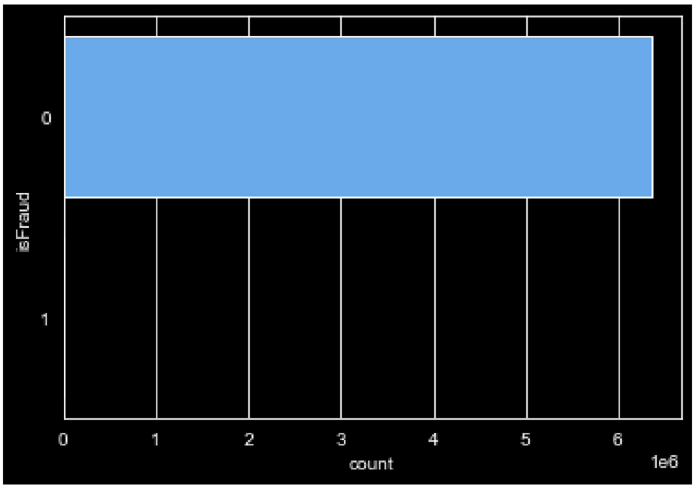
```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

In [ ]:

```
sns.countplot(y = dataset['isFraud'], palette='cool')
```

Out[ ]:

<AxesSubplot:xlabel='count', ylabel='isFraud'>



In [ ]:

In [ ]:

```
dataset.type.value_counts()
```

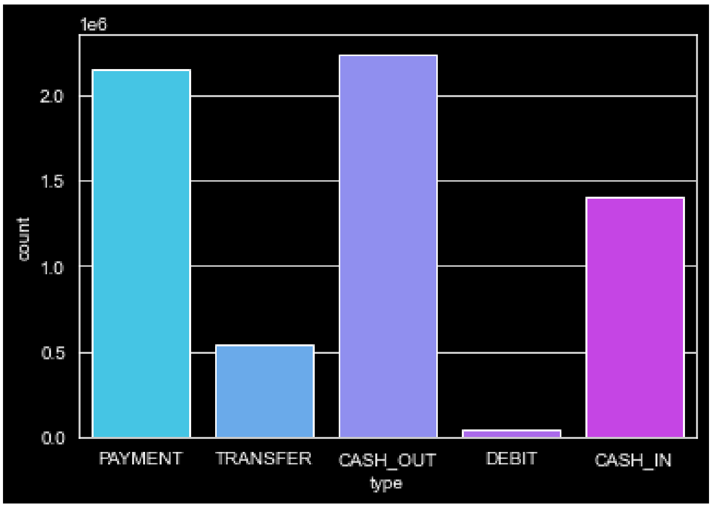
Out[ ]:

```
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: type, dtype: int64
```

In [ ]:

```
sns.countplot(dataset['type'], palette='cool');
```

```
c:\Users\prach\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [ ]:

```
# Finding which categories have most frauds
dfFraudCash_Out = dataset.loc[(dataset.isFraud == 1) & (dataset.type == 'CASH_OUT')]
dfFraudPayment = dataset.loc[(dataset.isFraud == 1) & (dataset.type == 'PAYMENT')]
dfFraudCash_in = dataset.loc[(dataset.isFraud == 1) & (dataset.type == 'CASH_IN')]
dfFraudTransfer = dataset.loc[(dataset.isFraud == 1) & (dataset.type == 'TRANSFER')]
dfFraudDebit = dataset.loc[(dataset.isFraud == 1) & (dataset.type == 'DEBIT')]
print("Tran. Type\tNo of Frauds")
print(f"Cash-Out\t {len(dfFraudCash_Out)}")
print(f"Payment \t {len(dfFraudPayment)}")
print(f"Cash-in \t {len(dfFraudCash_in)}")
print(f"Transfer \t {len(dfFraudTransfer)}")
print(f"Debit \t         {len(dfFraudDebit)}")
```

```
Tran. Type No of Frauds
Cash-Out   4116
Payment    0
Cash-in    0
Transfer    4097
Debit            0
```

It is clear from this that only 2 type of transactions have fraudulent transactions, so we will take them as our dataset for training

In [ ]:

```
# we will be taking only these types into our data for analysis as they are the only one which have fraudulent transactions
fraud_cashout = dataset.loc[(dataset.isFraud == 1) & (dataset['type'] == 'CASH_OUT')]
fraud_transfer = dataset.loc[(dataset.isFraud == 1) & (dataset['type'] == 'TRANSFER')]
```

```
fraud_cashout.head()
fraud_transfer.head()
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalance
2	1	TRANSFER	181.00	C1305486145	181.00	0.0	C553264065	0.0	
251	1	TRANSFER	2806.00	C1420196421	2806.00	0.0	C972765878	0.0	
680	1	TRANSFER	20128.00	C137533655	20128.00	0.0	C1848415041	0.0	
969	1	TRANSFER	1277212.77	C1334405552	1277212.77	0.0	C431687661	0.0	
1115	1	TRANSFER	35063.63	C1364127192	35063.63	0.0	C1136419747	0.0	

In [ ]:

```
# finding how many not fraudulent transactions cash out and transfers have
dfnotFraudCash_Out = dataset.loc[(dataset.isFraud == 0) & (dataset.type == 'CASH_OUT')]
dfnotFraudTransfer = dataset.loc[(dataset.isFraud == 0) & (dataset.type == 'TRANSFER')]

print(len(dfnotFraudCash_Out))
print(len(dfnotFraudTransfer))
```

```
2233384
528812
```

Reshaping the data randomly to balance the dataset and reduce overfitting

In [ ]:

```
#reshaping the data randomly to balance the dataset and reduce overfitting
data1 = dataset.loc[(dataset.isFraud == 0) & (dataset['type'] == 'CASH_OUT')].sample(fra
c=0.002)
data2 = dataset.loc[(dataset.isFraud == 0) & (dataset['type'] == 'TRANSFER')].sample(fra
c=0.008)
```

In [ ]:

```
data1
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalan
5297153	373	CASH_OUT	190404.20	C1863770860	255392.00	64987.80	C2062256711	0.00	19
2920898	229	CASH_OUT	244617.86	C1651156825	0.00	0.00	C1457685836	2621835.29	286
5667167	396	CASH_OUT	148030.81	C1036218413	7605.00	0.00	C1336579882	817.00	14
3066283	234	CASH_OUT	53140.75	C1275070321	0.00	0.00	C828389234	5956313.25	600
5482461	379	CASH_OUT	102506.05	C503072038	105636.75	3130.70	C1507706628	841586.14	94
...	...	...	...	...	...	...	...	...	
54842	9	CASH_OUT	187896.00	C1726746477	0.00	0.00	C358804045	785894.29	119
1948575	177	CASH_OUT	131077.91	C1949332980	0.00	0.00	C1514338793	12226030.83	1235
5815648	401	CASH_OUT	109503.77	C2129848349	0.00	0.00	C236736835	536971.78	64
2434297	203	CASH_OUT	31425.63	C1223567408	0.00	0.00	C499232658	143915.76	17
511686	20	CASH_OUT	173036.89	C15894394	496741.27	323704.38	C1212218920	607868.41	78

4467 rows × 11 columns

In [ ]:

```
data2
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbala
3681699	276	TRANSFER	221979.16	C1595923938	0.0	0.0	C2090775687	679324.04	9
1995302	179	TRANSFER	2248475.89	C1714374648	0.0	0.0	C1794278144	2626616.93	48
4000070	298	TRANSFER	647849.92	C541518341	0.0	0.0	C1461222852	819732.91	14
2600929	208	TRANSFER	915228.18	C1147091765	0.0	0.0	C287259122	2147225.36	30
1010323	46	TRANSFER	303251.37	C686114206	43659.0	0.0	C1031864738	111766.03	4
...	...	...	...	...	...	...	...	...	
2255141	187	TRANSFER	761188.52	C1597156216	0.0	0.0	C81847230	4222378.54	49
51995	9	TRANSFER	1067282.76	C637763170	61617.0	0.0	C1165398731	0.00	14
2074817	182	TRANSFER	114814.29	C1783363242	0.0	0.0	C37833977	1441501.99	15
2589511	207	TRANSFER	1335958.23	C1171885319	6082.0	0.0	C929780613	247498.00	15
5578772	393	TRANSFER	924877.66	C966808862	164505.0	0.0	C2060267019	0.00	9

4230 rows × 11 columns

In [ ]:

```
# creating our data set by concatenating the segregated data frames
fraud_data = pd.concat([data1,fraud_cashout, data2, fraud_transfer])
fraud_data
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbala
5297153	373	CASH_OUT	190404.20	C1863770860	255392.00	64987.8	C2062256711	0.00	1
2920898	229	CASH_OUT	244617.86	C1651156825	0.00	0.0	C1457685836	2621835.29	28
5667167	396	CASH_OUT	148030.81	C1036218413	7605.00	0.0	C1336579882	817.00	1
3066283	234	CASH_OUT	53140.75	C1275070321	0.00	0.0	C828389234	5956313.25	60
5482461	379	CASH_OUT	102506.05	C503072038	105636.75	3130.7	C1507706628	841586.14	9
...	...	...	...	...	...	...	...	...	
6362610	742	TRANSFER	63416.99	C778071008	63416.99	0.0	C1812552860	0.00	
6362612	743	TRANSFER	1258818.82	C1531301470	1258818.82	0.0	C1470998563	0.00	
6362614	743	TRANSFER	339682.13	C2013999242	339682.13	0.0	C1850423904	0.00	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	

16910 rows × 11 columns

In [ ]:

```
fraud_data.isFraud.sum()
```

Out[ ]:

```
8213
```

Now, column type is Categorical object which we convert into Numerical Data to apply operations on it

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
label = le.fit_transform(fraud_data['type'])
label
```

Out[ ]:

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

In [ ]:

```
# Inserting converted type data into our dataset
fraud_data.insert(2,'type_num', label)
```

In [ ]:

```
# Converting nameOrig column into only ids by removing C from its front
fraud_data['nameOrig'] = fraud_data['nameOrig'].replace({'C': ''}, regex=True)
fraud_data
```

Out[ ]:

	step	type	type_num	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
5297153	373	CASH_OUT	0	190404.20	1863770860	255392.00	64987.8	C2062256711	0.00
2920898	229	CASH_OUT	0	244617.86	1651156825	0.00	0.0	C1457685836	2621835.29
5667167	396	CASH_OUT	0	148030.81	1036218413	7605.00	0.0	C1336579882	817.00
3066283	234	CASH_OUT	0	53140.75	1275070321	0.00	0.0	C828389234	5956313.25
5482461	379	CASH_OUT	0	102506.05	503072038	105636.75	3130.7	C1507706628	841586.14
...	...	...	...	...	...	...	...	...	...
6362610	742	TRANSFER	1	63416.99	778071008	63416.99	0.0	C1812552860	0.00
6362612	743	TRANSFER	1	1258818.82	1531301470	1258818.82	0.0	C1470998563	0.00
6362614	743	TRANSFER	1	339682.13	2013999242	339682.13	0.0	C1850423904	0.00
6362616	743	TRANSFER	1	6311409.28	1529008245	6311409.28	0.0	C1881841831	0.00
6362618	743	TRANSFER	1	850002.52	1685995037	850002.52	0.0	C2080388513	0.00

16910 rows × 12 columns

In [ ]:

```
fraud_data['nameOrig'] = pd.to_numeric(fraud_data['nameOrig'])
fraud_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16910 entries, 5297153 to 6362618
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   step            16910 non-null  int64  
 1   type            16910 non-null  object 
 2   type_num        16910 non-null  int32  
 3   amount          16910 non-null  float64
 4   nameOrig        16910 non-null  int64  
 5   oldbalanceOrg   16910 non-null  float64
 6   newbalanceOrig  16910 non-null  float64
 7   nameDest        16910 non-null  object 
 8   oldbalanceDest  16910 non-null  float64
 9   newbalanceDest  16910 non-null  float64
 10  isFraud         16910 non-null  int64  
 11  isFlaggedFraud  16910 non-null  int64  
dtypes: float64(5), int32(1), int64(4), object(2)
memory usage: 1.6+ MB
```

Feature Selection

Columns step, nameDest, type and isFlaggedFraud are not taken into consideration for training our prediction models because column nameDest is string which are not providing any significance to our data, type column is dropped because we already converted it into numerical data and isFlaggedFraud is removed because we believe that it is the pre determined output which needs to be found out by the model. Also step is just hour out of 30 days of simulation

In [ ]:

```
fraud_data = fraud_data.drop(['step','nameDest', 'type','isFlaggedFraud'], axis=1)
```

In [ ]:

```
fraud_data
```

Out[ ]:

	type_num	amount	nameOrig	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
5297153	0	190404.20	1863770860	255392.00	64987.8	0.00	190404.20	0
2920898	0	244617.86	1651156825	0.00	0.0	2621835.29	2866453.15	0
5667167	0	148030.81	1036218413	7605.00	0.0	817.00	148847.81	0
3066283	0	53140.75	1275070321	0.00	0.0	5956313.25	6009454.00	0
5482461	0	102506.05	503072038	105636.75	3130.7	841586.14	944092.19	0
...	...	...	...	...	...	...	...	...
6362610	1	63416.99	778071008	63416.99	0.0	0.00	0.00	1
6362612	1	1258818.82	1531301470	1258818.82	0.0	0.00	0.00	1
6362614	1	339682.13	2013999242	339682.13	0.0	0.00	0.00	1
6362616	1	6311409.28	1529008245	6311409.28	0.0	0.00	0.00	1
6362618	1	850002.52	1685995037	850002.52	0.0	0.00	0.00	1

16910 rows × 8 columns

In [ ]:

```
fraud_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16910 entries, 5297153 to 6362618
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   type_num        16910 non-null  int32
 1   amount          16910 non-null  float64
 2   nameOrig        16910 non-null  int64
 3   oldbalanceOrg   16910 non-null  float64
 4   newbalanceOrig  16910 non-null  float64
 5   oldbalanceDest  16910 non-null  float64
 6   newbalanceDest  16910 non-null  float64
 7   isFraud         16910 non-null  int64
dtypes: float64(5), int32(1), int64(2)
memory usage: 1.1 MB
```

In [ ]:

```
fraud_data.to_csv('fraud_data_final.csv')
```

We save this final dataset in csv format for future reference.

In [ ]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [25]:

```
fraud_data = pd.read_csv("fraud_data.csv")
```

Out[25]:

	type_num	amount	nameOrig	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
420497	0	46451.63	1934325941	0.00	0.00	615786.75	869426.68	0
5863189	0	128658.63	1656527982	79376.00	0.00	0.00	128658.63	0
6282695	0	96796.76	883615948	111244.00	14447.24	1694680.25	1791477.01	0
425178	0	158725.89	98294277	20887.00	0.00	0.00	158725.89	0
285852	0	142902.85	484082975	11276.00	0.00	492585.70	635488.54	0
...	...	...	...	...	...	...	...	...
6362610	1	63416.99	778071008	63416.99	0.00	0.00	0.00	1
6362612	1	1258818.82	1531301470	1258818.82	0.00	0.00	0.00	1
6362614	1	339682.13	2013999242	339682.13	0.00	0.00	0.00	1
6362616	1	6311409.28	1529008245	6311409.28	0.00	0.00	0.00	1
6362618	1	850002.52	1685995037	850002.52	0.00	0.00	0.00	1

16910 rows × 8 columns

In [26]:

```
fraud_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16910 entries, 420497 to 6362618
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   type_num        16910 non-null  int32
 1   amount          16910 non-null  float64
 2   nameOrig        16910 non-null  int64
 3   oldbalanceOrg   16910 non-null  float64
 4   newbalanceOrig  16910 non-null  float64
 5   oldbalanceDest  16910 non-null  float64
 6   newbalanceDest  16910 non-null  float64
 7   isFraud         16910 non-null  int64
dtypes: float64(5), int32(1), int64(2)
memory usage: 1.1 MB
```

In [27]:

```
fraud_data.to_csv('fraud_data_final.csv')
```

In [28]:

```
fraud_data.describe()
```

Out[28]:

	type_num	amount	nameOrig	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isF
count	16910.000000	1.691000e+04	1.691000e+04	1.691000e+04	1.691000e+04	1.691000e+04	1.691000e+04	16910.00

	type_num	amount	nameOrig	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isF
mean	0.492431	9.717754e+05	1.073634e+09	8.221470e+05	9.989258e+04	1.358271e+06	1.977505e+06	0.48
std	0.499957	1.923635e+06	6.180039e+08	2.601401e+06	1.374286e+06	4.607081e+06	5.147585e+06	0.49
min	0.000000	0.000000e+00	1.453640e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00
25%	0.000000	1.146486e+05	5.365574e+08	0.000000e+00	0.000000e+00	0.000000e+00	1.519429e+04	0.00
50%	0.000000	2.880359e+05	1.069090e+09	5.773459e+04	0.000000e+00	1.732895e+05	5.950143e+05	0.00
75%	1.000000	8.422898e+05	1.602564e+09	4.469653e+05	0.000000e+00	1.118714e+06	2.036695e+06	1.00
max	1.000000	3.877180e+07	2.147456e+09	5.958504e+07	4.958504e+07	2.362305e+08	2.367265e+08	1.00

In [29]:

```
# Assigning values
X = fraud_data.iloc[:, :-1].values
y = fraud_data.iloc[:, -1].values
```

In [30]:

```
X
```

Out[30]:

```
array([[0.00000000e+00, 4.64516300e+04, 1.93432594e+09, ...,
        0.00000000e+00, 6.15786750e+05, 8.69426680e+05],
       [0.00000000e+00, 1.28658630e+05, 1.65652798e+09, ...,
        0.00000000e+00, 0.00000000e+00, 1.28658630e+05],
       [0.00000000e+00, 9.67967600e+04, 8.83615948e+08, ...,
        1.44472400e+04, 1.69468025e+06, 1.79147701e+06],
       ...,
       [1.00000000e+00, 3.39682130e+05, 2.01399924e+09, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 6.31140928e+06, 1.52900824e+09, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 8.50002520e+05, 1.68599504e+09, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

In [31]:

```
y
```

Out[31]:

```
array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

Splitting data into train and test set and also apply Feature Scaling

In [32]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state
= 0)
```

In [33]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Model Selection

Training the Logistic Regression Model on training set

In [34]:

```
# Training the Logistic Regression model on the Training set
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 0)
lr.fit(X_train, y_train)
```

Out[34]:

```
LogisticRegression(random_state=0)
```

In [35]:

```
y_pred = lr.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)), 1))
print(y_pred)
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
 [0 0]
 [0 0]]
[1 1 1 ... 1 0 0]
```

Making the Confusion Matrix

In [36]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score, pre
cision_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Accuracy of model: {accuracy_score(y_test, y_pred)}")
```

```
[[2056   99]
 [ 310 1763]]
Accuracy of model: 0.9032639545884579
```

Recall Calculation

In [37]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.8504582730342499
```

F1 Score Calculation

In [38]:

```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

```
F1 Score of model: 0.8960609911054638
```

Training on SVM

In [39]:

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[39]:

```
SVC(kernel='linear', random_state=0)
```

In [40]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)), 1))
print(y_pred)
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
 [0 0]
 [0 0]]
[1 1 1 ... 1 0 0]
```

In [41]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score, pre
cision_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Accuracy of model: {accuracy_score(y_test, y_pred)}")
```

```
[[2056   99]
 [ 317 1756]]
Accuracy of model: 0.9016083254493851
```

In [42]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.8470815243608297
```

In [43]:

```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

```
F1 Score of model: 0.8940936863543789
```

Training on Naive bayes

In [44]:

```
from sklearn.naive_bayes import GaussianNB
bayes = GaussianNB()
bayes.fit(X_train, y_train)
```

Out[44]:

```
GaussianNB()
```

In [45]:

```
y_pred = bayes.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)), 1))
print(y_pred)
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
```

```
 [0 0]
 [0 0]]
[1 1 1 ... 1 0 0]
```

In [46]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score, pre
cision_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Accuracy of model: {accuracy_score(y_test, y_pred)}")
```

```
[[2117   38]
 [1121  952]]
Accuracy of model: 0.7258751182592242
```

In [47]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.4592378195851423
```

In [48]:

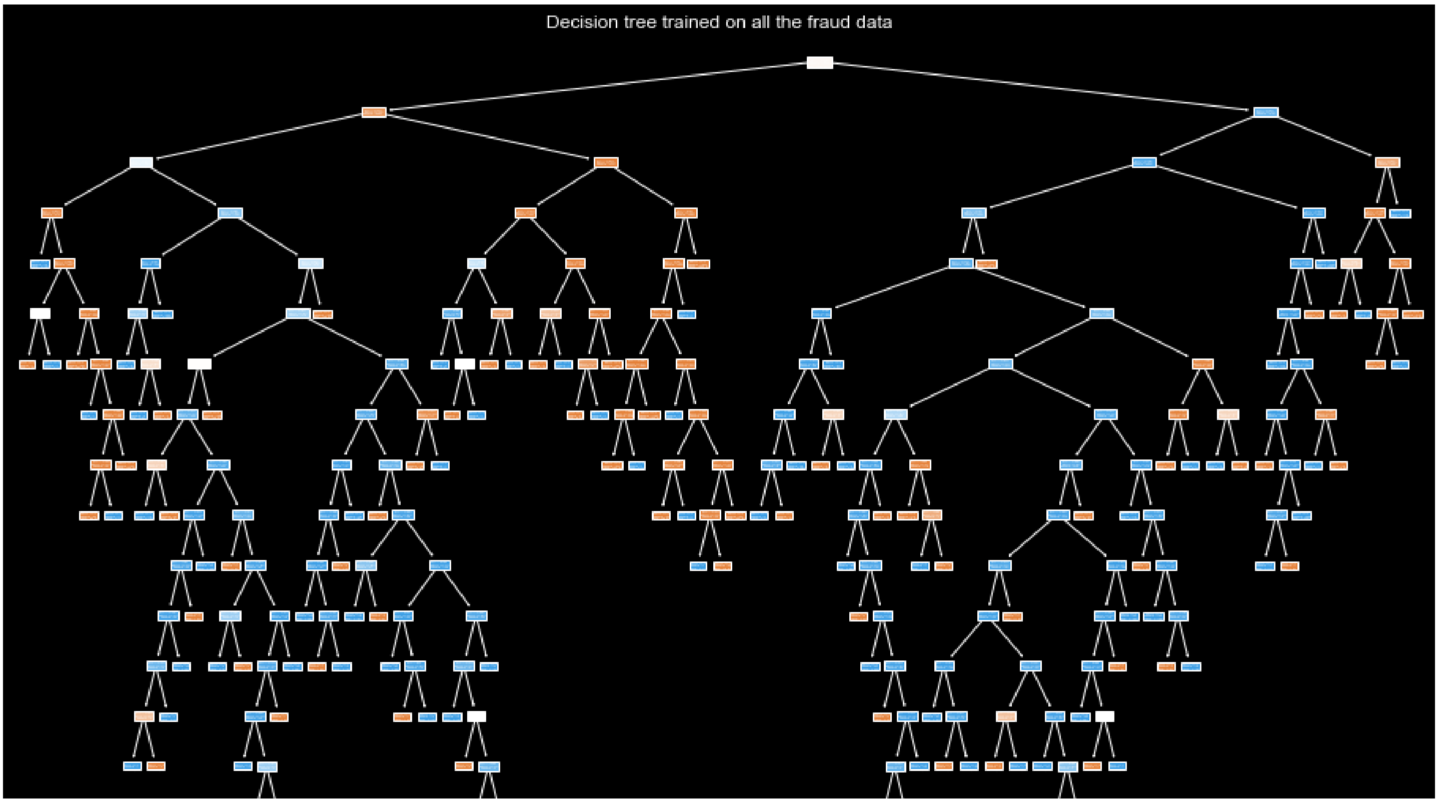
```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

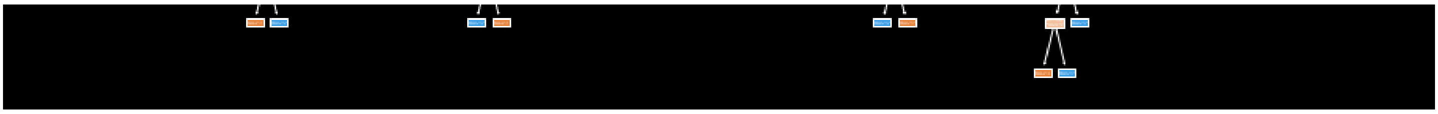
```
F1 Score of model: 0.6216127979105452
```

Decision Tree

In [49]:

```
# Training the Decision Tree Classification model on the Training set
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
dr = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
dr.fit(X_train, y_train)
fig = plt.figure(figsize=(16,10))
tree.plot_tree(dr, filled=True)
plt.title("Decision tree trained on all the fraud data")
plt.show()
```





In [50]:

```
# fig.savefig("decistion_tree.png")
```

In [51]:

```
y_pred = dr.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
 [0 0]
 [0 0]]
```

In [52]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score, pre
cision_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Accuracy of model: {accuracy_score(y_test, y_pred)}")
```

```
[[2134   21]
 [  14 2059]]
Accuracy of model: 0.9917218543046358
```

In [53]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.9932465026531597
```

In [54]:

```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

```
F1 Score of model: 0.9915723573320492
```

Random Forest

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 25, criterion = 'entropy', random_state = 0)
rf.fit(X_train, y_train)
```

Out[55]:

```
RandomForestClassifier(criterion='entropy', n_estimators=25, random_state=0)
```

In [56]:

```
y_pred = rf.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
```

```
 [0 0]
 [0 0]]
```

In [57]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score, pre
cision_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(f"Accuracy of model: {accuracy_score(y_test, y_pred)}")
```

```
[[2122   33]
 [  12 2061]]
Accuracy of model: 0.989356669820246
```

In [58]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.9942112879884226
```

In [59]:

```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

```
F1 Score of model: 0.9892008639308856
```

Training with Kernel SVM

In [60]:

```
from sklearn.svm import SVC
kernel_svm = SVC(kernel = 'rbf', random_state = 0)
kernel_svm.fit(X_train, y_train)
```

Out[60]:

```
SVC(random_state=0)
```

In [61]:

```
y_pred = kernel_svm.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
 [0 0]
 [0 0]]
```

In [62]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, f1_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[2054  101]
 [ 386 1687]]
```

Out[62]:

```
0.8848155156102177
```

In [63]:

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
```

```
# Recall = TruePositives / (TruePositives + FalseNegatives)
print(f"Recall Score of model: {recall_score(y_test, y_pred)}")
```

```
Recall Score of model: 0.8137964302942595
```

In [64]:

```
# 2*true positive /( 2*true positive + false positive + false negative)
print(f"F1 Score of model: {f1_score(y_test, y_pred)}")
```

```
F1 Score of model: 0.8738668738668739
```

Our Model Analysis

- **Decision Tree Classifier showed the highest recall, accuracy and F1 score when compared to others:**
- **Recall - 99.1799 % | Acc - 99.124 % | f1 - 99.108 %**
- **Rejecting other networks because:**
- **Neural Networks are rejected because it obviously takes a LOT of training time & computation power**
- **Performance of other models that were tested are:**

| Model | Recall(%) | F1(%) |

| Decision Tree | 99.1799 | 99.108 |

| Logistic Regression | 85.721 | 88.7021 |

| SVM | 84.02 | 88 |

| Naive bayes | 46 | 88 |

| Kernel SVM | 81.478 | 87.06 |

Predicting a single input

In [65]:

```
# type_num amount nameOrig oldbalanceOrg newbalanceOrig oldbalanceDest newbalanceDest
# print(classifier.predict(sc.transform([[0,74445.62,1796046115,0.00,0.0,1371784.99,1446230.61]])))

print(dr.predict([[1,34518.82,356646316,0.0,0.0,851831.14,886349.96],
               [1,278568.31,912325874,0.0,0.0,641896.48,920464.79],
               [1,475369.51,1184256533,0.0,0.0,1201817.38,1677186.90],
               [1,475368.94,916986889,475368.94,0.0,1348026.73,1823395.67],
               [1,594471.04,1345990968,594471.04,0.0,1788456.17,2382927.21]]))
```

```
[1 1 1 1 1]
```

We can see from above prediction that it came up with correct prediction for the above query

Saving our model

In [66]:

```
import pickle

# Dumping our model into a file
with open('fraud_model.bin', 'wb') as f_out:
    pickle.dump(dr, f_out)
```