__Credit Card Fraud Detection___

Importing Basic Libraries Pandas,Numpy and Data file

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\Hp\Desktop\DS_work\Deep Learning\12.02_Practical Session
df.head(2)
```

Out[2]:

	Transaction ID	Date	Time	Type of Card	Entry Mode	Amount	Type of Transaction	Merchant Group	Country of Transaction	Shi Ac
0	#2546 884	13-Oct-20	23	MasterCard	Tap	17.0	Online	Restaurant	United Kingdom	
1	#2546 895	14-Oct-20	21	Visa	Tap	28.0	Online	Gaming	United Kingdom	U Kin

EDA-Checking dataset for anamolies, disturbances and datatypes

In [3]:
```
df.shape
```

Out[3]:
```
(100000, 15)
```

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

Out[4]:

	Time	Amount	Age	Fraud
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	14.559320	112.566480	53.081630	0.071900
std	5.315905	123.428493	18.742452	0.258324
min	0.000000	5.000000	21.000000	0.000000
25%	10.000000	17.000000	37.000000	0.000000
50%	15.000000	30.000000	53.000000	0.000000
75%	19.000000	208.000000	69.000000	0.000000
max	23.000000	400.000000	85.000000	1.000000

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Transaction ID            0
Date                      0
Time                      0
Type of Card              0
Entry Mode                0
Amount                    0
Type of Transaction       0
Merchant Group            0
Country of Transaction    0
Shipping Address          0
Country of Residence      0
Gender                    0
Age                       0
Bank                      0
Fraud                     0
dtype: int64
```

Data has no null values or disturbances, data looks pretty clean...

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Transaction ID', 'Date', 'Time', 'Type of Card', 'Entry Mode',
       'Amount', 'Type of Transaction', 'Merchant Group',
       'Country of Transaction', 'Shipping Address', 'Country of Residence',
       'Gender', 'Age', 'Bank', 'Fraud'],
      dtype='object')
```

In [7]:
```
df.columns.nunique()
```

Out[7]:
```
15
```

In [8]:
```
df2=df.iloc[:,3:15]
df2.head()
```

Out[8]:

	Type of Card	Entry Mode	Amount	Type of Transaction	Merchant Group	Country of Transaction	Shipping Address	Country of Residence	Gen
0	MasterCard	Tap	17.0	Online	Restaurant	United Kingdom	Russia	United Kingdom	
1	Visa	Tap	28.0	Online	Gaming	United Kingdom	United Kingdom	United Kingdom	
2	Visa	Tap	8.0	Online	Subscription	United Kingdom	United Kingdom	United Kingdom	
3	MasterCard	PIN	186.0	ATM	Entertainment	United Kingdom	United Kingdom	United Kingdom	
4	Visa	PIN	86.0	Online	Children	United Kingdom	United Kingdom	United Kingdom	

In [9]:
```
type(list(df2.columns.unique()))
```

Out[9]:
```
list
```

In [10]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 12 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   Type of Card            100000 non-null  object 
 1   Entry Mode              100000 non-null  object 
 2   Amount                  100000 non-null  float64
 3   Type of Transaction     100000 non-null  object 
 4   Merchant Group          100000 non-null  object 
 5   Country of Transaction  100000 non-null  object 
 6   Shipping Address        100000 non-null  object 
 7   Country of Residence    100000 non-null  object 
 8   Gender                  100000 non-null  object 
 9   Age                     100000 non-null  int64  
 10  Bank                    100000 non-null  object 
 11  Fraud                   100000 non-null  int64  
dtypes: float64(1), int64(2), object(9)
memory usage: 9.2+ MB
```

Used loop to get columns with their unique values

In [11]:
```
for i in range(len(list(df2.columns.unique()))):
        x=list(df2.columns.unique())
        y=df2[x[i]].unique()
        print(x[i],y)
```

```
Type of Card ['MasterCard' 'Visa']
Entry Mode ['Tap' 'PIN']
Amount [ 17.  28.   8. 186.  86.  24. 129.  12. 153. 295.  44. 325.  79.  21.
 113. 370. 123.   5.  47. 151. 244.  16.  23. 213. 335.  25. 305.  15.
 210. 371. 134. 336.  14. 169. 193. 393.  74.   9. 179. 338.  70.  26.
 317.  22.  89.  20.   6. 212. 102.  11. 226. 373.  75. 358. 342.  13.
 156. 360. 392. 248. 149.  19. 351.  10.  96. 361. 110. 108. 137.  39.
  29. 381. 243.  30. 344. 302.   7. 207. 144. 237.  55.  18.  27.  57.
 334. 154. 109. 322. 398. 352. 104. 375. 367. 297. 301. 165. 139. 225.
 131. 189. 115. 242.  76.  61. 346. 206. 276. 219.  38. 382.  97. 331.
 383. 292. 176. 387. 296. 271.  58. 363. 357. 259. 234.  41. 368. 251.
 182. 291.  40.  67. 200. 275. 254.  50.  90. 315. 172.  62.  60. 191.
 204. 333.  95. 282. 262. 253. 356. 209.  54. 340. 314. 232. 195.  32.
 294. 170.  80. 146. 309.  42. 178.  52.  53. 268. 223. 194. 168. 201.
 127.  87. 270. 181. 277. 261. 300. 299.  88. 391. 180. 255. 236. 318.
 400. 269. 235.  35. 196. 174.  48. 319. 321. 274.  65. 293. 198. 284.
 265. 354. 380. 241. 173. 399. 329. 280. 378. 252. 215. 228. 264. 238.
 188. 133. 307. 162. 218.  81. 136. 135. 122. 316. 121. 263. 158. 155.
 199. 217. 143. 266. 247.  82.  36. 203. 171. 308. 374. 272. 384.  31.
 341. 349.  56. 397. 369. 290. 214.  78. 246. 258.  71. 216. 311. 364.
  66. 221. 281. 185.  84. 157. 230. 385. 287. 372. 379. 312.  91. 362.
 303. 202. 107. 327. 231. 138. 166. 117.  43. 260. 359. 278. 119. 285.
  45. 128. 366. 353. 116. 324. 389.  85. 118.  98. 130. 313.  94. 211.
  99. 132. 126. 205. 304.  46. 279. 192. 197. 106. 163. 257. 320. 177.
  49. 240. 345. 224. 150.  72. 310. 267. 161. 388. 124.  68. 100. 229.
 249.  59.  37.  34. 343. 233. 222. 125.  92. 141. 350.  83. 283. 239.
 114. 386. 145. 220. 326. 167. 390. 298. 256. 306. 175. 337. 286. 328.
 147. 120.  93. 396.  51. 190.  64. 273. 148. 208. 289. 103. 227. 140.
 187.  69. 183. 111. 250. 164. 159.  33. 245. 355.  63. 323. 160. 348.
 105. 377. 112. 332. 142. 365. 330. 395.  77. 394. 288. 339. 152. 347.
 101. 376. 184.  73.]
Type of Transaction ['Online' 'ATM' 'POS']
Merchant Group ['Restaurant' 'Gaming' 'Subscription' 'Entertainment' 'Children' 'F
ashion'
 'Electronics' 'Services' 'Food' 'Products']
Country of Transaction ['United Kingdom' 'India' 'China' 'Russia' 'USA']
Shipping Address ['Russia' 'United Kingdom' 'USA' 'India' 'China']
Country of Residence ['United Kingdom' 'China' 'India' 'Russia' 'USA']
Gender ['F' 'M']
Age [36 41 32 65 61 67 21 48 53 78 81 25 47 42 29 71 22 85 52 59 79 37 55 26
 43 33 27 75 38 77 51 68 63 82 70 74 56 84 23 30 24 83 64 80 28 34 40 54
 39 72 46 44 35 49 76 57 66 69 62 31 73 58 50 45 60]
Bank ['Barclays' 'RBS' 'Monzo' 'Lloyds' 'Barlcays' 'Halifax' 'HSBC' 'Metro']
Fraud [0 1]
```

In [12]:
```
df2.shape
```

Out[12]:
```
(100000, 12)
```

Dividing data in dependent and independent variables

In [13]:
```
X=df2.iloc[:,0:11]
X.shape
```

Out[13]:
```
(100000, 11)
```

In [14]:
```
y = df.iloc[:, -1]
y.shape
```

Out[14]:
```
(100000,)
```

In [15]:
```
X.head()
```

Out[15]:

	Type of Card	Entry Mode	Amount	Type of Transaction	Merchant Group	Country of Transaction	Shipping Address	Country of Residence	Gen
0	MasterCard	Tap	17.0	Online	Restaurant	United Kingdom	Russia	United Kingdom	
1	Visa	Tap	28.0	Online	Gaming	United Kingdom	United Kingdom	United Kingdom	
2	Visa	Tap	8.0	Online	Subscription	United Kingdom	United Kingdom	United Kingdom	
3	MasterCard	PIN	186.0	ATM	Entertainment	United Kingdom	United Kingdom	United Kingdom	
4	Visa	PIN	86.0	Online	Children	United Kingdom	United Kingdom	United Kingdom	

Feature engineering--> Encoding categorical values, OneHotEncoded

In [16]:
```
x=pd.get_dummies(X)
x.head()
```

Out[16]:

	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Type of Transaction_ATM	Transactio
0	17.0	36	1	0	0	1	0	
1	28.0	41	0	1	0	1	0	
2	8.0	32	0	1	0	1	0	
3	186.0	65	1	0	1	0	1	
4	86.0	61	0	1	1	0	0	

5 rows × 44 columns

Feature engineering--> Scaling down the values to bring between 0 to 1

In [17]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [18]:
```
scaler=MinMaxScaler()
```

In [19]:
```
x[["Amount","Age"]]=scaler.fit_transform(x[["Amount","Age"]])
```

In [20]:
```
x.describe()
```

Out[20]:

	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.00000	100000.00000
mean	0.272320	0.501275	0.461880	0.538120	0.50163	0.49837
std	0.312477	0.292851	0.498547	0.498547	0.50000	0.50000
min	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000
25%	0.030380	0.250000	0.000000	0.000000	0.00000	0.00000
50%	0.063291	0.500000	0.000000	1.000000	1.00000	0.00000
75%	0.513924	0.750000	1.000000	1.000000	1.00000	1.00000
max	1.000000	1.000000	1.000000	1.000000	1.00000	1.00000

8 rows × 44 columns

In [22]:
```
y.shape
```

Out[22]:
```
(100000,)
```

In [23]:
```
y_2=pd.DataFrame({"Fraud": y})
y_2.shape
```

Out[23]:
```
(100000, 1)
```

In [24]:
```
df_new=pd.concat([x,y_2],axis=1)
```

Saving cleaned file as a backup

In [25]:
```
df_new.to_csv("Scaled,Encoded Credit card Fraud detection full dataset.csv")
```

Importing Basic Libraries Pandas,Numpy and Data file to start with

In [32]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [33]:
```
df=pd.read_csv("Scaled,Encoded Credit card Fraud detection full dataset.csv")
df.head(1)
```

Out[33]:

	Unnamed: 0	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Ty Transaction_
0	0	0.03038	0.234375	1	0	0	1	

1 rows × 46 columns

In [34]:
```
df=df.iloc[:,1:]
df.head(1)
```

Out[34]:

	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Type of Transaction_ATM	Trans
0	0.03038	0.234375	1	0	0	1	0	

1 rows × 45 columns

Verifying all datasets for Application of Machine Learning Algorithms

In [35]:
```
df.describe()
```

Out[35]:

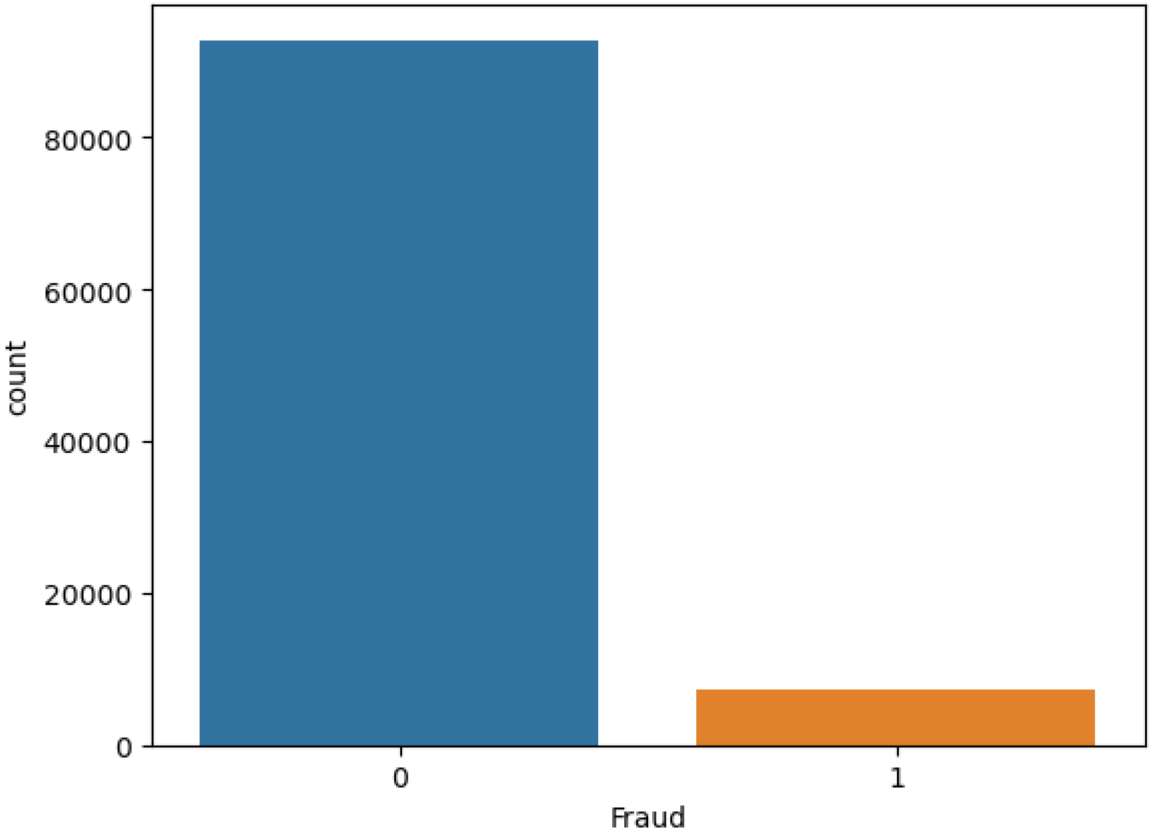
	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.00000	100000.00000
mean	0.272320	0.501275	0.461880	0.538120	0.50163	0.49837
std	0.312477	0.292851	0.498547	0.498547	0.50000	0.50000
min	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000
25%	0.030380	0.250000	0.000000	0.000000	0.00000	0.00000
50%	0.063291	0.500000	0.000000	1.000000	1.00000	0.00000
75%	0.513924	0.750000	1.000000	1.000000	1.00000	1.00000
max	1.000000	1.000000	1.000000	1.000000	1.00000	1.00000

8 rows × 45 columns

It is observed data is highly biased on Non-Fraud side

In [36]:
```
sns.countplot(df.iloc[:,-1])
```

Out[36]: <AxesSubplot:xlabel='Fraud', ylabel='count'>



Understanding inter-relationships

In [82]:
```
df.corr()
```

Out[82]:

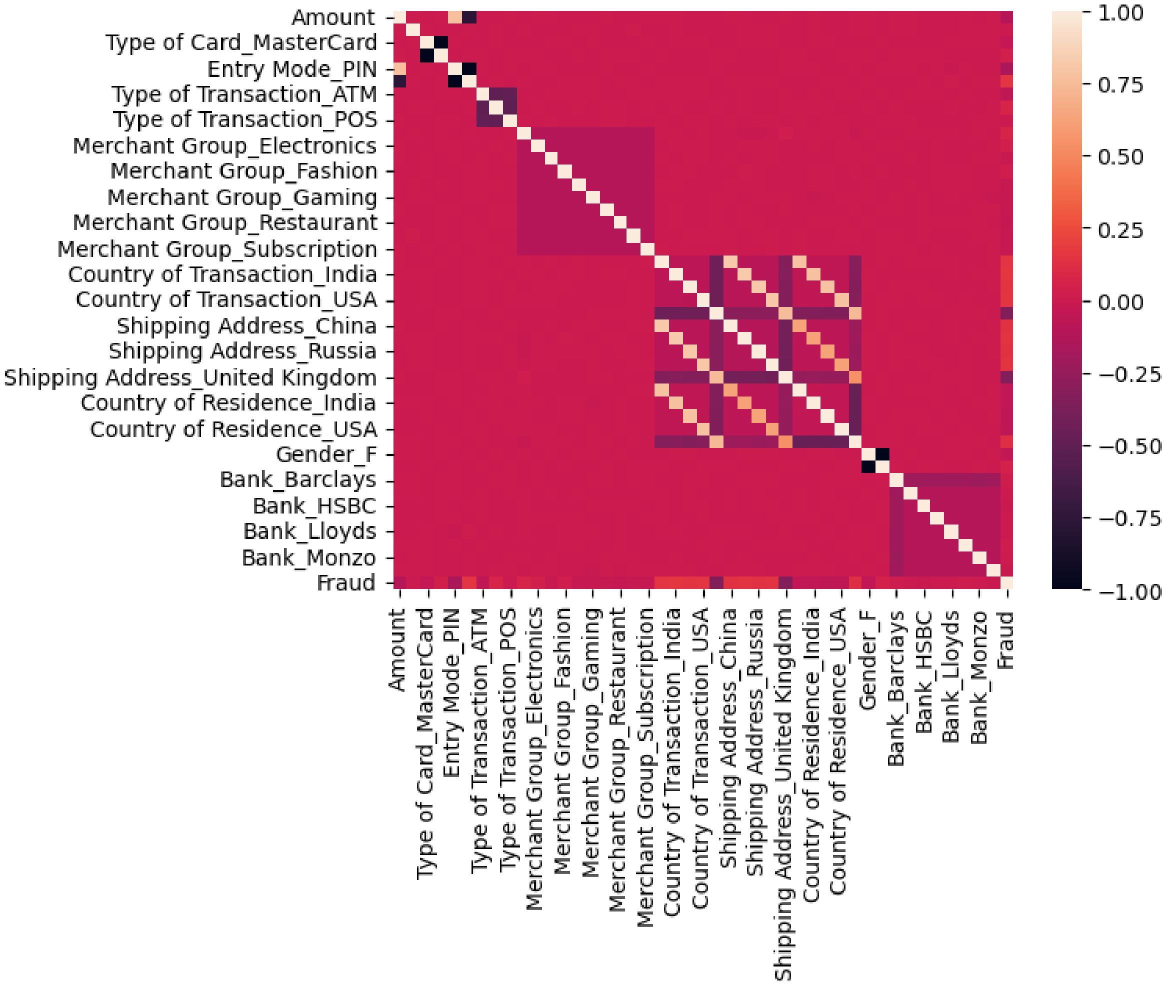
	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Tr
Amount	1.000000	0.001335	-0.001359	0.001359	0.767967	-0.767967	
Age	0.001335	1.000000	0.006618	-0.006618	-0.003054	0.003054	
Type of Card_MasterCard	-0.001359	0.006618	1.000000	-1.000000	0.000871	-0.000871	
Type of Card_Visa	0.001359	-0.006618	-1.000000	1.000000	-0.000871	0.000871	
Entry Mode_PIN	0.767967	-0.003054	0.000871	-0.000871	1.000000	-1.000000	
Entry Mode_Tap	-0.767967	0.003054	-0.000871	0.000871	-1.000000	1.000000	
Type of Transaction_ATM	-0.000660	-0.000039	-0.006430	0.006430	0.004680	-0.004680	
Type of Transaction_Online	-0.000855	0.000139	0.006772	-0.006772	-0.005661	0.005661	
Type of Transaction_POS	0.001517	-0.000100	-0.000352	0.000352	0.000990	-0.000990	
Merchant Group_Children	-0.004026	0.001258	-0.000412	0.000412	-0.005369	0.005369	
Merchant Group_Electronics	0.002302	-0.001000	-0.002517	0.002517	0.002746	-0.002746	
Merchant Group_Entertainment	0.004304	-0.001700	0.000141	-0.000141	0.004362	-0.004362	
Merchant Group_Fashion	-0.000780	-0.006501	0.004634	-0.004634	-0.002387	0.002387	
Merchant Group_Food	-0.000314	0.006188	-0.000035	0.000035	-0.001279	0.001279	
Merchant Group_Gaming	-0.003716	0.002517	0.000979	-0.000979	-0.002087	0.002087	
Merchant Group_Products	0.001828	-0.002453	-0.001407	0.001407	0.002146	-0.002146	
Merchant Group_Restaurant	-0.004744	-0.003973	-0.001188	0.001188	-0.001091	0.001091	
Merchant Group_Services	-0.002169	0.008175	-0.002382	0.002382	-0.003786	0.003786	
Merchant Group_Subscription	0.007533	-0.002548	0.002166	-0.002166	0.006989	-0.006989	
Country of Transaction_China	0.000582	-0.001511	-0.003142	0.003142	-0.000051	0.000051	
Country of Transaction_India	0.000536	0.003716	0.002726	-0.002726	-0.000784	0.000784	
Country of Transaction_Russia	-0.000145	-0.001597	0.005191	-0.005191	-0.002955	0.002955	
Country of Transaction_USA	-0.000617	-0.000479	0.000555	-0.000555	-0.001145	0.001145	
Country of Transaction_United Kingdom	-0.000198	-0.000057	-0.003050	0.003050	0.002824	-0.002824	

	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Tr
Shipping Address_China	0.000227	-0.001471	-0.005167	0.005167	-0.001677	0.001677	
Shipping Address_India	0.000714	0.004704	0.004166	-0.004166	-0.000175	0.000175	
Shipping Address_Russia	-0.000982	0.000373	0.001689	-0.001689	-0.004404	0.004404	
Shipping Address_USA	-0.000373	0.000274	0.001260	-0.001260	-0.003817	0.003817	
Shipping Address_United Kingdom	0.000261	-0.002338	-0.001181	0.001181	0.006157	-0.006157	
Country of Residence_China	-0.000338	-0.005690	-0.003652	0.003652	0.000685	-0.000685	
Country of Residence_India	0.001364	0.004417	0.002380	-0.002380	0.000811	-0.000811	
Country of Residence_Russia	-0.000980	-0.005227	0.006216	-0.006216	-0.004720	0.004720	
Country of Residence_USA	-0.004759	-0.000377	-0.000099	0.000099	-0.004251	0.004251	
Country of Residence_United Kingdom	0.002575	0.003770	-0.002628	0.002628	0.004078	-0.004078	
Gender_F	0.000297	0.007241	-0.000158	0.000158	-0.001383	0.001383	
Gender_M	-0.000297	-0.007241	0.000158	-0.000158	0.001383	-0.001383	
Bank_Barclays	0.004839	0.003984	-0.001967	0.001967	0.002411	-0.002411	
Bank_Barlcays	0.005233	0.001865	-0.001309	0.001309	0.003348	-0.003348	
Bank_HSBC	-0.005702	-0.004560	-0.001707	0.001707	-0.001752	0.001752	
Bank_Halifax	-0.003191	-0.003549	-0.001355	0.001355	0.003517	-0.003517	
Bank_Lloyds	-0.005398	-0.004742	0.003877	-0.003877	-0.008629	0.008629	
Bank_Metro	0.005581	0.000053	0.001185	-0.001185	0.004243	-0.004243	
Bank_Monzo	-0.005151	0.001004	0.000973	-0.000973	-0.004112	0.004112	
Bank_RBS	0.001239	0.003818	0.001331	-0.001331	-0.000291	0.000291	
Fraud	-0.110096	-0.000867	-0.034703	0.034703	-0.152347	0.152347	

In [37]:

```
sns.heatmap(df.corr())
```

Out[37]: <AxesSubplot:>



Divided the data into x and y as dependent and independent variables

In [38]:
```
y=df.iloc[:,-1]
x=df.iloc[:,:-1]
x.shape,y.shape
```

Out[38]:
```
((100000, 44), (100000,))
```

Spliiting dataset and keeping 30% as testing data

In [39]:
```
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.3,random_state=10)
```

In [40]:
```
x_train.shape,y_test.shape
```

Out[40]:
```
((70000, 44), (30000,))
```

Applying Logistic Algorithm -ML

In [41]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

Out[41]:
```
LogisticRegression()
```

In [42]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

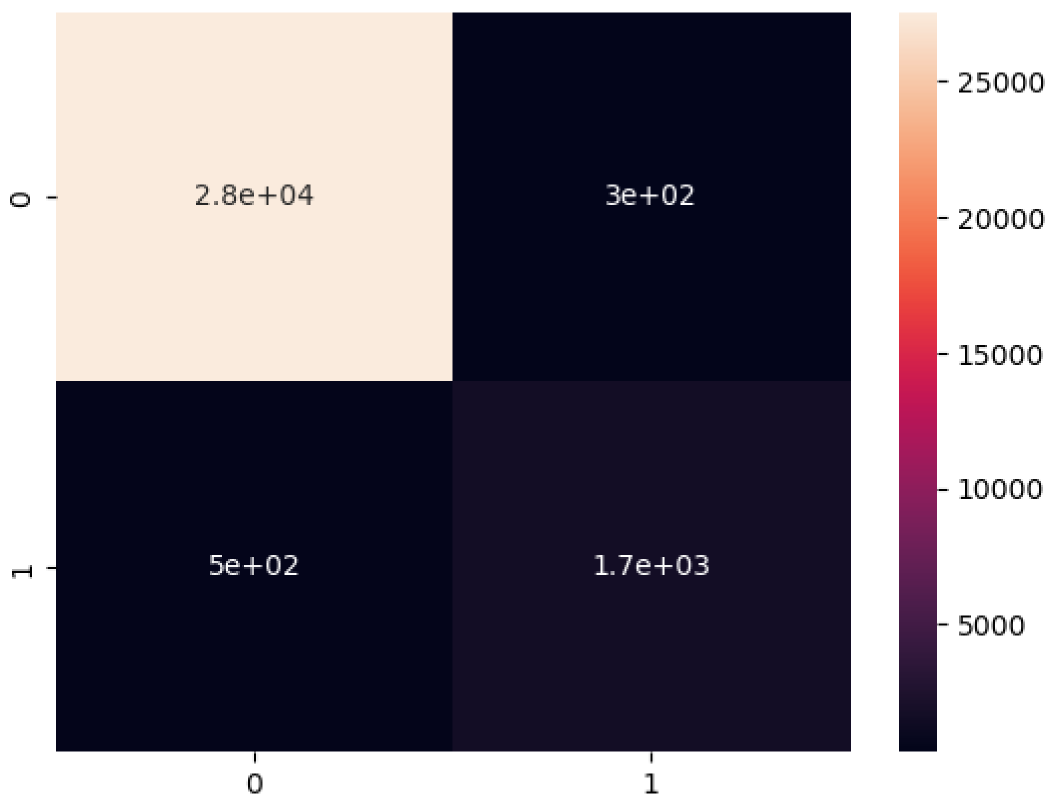
In [43]:
```
model.score(x_train,y_train)
```
Out[43]:
```
0.9715142857142857
```

In [44]:
```
model.score(x_test,y_test)
```
Out[44]:
```
0.9735333333333334
```

In [45]:
```
from sklearn.metrics import confusion_matrix
```

In [46]:
```
con=confusion_matrix(y_test,y_pred)
```

In [47]:
```
sns.heatmap(con,annot=True)
```
Out[47]:
```
<AxesSubplot:>
```



In [48]:
```
print(classification_report(y_test,y_pred))
```
```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     27826
           1       0.85      0.77      0.81      2174

    accuracy                           0.97     30000
   macro avg       0.92      0.88      0.90     30000
weighted avg       0.97      0.97      0.97     30000
```

Applying Decision Tree Classifier Algorithm- ML

In [49]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix
```

In [50]:
```
model=DecisionTreeClassifier()
```

In [51]:
```
model.fit(x_train,y_train)
```

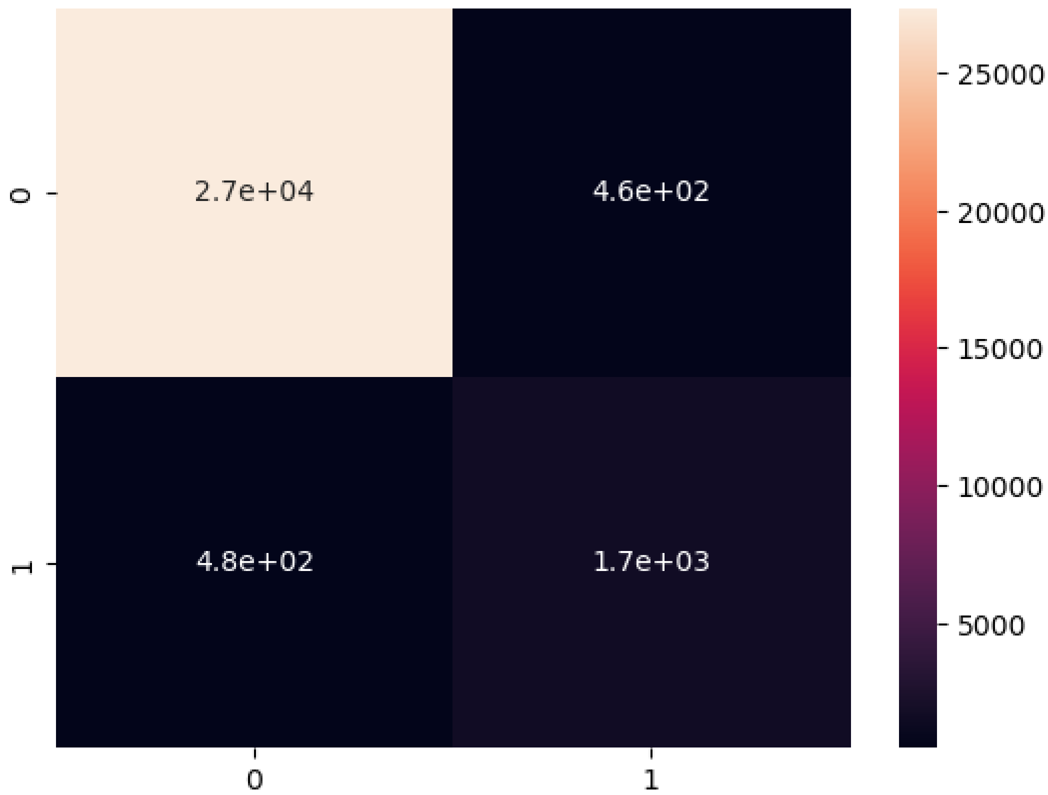
Out[51]:
```
DecisionTreeClassifier()
```

In [52]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

In [53]:
```
con=confusion_matrix(y_pred,y_test)
sns.heatmap(con,annot=True)
```

Out[53]:
```
<AxesSubplot:>
```



In [54]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     27812
           1       0.79      0.78      0.78      2188

    accuracy                           0.97     30000
   macro avg       0.88      0.88      0.88     30000
weighted avg       0.97      0.97      0.97     30000
```

Applying Random Forest Classifier Algorithm-ML

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
model=RandomForestClassifier()
```

In [57]:
```
model.fit(x_train,y_train)
```

Out[57]:
```
RandomForestClassifier()
```

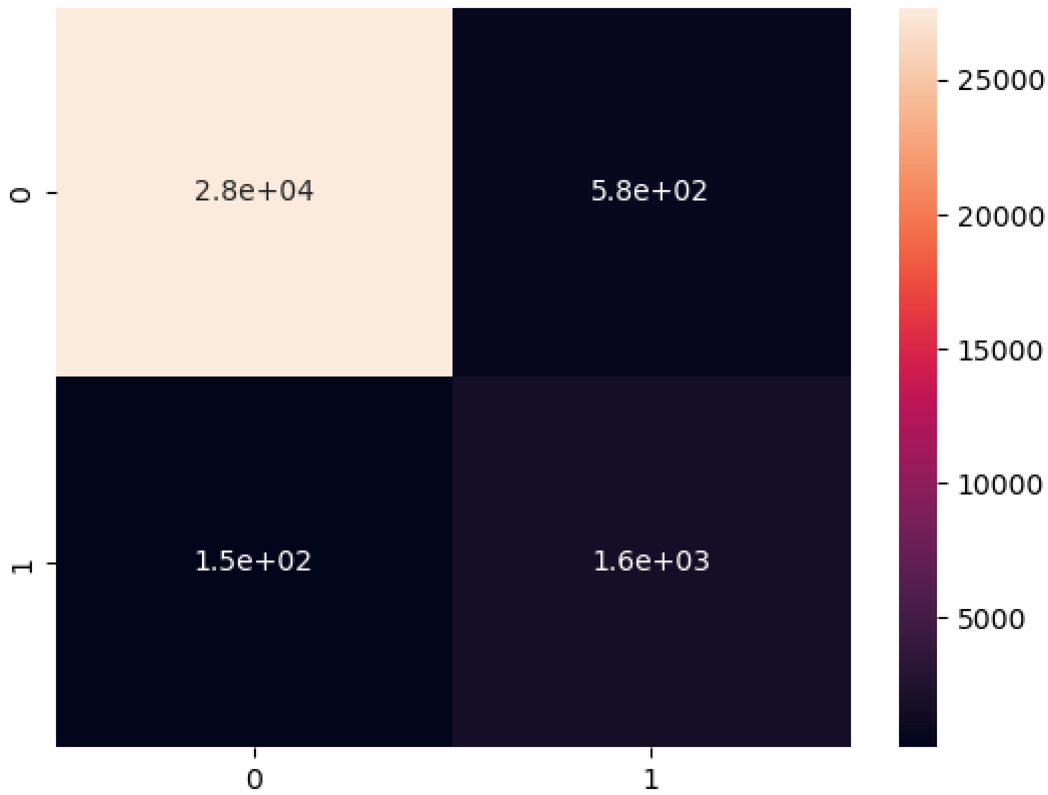
In [58]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

In [59]:
```
con=confusion_matrix(y_pred,y_test)
```

In [42]:
```
sns.heatmap(con,annot=True)
```

Out[42]:
```
<AxesSubplot:>
```



In [60]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99     28268
           1       0.73      0.92      0.81      1732

    accuracy                           0.98     30000
   macro avg       0.86      0.95      0.90     30000
weighted avg       0.98      0.98      0.98     30000
```

In [61]:
```
x_test.head()
```

Out[61]:

	Amount	Age	Type of Card_MasterCard	Type of Card_Visa	Entry Mode_PIN	Entry Mode_Tap	Type of Transaction_ATM	T
33226	0.577215	0.03125	1	0	1	0	0	
64804	0.020253	0.93750	1	0	0	1	0	
39763	0.177215	0.59375	1	0	1	0	0	
51270	0.235443	0.71875	0	1	1	0	0	
9698	0.060759	0.25000	1	0	0	1	1	

5 rows × 44 columns

Applying Support Vector Machine Classifier Algorithm-ML

In [62]:
```
from sklearn.svm import SVC
```

In [63]:
```
model=SVC()
```

In [64]:
```
model.fit(x_train,y_train)
```

Out[64]:
```
SVC()
```

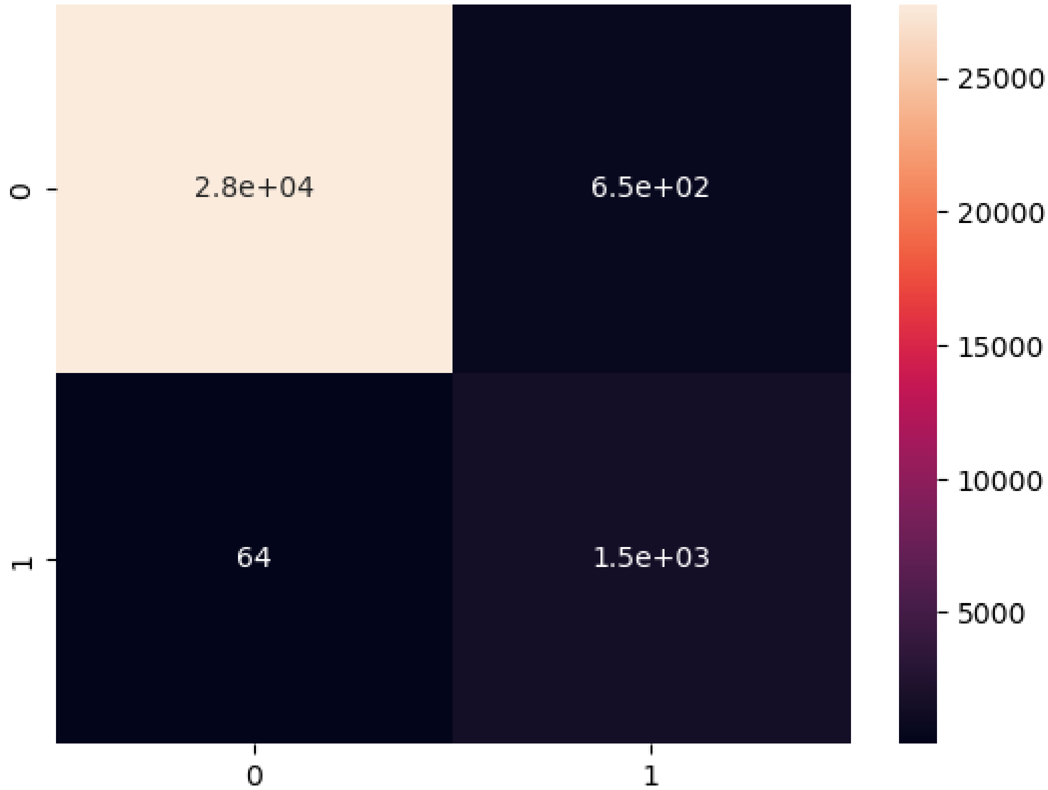
In [65]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

In [66]:
```
con=confusion_matrix(y_pred,y_test)
```

In [67]:
```
sns.heatmap(con,annot=True)
```

Out[67]:
```
<AxesSubplot:>
```



In [68]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     28409
           1       0.70      0.96      0.81      1591

    accuracy                           0.98     30000
   macro avg       0.85      0.97      0.90     30000
weighted avg       0.98      0.98      0.98     30000
```

Applying Naive Bayes Algorithm-ML

In [69]:
```
from sklearn.naive_bayes import GaussianNB
```

In [70]:
```
model=GaussianNB()
```

In [71]:
```
model.fit(x_train,y_train)
```

Out[71]:
```
GaussianNB()
```

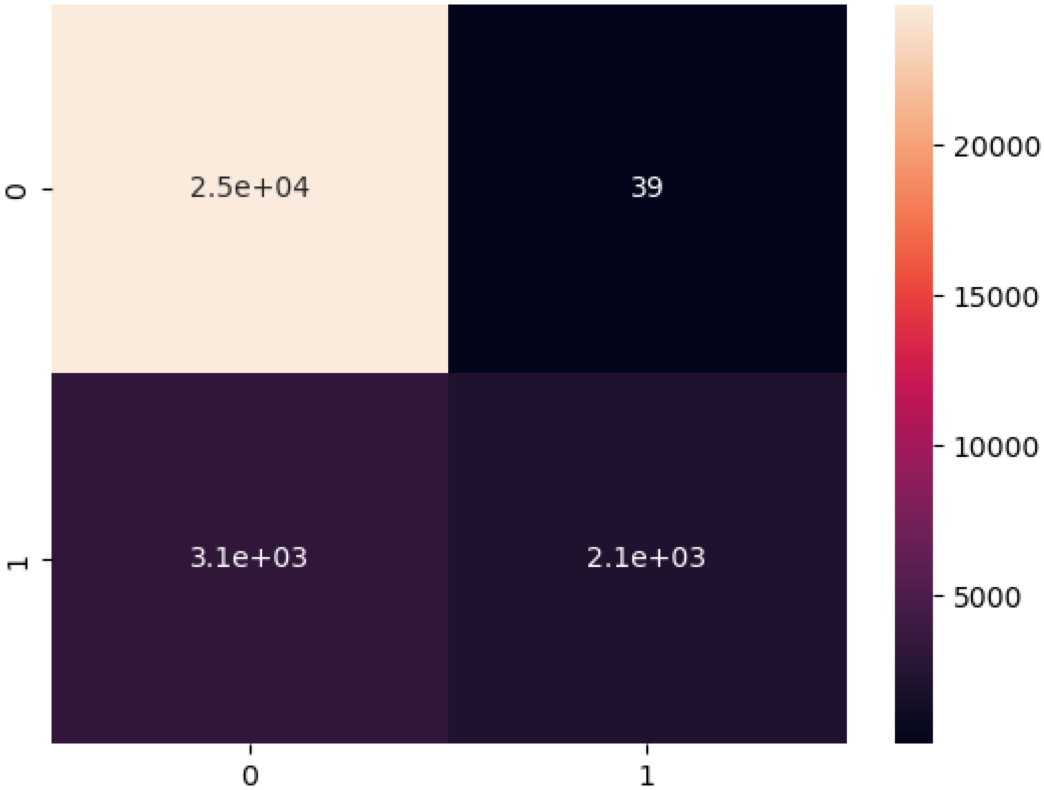
In [72]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

In [73]:
```
con=confusion_matrix(y_pred,y_test)
```

In [74]:
```
sns.heatmap(con,annot=True)
```

Out[74]:
```
<AxesSubplot:>
```



In [75]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     24726
           1       0.98      0.40      0.57      5274

    accuracy                           0.89     30000
   macro avg       0.93      0.70      0.76     30000
weighted avg       0.90      0.89      0.88     30000
```

Applying K nearest neighbours Classifier Algorithm -ML

In [76]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [77]:
```
model=KNeighborsClassifier()
```

In [78]:
```
model.fit(x_train,y_train)
```

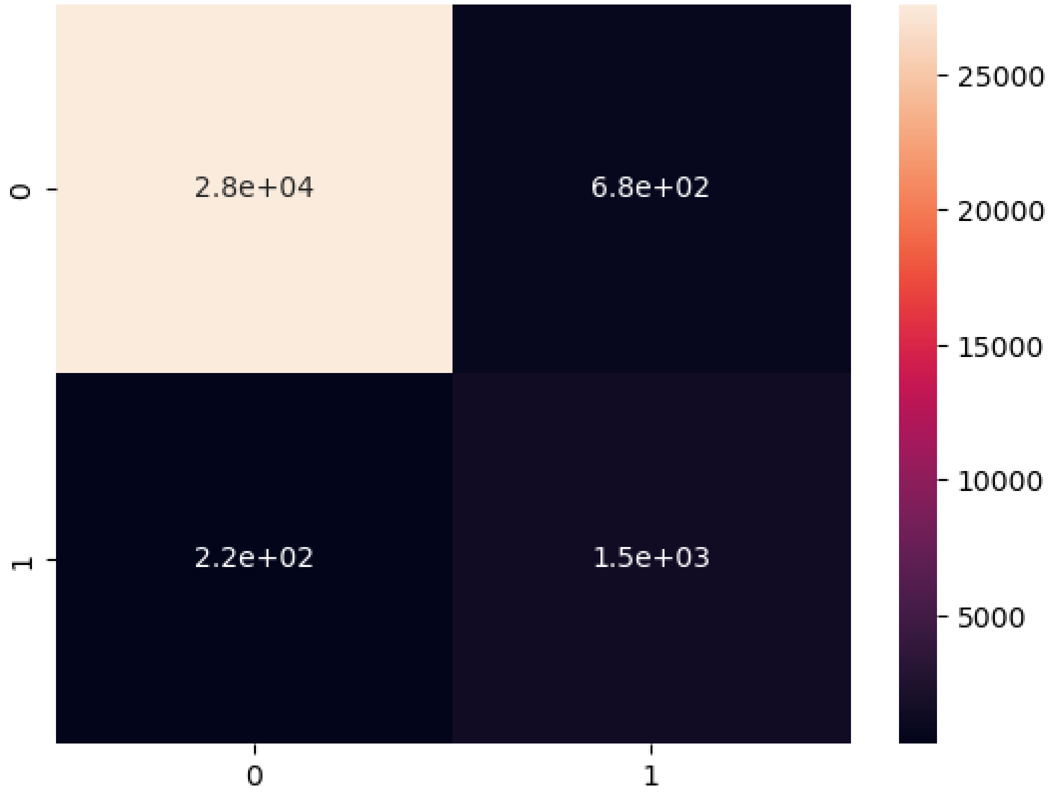
Out[78]:
```
KNeighborsClassifier()
```

In [79]:
```
y_pred=model.predict(x_test)
```

Checking accuracies for predicted model with, confusion matrix & classification report

In [80]:
```
con=confusion_matrix(y_pred,y_test)
sns.heatmap(con,annot=True)
```

Out[80]:
```
<AxesSubplot:>
```



In [81]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98     28286
           1       0.69      0.87      0.77      1714

    accuracy                           0.97     30000
   macro avg       0.84      0.92      0.88     30000
weighted avg       0.97      0.97      0.97     30000
```

Insights/ Outcomes from the Activity

1. Here models have less data for training Fraud Detected("1")
2. Almost all the models are giving good precision, recall and f1-score as they are having good amount of data
3. From comaprison of Recall score Random forest proved the best algorithm of all. Logistic regression gave 0.77 recall, decision tree gave 0.78 recall and Random forest gave 0.92 recall, where all three algorithms gave almost equal F1 score.
4. Naive bayes model is giving least type 2 error amount. But, this is not enough, we should also focus on its classification report. Its F1-score for naive bayes for Frauds is just 0.57. Naive bayes model is cleary not working accurately in this dataset, as expected.
5. SVM and KNN did good in terms of recall score but they lead to decrease precision values.

In [ ]: