Credit Card Fraud Detection Model

By DAJAH VINCENT

In [2]:
```
#Importing data manipulation and visualization libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
```

In [3]:
```
#reading the creditcard dataset

creditCard = pd.read_csv("creditcard.csv")
```

In [4]:
```
#viewing the structure of the creditCard datset

creditCard
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.9
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.1
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.3

284807 rows × 31 columns

Exploratory Data Analysis

In [5]:
```
#viewing the data first 5 rows from the creditcard dataset

creditCard.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.3
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.6
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.1

5 rows × 31 columns

In [6]:
```
#viewing the last 5 rows from the creditcard dataset

creditCard.tail()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376

5 rows × 31 columns

In [7]:
```
creditCard.columns
```

Out[7]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [8]:
```
# I wanted to see the description of the dataset while rounding up the floated fraction to 2 decimals.
#I also transpose the result to be able to see the entire columns

round(creditCard.describe(), 2).T

#creditCard.round(2)
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Time	284807.0	94813.86	47488.15	0.00	54201.50	84692.00	139320.50	172792.00
V1	284807.0	0.00	1.96	-56.41	-0.92	0.02	1.32	2.45
V2	284807.0	0.00	1.65	-72.72	-0.60	0.07	0.80	22.06
V3	284807.0	-0.00	1.52	-48.33	-0.89	0.18	1.03	9.38
V4	284807.0	0.00	1.42	-5.68	-0.85	-0.02	0.74	16.88
V5	284807.0	0.00	1.38	-113.74	-0.69	-0.05	0.61	34.80
V6	284807.0	0.00	1.33	-26.16	-0.77	-0.27	0.40	73.30
V7	284807.0	-0.00	1.24	-43.56	-0.55	0.04	0.57	120.59
V8	284807.0	0.00	1.19	-73.22	-0.21	0.02	0.33	20.01
V9	284807.0	-0.00	1.10	-13.43	-0.64	-0.05	0.60	15.59
V10	284807.0	0.00	1.09	-24.59	-0.54	-0.09	0.45	23.75
V11	284807.0	0.00	1.02	-4.80	-0.76	-0.03	0.74	12.02
V12	284807.0	-0.00	1.00	-18.68	-0.41	0.14	0.62	7.85
V13	284807.0	0.00	1.00	-5.79	-0.65	-0.01	0.66	7.13
V14	284807.0	0.00	0.96	-19.21	-0.43	0.05	0.49	10.53
V15	284807.0	0.00	0.92	-4.50	-0.58	0.05	0.65	8.88
V16	284807.0	0.00	0.88	-14.13	-0.47	0.07	0.52	17.32
V17	284807.0	-0.00	0.85	-25.16	-0.48	-0.07	0.40	9.25
V18	284807.0	0.00	0.84	-9.50	-0.50	-0.00	0.50	5.04
V19	284807.0	0.00	0.81	-7.21	-0.46	0.00	0.46	5.59
V20	284807.0	0.00	0.77	-54.50	-0.21	-0.06	0.13	39.42
V21	284807.0	0.00	0.73	-34.83	-0.23	-0.03	0.19	27.20
V22	284807.0	-0.00	0.73	-10.93	-0.54	0.01	0.53	10.50
V23	284807.0	0.00	0.62	-44.81	-0.16	-0.01	0.15	22.53
V24	284807.0	0.00	0.61	-2.84	-0.35	0.04	0.44	4.58
V25	284807.0	0.00	0.52	-10.30	-0.32	0.02	0.35	7.52
V26	284807.0	0.00	0.48	-2.60	-0.33	-0.05	0.24	3.52
V27	284807.0	-0.00	0.40	-22.57	-0.07	0.00	0.09	31.61
V28	284807.0	-0.00	0.33	-15.43	-0.05	0.01	0.08	33.85
Amount	284807.0	88.35	250.12	0.00	5.60	22.00	77.16	25691.16
Class	284807.0	0.00	0.04	0.00	0.00	0.00	0.00	1.00

In [9]:
```python
#checking for any NaN or null values in the columns of the dataset

creditCard.isna().sum()
```

Out[9]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [10]:
```python
#checking the number of duplicated values in the datsets

creditCard.duplicated().sum()
```

Out[10]:
```
1081
```

In [11]:
```python
#dropping or deleting the duplicated data values

creditCard.drop_duplicates()
```

Out[11]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.9
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.1
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.3

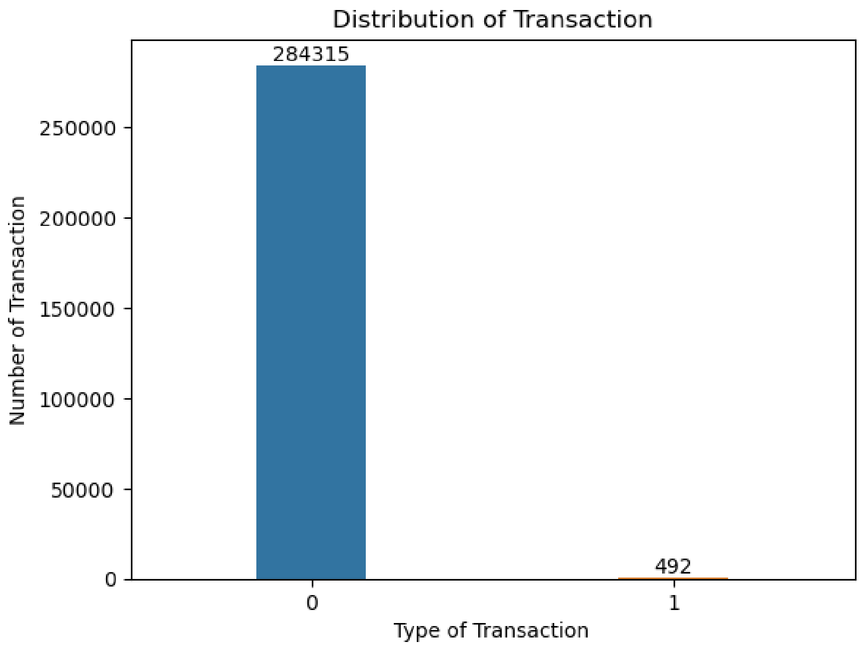
283726 rows × 31 columns

In [12]:
```python
#Checking the class of identified legit and fraudulent transactions

creditCard["Class"].value_counts()
```

Out[12]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

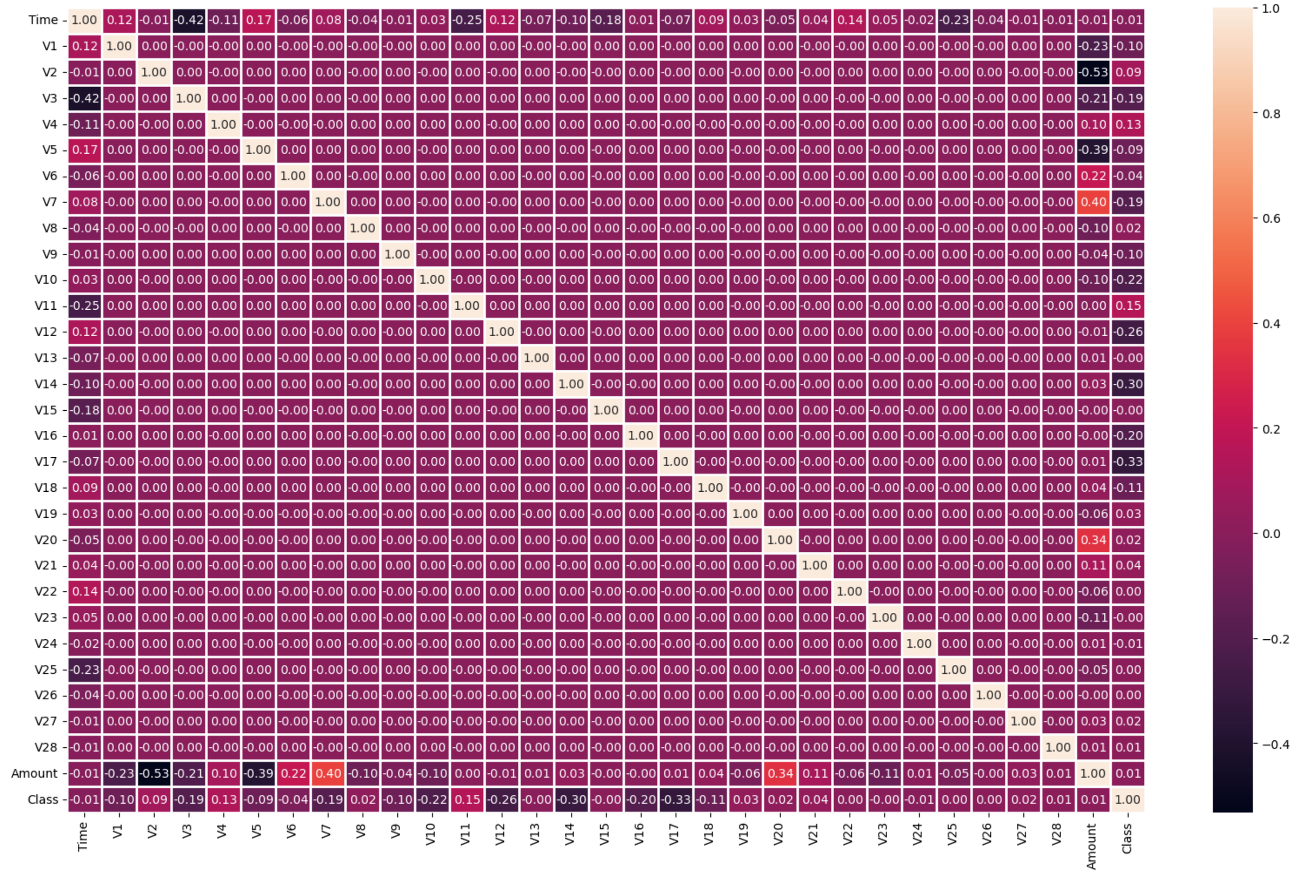
Creating Visualization

In [13]:
```python
ax = sb.countplot(data = creditCard, x = "Class", width = 0.3)
ax.set_title("Distribution of Transaction")
plt.xlabel("Type of Transaction")
plt.ylabel("Number of Transaction")


for i in ax.containers:
    ax.bar_label(i)

plt.show()
```



In [14]:

```python
plt.figure(figsize = (20, 12))
ax = sb.heatmap(creditCard.corr(), annot = True, fmt = '.2f')

for i in range(creditCard.shape[1] + 1):
    ax.axvline(i, color='white', lw = 2)
    ax.axhline(i, color='white', lw = 2)
#plt.tight_layout()
plt.show()
```



Normalizing the legitimate and fraudeulent data

In [15]:

```python
#create a function that seperates the class of transaction between fraud and legit transactions

def split_data_by_class(creditCard):

    legit = creditCard[creditCard["Class"] == 0]
    fraud = creditCard[creditCard["Class"] == 1]
    return legit, fraud

# Example usage:
legit_df, fraud_df = split_data_by_class(creditCard)
```

In [16]:
```
legit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 284315 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284315 non-null  float64
 1   V1      284315 non-null  float64
 2   V2      284315 non-null  float64
 3   V3      284315 non-null  float64
 4   V4      284315 non-null  float64
 5   V5      284315 non-null  float64
 6   V6      284315 non-null  float64
 7   V7      284315 non-null  float64
 8   V8      284315 non-null  float64
 9   V9      284315 non-null  float64
 10  V10     284315 non-null  float64
 11  V11     284315 non-null  float64
 12  V12     284315 non-null  float64
 13  V13     284315 non-null  float64
 14  V14     284315 non-null  float64
 15  V15     284315 non-null  float64
 16  V16     284315 non-null  float64
 17  V17     284315 non-null  float64
 18  V18     284315 non-null  float64
 19  V19     284315 non-null  float64
 20  V20     284315 non-null  float64
 21  V21     284315 non-null  float64
 22  V22     284315 non-null  float64
 23  V23     284315 non-null  float64
 24  V24     284315 non-null  float64
 25  V25     284315 non-null  float64
 26  V26     284315 non-null  float64
 27  V27     284315 non-null  float64
 28  V28     284315 non-null  float64
 29  Amount  284315 non-null  float64
 30  Class   284315 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 69.4 MB
```

In [17]:
```
fraud_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 492 entries, 541 to 281674
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    492 non-null    float64
 1   V1      492 non-null    float64
 2   V2      492 non-null    float64
 3   V3      492 non-null    float64
 4   V4      492 non-null    float64
 5   V5      492 non-null    float64
 6   V6      492 non-null    float64
 7   V7      492 non-null    float64
 8   V8      492 non-null    float64
 9   V9      492 non-null    float64
 10  V10     492 non-null    float64
 11  V11     492 non-null    float64
 12  V12     492 non-null    float64
 13  V13     492 non-null    float64
 14  V14     492 non-null    float64
 15  V15     492 non-null    float64
 16  V16     492 non-null    float64
 17  V17     492 non-null    float64
 18  V18     492 non-null    float64
 19  V19     492 non-null    float64
 20  V20     492 non-null    float64
 21  V21     492 non-null    float64
 22  V22     492 non-null    float64
 23  V23     492 non-null    float64
 24  V24     492 non-null    float64
 25  V25     492 non-null    float64
 26  V26     492 non-null    float64
 27  V27     492 non-null    float64
 28  V28     492 non-null    float64
 29  Amount  492 non-null    float64
 30  Class   492 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 123.0 KB
```

In [18]:
```
legit_df.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%	max
Time	284315.0	94838.202258	47484.015786	0.000000	54230.000000	84711.000000	139333.000000	172792.000000
V1	284315.0	0.008258	1.929814	-56.407510	-0.917544	0.020023	1.316218	2.454930
V2	284315.0	-0.006271	1.636146	-72.715728	-0.599473	0.064070	0.800446	18.902453
V3	284315.0	0.012171	1.459429	-48.325589	-0.884541	0.182158	1.028372	9.382558
V4	284315.0	-0.007860	1.399333	-5.683171	-0.850077	-0.022405	0.737624	16.875344
V5	284315.0	0.005453	1.356952	-113.743307	-0.689398	-0.053457	0.612181	34.801666
V6	284315.0	0.002419	1.329913	-26.160506	-0.766847	-0.273123	0.399619	73.301626
V7	284315.0	0.009637	1.178812	-31.764946	-0.551442	0.041138	0.571019	120.589494
V8	284315.0	-0.000987	1.161283	-73.216718	-0.208633	0.022041	0.326200	18.709255
V9	284315.0	0.004467	1.089372	-6.290730	-0.640412	-0.049964	0.598230	15.594995
V10	284315.0	0.009824	1.044204	-14.741096	-0.532880	-0.091872	0.455135	23.745136
V11	284315.0	-0.006576	1.003112	-4.797473	-0.763447	-0.034923	0.736362	10.002190
V12	284315.0	0.010832	0.945939	-15.144988	-0.402102	0.141679	0.619207	7.848392
V13	284315.0	0.000189	0.995067	-5.791881	-0.648067	-0.013547	0.662492	7.126883
V14	284315.0	0.012064	0.897007	-18.392091	-0.422453	0.051947	0.494104	10.526766
V15	284315.0	0.000161	0.915060	-4.391307	-0.582812	0.048294	0.648842	8.877742
V16	284315.0	0.007164	0.844772	-10.115560	-0.465543	0.067377	0.523738	17.315112
V17	284315.0	0.011535	0.749457	-17.098444	-0.482644	-0.064833	0.399922	9.253526
V18	284315.0	0.003887	0.824919	-5.366660	-0.497414	-0.002787	0.501103	5.041069
V19	284315.0	-0.001178	0.811733	-7.213527	-0.456366	0.003117	0.457499	5.591971
V20	284315.0	-0.000644	0.769404	-54.497720	-0.211764	-0.062646	0.132401	39.420904
V21	284315.0	-0.001235	0.716743	-34.830382	-0.228509	-0.029821	0.185626	22.614889
V22	284315.0	-0.000024	0.723668	-10.933144	-0.542403	0.006736	0.528407	10.503090
V23	284315.0	0.000070	0.621541	-44.807735	-0.161702	-0.011147	0.147522	22.528412
V24	284315.0	0.000182	0.605776	-2.836627	-0.354425	0.041082	0.439869	4.584549
V25	284315.0	-0.000072	0.520673	-10.295397	-0.317145	0.016417	0.350594	7.519589
V26	284315.0	-0.000089	0.482241	-2.604551	-0.327074	-0.052227	0.240671	3.517346
V27	284315.0	-0.000295	0.399847	-22.565679	-0.070852	0.001230	0.090573	31.612198
V28	284315.0	-0.000131	0.329570	-15.430084	-0.052950	0.011199	0.077962	33.847808
Amount	284315.0	88.291022	250.105092	0.000000	5.650000	22.000000	77.050000	25691.160000
Class	284315.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

In [19]:
```
fraud_df.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
Time	492.0	80746.806911	47835.365138	406.000000	41241.500000	75568.500000	128483.000000	170348.000000
V1	492.0	-4.771948	6.783687	-30.552380	-6.036063	-2.342497	-0.419200	2.132386
V2	492.0	3.623778	4.291216	-8.402154	1.188226	2.717869	4.971257	22.057729
V3	492.0	-7.033281	7.110937	-31.103685	-8.643489	-5.075257	-2.276185	2.250210
V4	492.0	4.542029	2.873318	-1.313275	2.373050	4.177147	6.348729	12.114672
V5	492.0	-3.151225	5.372468	-22.105532	-4.792835	-1.522962	0.214562	11.095089
V6	492.0	-1.397737	1.858124	-6.406267	-2.501511	-1.424616	-0.413216	6.474115
V7	492.0	-5.568731	7.206773	-43.557242	-7.965295	-3.034402	-0.945954	5.802537
V8	492.0	0.570636	6.797831	-41.044261	-0.195336	0.621508	1.764879	20.007208
V9	492.0	-2.581123	2.500896	-13.434066	-3.872383	-2.208768	-0.787850	3.353525
V10	492.0	-5.676883	4.897341	-24.588262	-7.756698	-4.578825	-2.614184	4.031435
V11	492.0	3.800173	2.678605	-1.702228	1.973397	3.586218	5.307078	12.018913
V12	492.0	-6.259393	4.654458	-18.683715	-8.688177	-5.502530	-2.974088	1.375941
V13	492.0	-0.109334	1.104518	-3.127795	-0.979117	-0.065566	0.672964	2.815440
V14	492.0	-6.971723	4.278940	-19.214325	-9.692723	-6.729720	-4.282821	3.442422
V15	492.0	-0.092929	1.049915	-4.498945	-0.643539	-0.057227	0.609189	2.471358
V16	492.0	-4.139946	3.865035	-14.129855	-6.562915	-3.549795	-1.226043	3.139656
V17	492.0	-6.665836	6.970618	-25.162799	-11.945057	-5.302949	-1.341940	6.739384
V18	492.0	-2.246308	2.899366	-9.498746	-4.664576	-1.664346	0.091772	3.790316
V19	492.0	0.680659	1.539853	-3.681904	-0.299423	0.646807	1.649318	5.228342
V20	492.0	0.372319	1.346635	-4.128186	-0.171760	0.284693	0.822445	11.059004
V21	492.0	0.713588	3.869304	-22.797604	0.041787	0.592146	1.244611	27.202839
V22	492.0	0.014049	1.494602	-8.887017	-0.533764	0.048434	0.617474	8.361985
V23	492.0	-0.040308	1.579642	-19.254328	-0.342175	-0.073135	0.308378	5.466230
V24	492.0	-0.105130	0.515577	-2.028024	-0.436809	-0.060795	0.285328	1.091435
V25	492.0	0.041449	0.797205	-4.781606	-0.314348	0.088371	0.456515	2.208209
V26	492.0	0.051648	0.471679	-1.152671	-0.259416	0.004321	0.396733	2.745261
V27	492.0	0.170575	1.376766	-7.263482	-0.020025	0.394926	0.826029	3.052358
V28	492.0	0.075667	0.547291	-1.869290	-0.108868	0.146344	0.381152	1.779364
Amount	492.0	122.211321	256.683288	0.000000	1.000000	9.250000	105.890000	2125.870000
Class	492.0	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [20]:
```
#sampling the legit transaction to a match a 492 rows

new_legit_df = legit_df.sample(n = 492)
```

In [21]:
```python
print(new_legit_df)
```

```
              Time        V1        V2        V3        V4        V5        V6  \
278637  168331.0  1.924745  0.476715 -0.972502  3.488065  0.895899  0.768858
191786  129409.0 -0.775259  0.435444  0.797722 -2.198371 -0.492238 -0.918507
182166  125289.0 -1.169186 -1.106668 -0.302463 -3.042068  2.134270  0.978630
211541  138458.0 -0.636187 -0.617568  0.771143  0.043623 -0.922183 -0.018679
195395  131031.0  1.882015  0.456126  0.091758  3.660667  0.211950  0.702324
...          ...       ...       ...       ...       ...       ...       ...
273013  165377.0 -3.617360  1.760947 -3.127011 -0.728001 -0.220947  1.096116
186375  127077.0  2.068718 -1.300177 -1.540357 -2.212075 -0.487425 -0.080340
14116    25114.0  1.090593 -0.054415  1.214231  1.844288 -0.427767  0.866735
84091    60158.0 -3.327739  0.519052  0.163019  2.461758  0.225975  1.093835
243234  151850.0 -2.783175  2.675236 -0.129446 -1.579990  0.163257 -0.491482

              V7        V8        V9  ...       V21       V22       V23  \
278637  0.041019  0.085593 -1.147285  ...  0.305405  0.936258 -0.056952
191786 -0.010259  0.308602 -1.103166  ...  0.293286  0.808270 -0.176873
182166  0.639190 -0.251930  2.129698  ...  0.015857  1.219414 -0.846781
211541  1.907556 -0.426148 -1.404138  ... -0.529620 -1.596487  1.071424
195395 -0.360926  0.137440 -0.650546  ... -0.193657 -0.516363  0.450914
...          ...       ...       ...  ...       ...       ...       ...
273013 -1.395944  2.864199 -0.551824  ...  0.080145 -0.316666  0.071379
186375 -0.719131  0.086375  0.953445  ...  0.022144  0.893774  0.096415
14116  -0.592573  0.200233  2.494517  ... -0.579125 -0.941481  0.038564
84091  -0.355378 -1.021039 -1.010152  ...  1.288496 -0.962541 -0.640448
243234  0.770386 -0.525309  3.019697  ... -0.596512 -0.839976 -0.086257

             V24       V25       V26       V27       V28  Amount  Class
278637  0.219160  0.290690  0.221421 -0.040837 -0.059026   10.59      0
191786  0.025685  0.010856 -0.270198  0.241484  0.126104   15.00      0
182166 -0.798398  0.508283 -0.551486 -0.387281 -0.407557   99.00      0
211541 -0.236820 -0.780467 -1.207436 -0.060758  0.008146  373.90      0
195395  0.543983 -0.503771 -0.408203  0.016273 -0.016981    4.54      0
...          ...       ...       ...       ...       ...     ...    ...
273013 -1.038712  0.208458 -0.035302 -0.986384 -0.613738   21.86      0
186375 -0.948739 -0.018133  0.192306  0.057336 -0.072699   15.17      0
14116  -0.327459  0.461936 -0.528445  0.065178  0.021236   19.89      0
84091  -0.774945 -0.009809 -0.043424 -0.502826 -0.434711   75.08      0
243234 -0.834365  0.170598 -0.378974 -0.618995 -0.374460    3.78      0

[492 rows x 31 columns]
```

In [22]:
```python
#Combine the fraud and the fraud datasets

combine_df = pd.concat([new_legit_df, fraud_df], axis = 0)
```

In [23]:
```python
#view the combined fraud and legit datasets

combine_df
```

Out[23]:

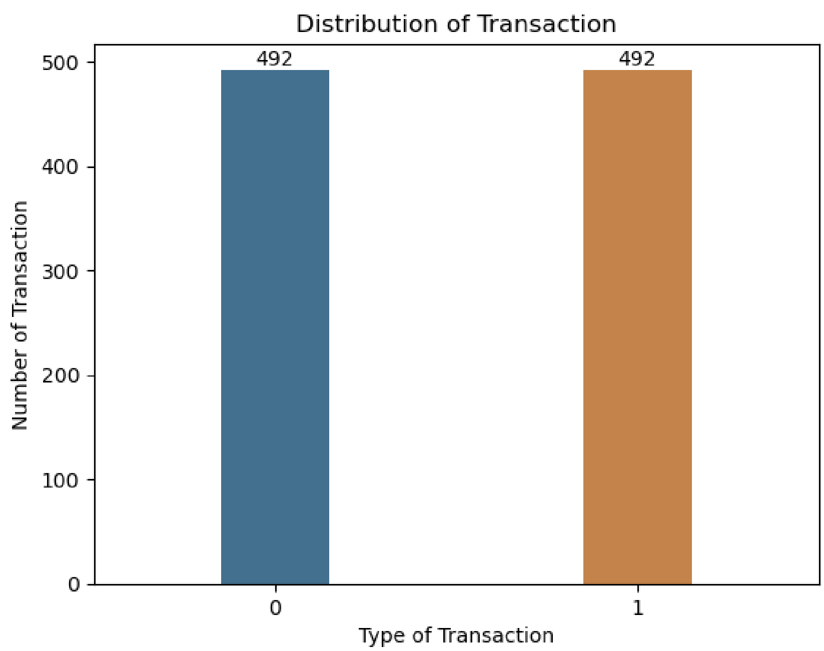
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
278637	168331.0	1.924745	0.476715	-0.972502	3.488065	0.895899	0.768858	0.041019	0.085593	-1.147285	...	0.305405	0.936258	-0.05
191786	129409.0	-0.775259	0.435444	0.797722	-2.198371	-0.492238	-0.918507	-0.010259	0.308602	-1.103166	...	0.293286	0.808270	-0.17
182166	125289.0	-1.169186	-1.106668	-0.302463	-3.042068	2.134270	0.978630	0.639190	-0.251930	2.129698	...	0.015857	1.219414	-0.84
211541	138458.0	-0.636187	-0.617568	0.771143	0.043623	-0.922183	-0.018679	1.907556	-0.426148	-1.404138	...	-0.529620	-1.596487	1.07
195395	131031.0	1.882015	0.456126	0.091758	3.660667	0.211950	0.702324	-0.360926	0.137440	-0.650546	...	-0.193657	-0.516363	0.45
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.63
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.14
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.19
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.45
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.07

984 rows × 31 columns

In [24]:
```
#viewing the new combine data

combine_df["Class"].value_counts()
```

Out[24]:
```
Class
0    492
1    492
Name: count, dtype: int64
```

In [26]:
```
#visualizing the new combine data

ax = sb.countplot(data = combine_df, x = "Class", width = 0.3, saturation = 0.5)
ax.set_title("Distribution of Transaction")
plt.xlabel("Type of Transaction")
plt.ylabel("Number of Transaction")


for i in ax.containers:
    ax.bar_label(i)

plt.show()
```



In [27]:
```
x = combine_df.drop(columns = "Class", axis = 1)
y = combine_df['Class']
```

In [28]:
```
x
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
278637	168331.0	1.924745	0.476715	-0.972502	3.488065	0.895899	0.768858	0.041019	0.085593	-1.147285	...	-0.198856	0.305405	0.93
191786	129409.0	-0.775259	0.435444	0.797722	-2.198371	-0.492238	-0.918507	-0.010259	0.308602	-1.103166	...	0.090186	0.293286	0.80
182166	125289.0	-1.169186	-1.106668	-0.302463	-3.042068	2.134270	0.978630	0.639190	-0.251930	2.129698	...	-0.552342	0.015857	1.21
211541	138458.0	-0.636187	-0.617568	0.771143	0.043623	-0.922183	-0.018679	1.907556	-0.426148	-1.404138	...	0.324785	-0.529620	-1.59
195395	131031.0	1.882015	0.456126	0.091758	3.660667	0.211950	0.702324	-0.360926	0.137440	-0.650546	...	-0.173118	-0.193657	-0.51
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	1.252967	0.778584	-0.31
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.226138	0.370612	0.02
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.247968	0.751826	0.83
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.306271	0.583276	-0.26
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.017652	-0.164350	-0.29

984 rows × 30 columns

In [29]:
```
y
```

Out[29]:
```
278637    0
191786    0
182166    0
211541    0
195395    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

In [30]:
```
# importing the model building libraries

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Splitting and training the datasets

In [31]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 5)
```

In [32]:
```
print(x_train)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
186381  127079.0  1.587861 -0.491081 -2.445199  1.104056  0.893652 -0.360610
146352   87623.0  1.671087 -1.590974 -0.238973 -1.425989 -1.555426 -0.270383
64449    51150.0  0.660132 -0.653132  0.271021  1.132269  0.153691  1.776040
42007    40918.0 -3.140260  3.367342 -2.778931  3.859701 -1.159518 -0.721552
230476  146344.0 -0.099724  2.795414 -6.423856  3.247513 -1.632290 -2.766665
...          ...       ...       ...       ...       ...       ...       ...
163922  116323.0  1.996808  0.202794 -1.777271  1.035389  0.713585 -0.622029
263759  161101.0  1.982028  0.140222 -1.794302  1.073558  0.608813 -0.823077
88897    62341.0 -5.267760  2.506719 -5.290925  4.886134 -3.343188 -1.100085
95795    65470.0 -0.505965  0.922689  1.633677 -0.146791  0.019813 -0.232812
191544  129308.0  0.054682  1.856500 -4.075451  4.100098 -0.800931 -0.292502

              V7        V8        V9  ...       V20       V21       V22  \
186381  0.994574 -0.330142 -0.425863  ...  0.290174  0.285707  0.296347
146352 -1.087410  0.180555  2.550509  ...  0.129945  0.366590  0.897864
64449  -0.261408  0.653031  0.310543  ... -0.026782  0.052919  0.036766
42007  -4.195342 -0.598346 -2.870145  ...  0.077781  2.452339 -0.292963
230476 -2.312223  0.961014 -1.896001  ...  0.340898  0.647714  0.126576
...          ...       ...       ...  ...       ...       ...       ...
163922  0.503880 -0.184656 -0.239743  ... -0.275135  0.089792  0.232353
263759  0.565404 -0.253044 -0.182285  ... -0.269483  0.078842  0.199763
88897  -5.810509  1.726343 -0.749277  ... -0.286043  0.764266  0.473262
95795   0.486511  0.239952 -0.582576  ...  0.025581 -0.143768 -0.378393
191544 -2.317431  1.189747 -0.786238  ...  0.509559  0.618248  0.800932

             V23       V24       V25       V26       V27       V28  Amount
186381 -0.279451  0.207181  0.442862 -0.532037 -0.088852 -0.027822  257.11
146352 -0.009289 -0.306027 -0.416120 -0.194896  0.028648 -0.015419  173.70
64449  -0.036683 -1.049271  0.169593 -0.358261  0.062488  0.024506  169.16
42007  -0.189330 -0.166482  0.038040 -0.015477  0.776691  0.397557    0.76
230476  0.203953  0.008495 -0.174501  0.575295  0.152876 -0.098173   94.82
...          ...       ...       ...       ...       ...       ...     ...
163922  0.099326  0.665664  0.271583 -0.642894 -0.035242 -0.061398   13.90
263759  0.003970 -0.461401  0.292898 -0.586398 -0.034691 -0.069784   30.00
88897   0.548482 -0.156850 -0.710187 -0.366423 -1.486766  0.677664    1.10
95795   0.021609  0.188897 -0.345413  0.072876  0.275264  0.107922    0.89
191544  0.130016  0.288946 -0.366658  0.030307  0.431182  0.110698   80.90

[738 rows x 30 columns]
```

In [33]:
```
print(x_train.shape, x_test.shape)
```

```
(738, 30) (246, 30)
```

In [34]:
```
model = LogisticRegression()
```

In [35]:
```
model.fit(x_train, y_train)
```

```
C:\Users\DajahV01\AppData\Local\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarni
ng: 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 (https://scikit-learn.org/stable/modules/preprocessi
ng.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/
modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[35]:
```
▾ LogisticRegression
LogisticRegression()
```

Model evaluation

In [36]:
```
#Testing the accuracy of the training data

x_train_predict = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_predict, y_train)
print(f"The model's training data accuracy is: {round(training_data_accuracy * 100, 2)}%")
```

```
The model's training data accuracy is: 93.5%
```

In [37]:
```
#Testing the accuracy of the testing data

x_test_predict = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_predict, y_test)
print(f"The model's testing data accuracy is: {round(test_data_accuracy * 100, 2)}%")
```

```
The model's testing data accuracy is: 95.12%
```

In [38]:
```
from sklearn.metrics import accuracy_score,recall_score,precision_score,classification_report, ConfusionMatrixDispla
ax = ConfusionMatrixDisplay.from_predictions(y_test, x_test_predict)

plt.show()
```