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

In [2]:
```
df = pd.read_csv('creditcard.csv')
```

In [3]:
```
df
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 31 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [5]:
```
#display all columns
pd.options.display.max_columns = None
```

In [6]:
```
df
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 31 columns

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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 [8]:
```
df = df.drop(['Time'],axis=1)
```

In [9]:
```
df
```

Out[9]:

	V1	V2	V3	V4	V5	V6	V7	V8
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533
...	...	...	...	...	...	...	...	..
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650

284807 rows × 30 columns

In [10]:
```
df.shape
```

Out[10]:
```
(284807, 30)
```

In [11]:
```
df.duplicated().any()
```

Out[11]:
```
True
```

In [12]:
```
df = df.drop_duplicates()
```

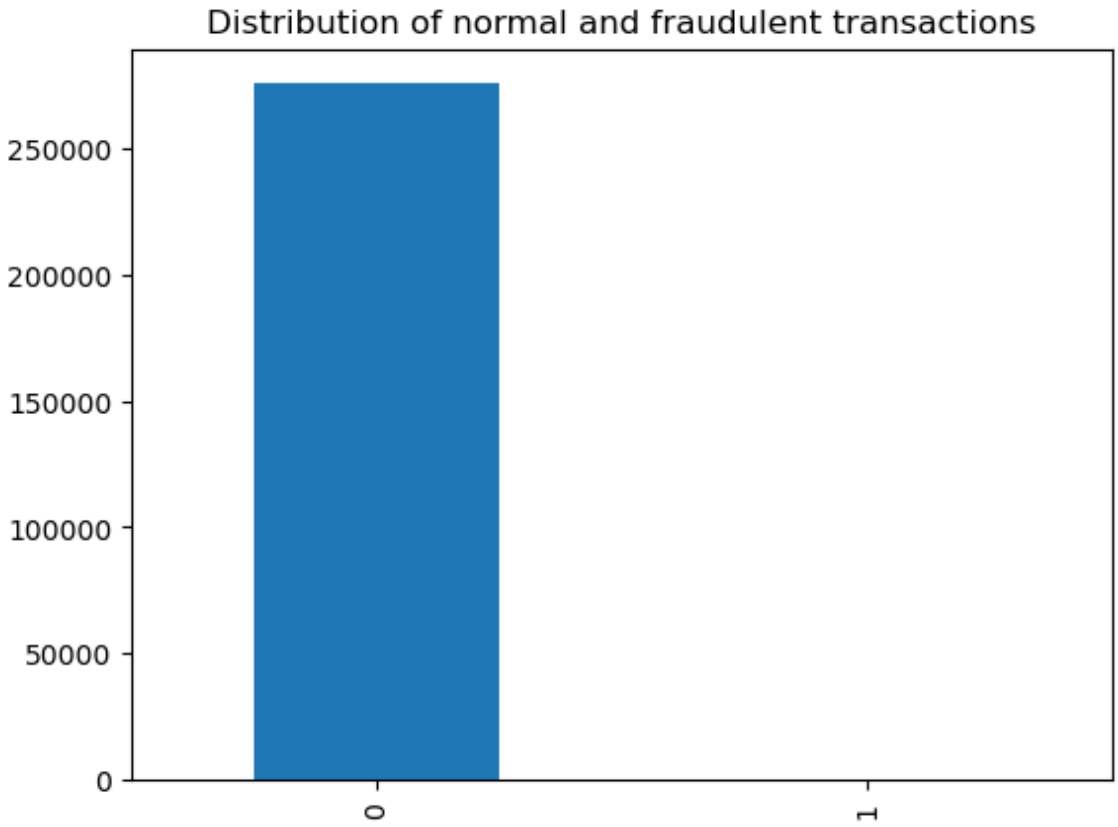
In [13]:
```
df.shape
```

Out[13]:
```
(275663, 30)
```

In [14]:
```
#distribution of legit transaction and fraudulent transactions
df['Class'].value_counts()
```

Out[14]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [15]:
```
df['Class'].value_counts().plot(kind='bar')
plt.title('Distribution of normal and fraudulent transactions')
plt.show()
```



In [16]:
```
#define features and target
x = df.drop('Class',axis=1)
y = df['Class']
```

In [17]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20,random_
```

In [18]:
```
#Dataset is highly imbalanced
#For handling it we will use undersampling (undersampling means randomly de
normal = df[df.Class==0]
fraud = df[df.Class==1]
```

In [19]:
```
print(normal.shape)
print(fraud.shape)
```

```
(275190, 30)
(473, 30)
```

In [20]:
```
normal_sample = normal.sample(n=473)
```

In [21]:
```
normal_sample.shape
```

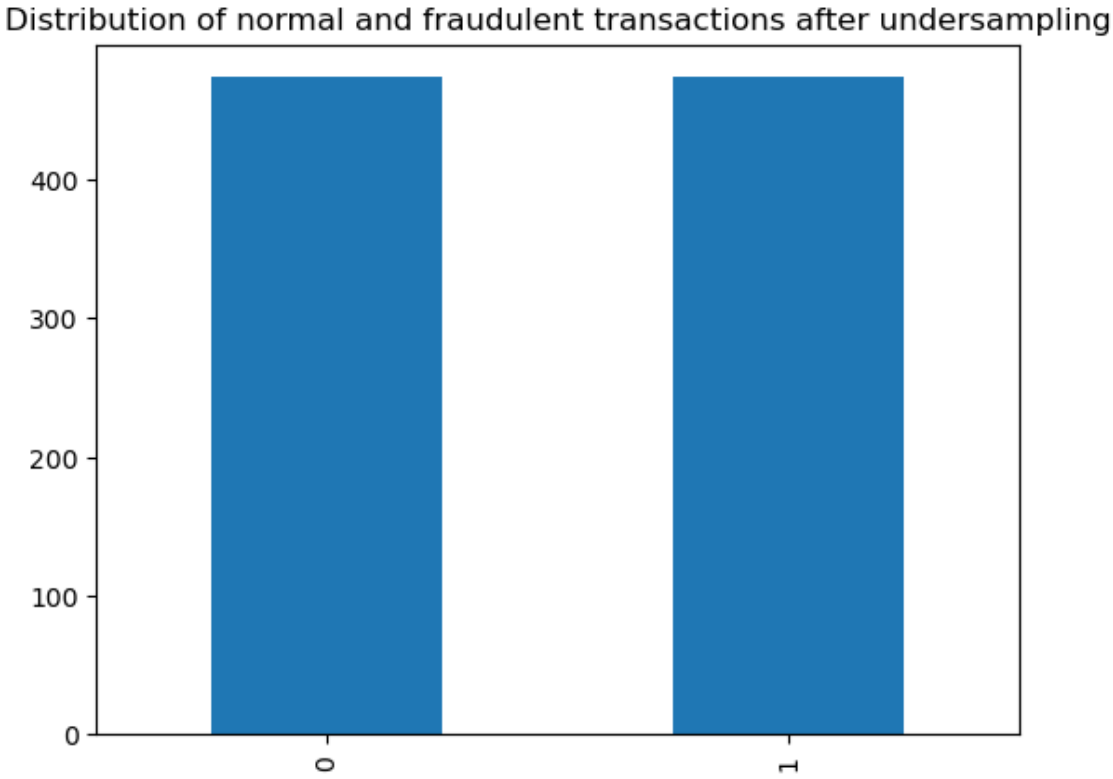
Out[21]: (473, 30)

In [22]:
```
new_df = pd.concat([normal_sample,fraud],ignore_index=True)
```

In [23]:
```
new_df['Class'].value_counts()
```

Out[23]:
```
0    473
1    473
Name: Class, dtype: int64
```

In [24]:
```
new_df['Class'].value_counts().plot(kind='bar')
plt.title('Distribution of normal and fraudulent transactions after undersa
plt.show()
```



In [25]:
```
new_df
```

Out[25]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-0.903065	0.276580	2.403558	-1.891507	-0.167944	-0.381166	0.139354	0.166015	1
1	1.965485	-0.423590	-0.643265	0.124170	-0.184927	0.291233	-0.627012	0.201283	1
2	1.215176	0.337917	0.287224	0.638964	-0.148269	-0.593767	0.034927	-0.087724	-0
3	1.094699	-0.024164	1.409170	1.270234	-0.962412	0.017873	-0.606755	0.181248	0
4	1.840058	-0.781385	-0.817864	0.015217	-0.300163	0.347595	-0.642807	0.160695	0
...	...	...	...	...	...	...	...	...	
941	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2
942	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1
943	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0
944	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1
945	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0

946 rows × 30 columns

In [26]:
```
#define features and target
x = new_df.drop('Class',axis=1)
y = new_df['Class']
```

In [27]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20,random_
```

In [28]:
```
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression()
model_lr.fit(x_train,y_train)
```

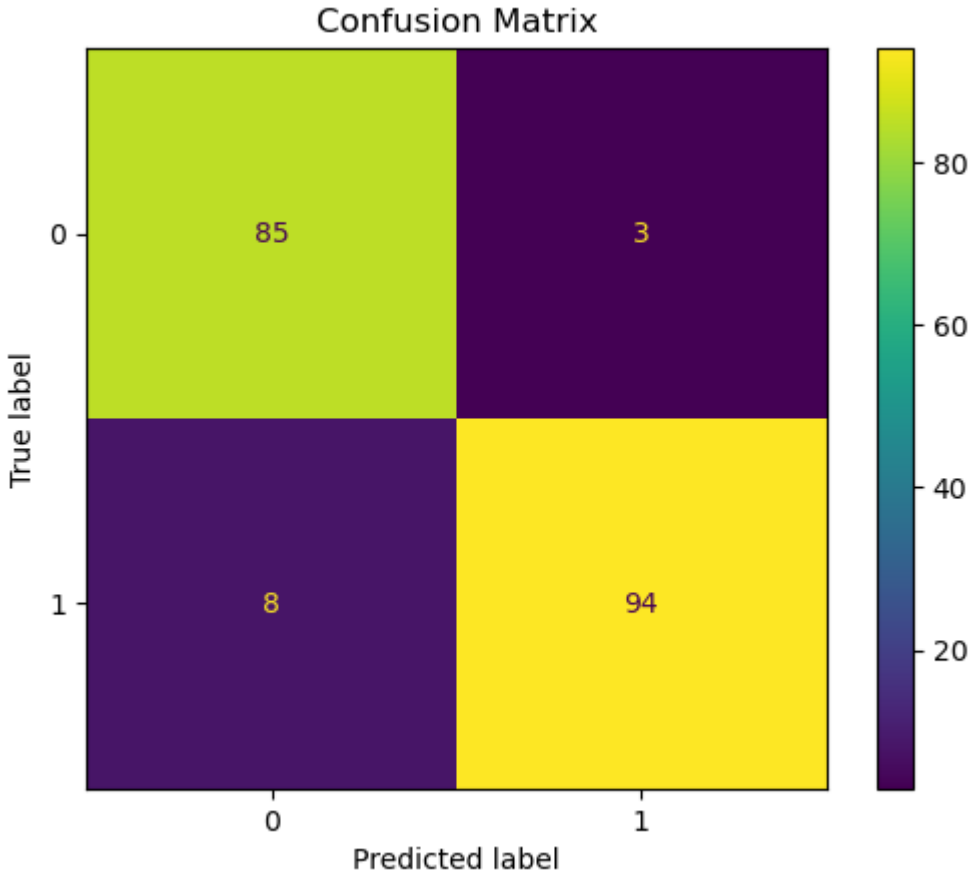
```
C:\Users\kalya\anaconda3\envs\Lib\site-packages\sklearn\linear_model\_logi
stic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-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-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[28]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]:
```
y_pred = model_lr.predict(x_test)
```

In [30]:
```
from sklearn.metrics import ConfusionMatrixDisplay,accuracy_score, classifi
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()
print(f'Accuracy: {accuracy_score(y_test,y_pred)}')
print(classification_report(y_test,y_pred))
```




```
Accuracy: 0.9421052631578948
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        88
           1       0.97      0.92      0.94       102

    accuracy                           0.94       190
   macro avg       0.94      0.94      0.94       190
weighted avg       0.94      0.94      0.94       190
```

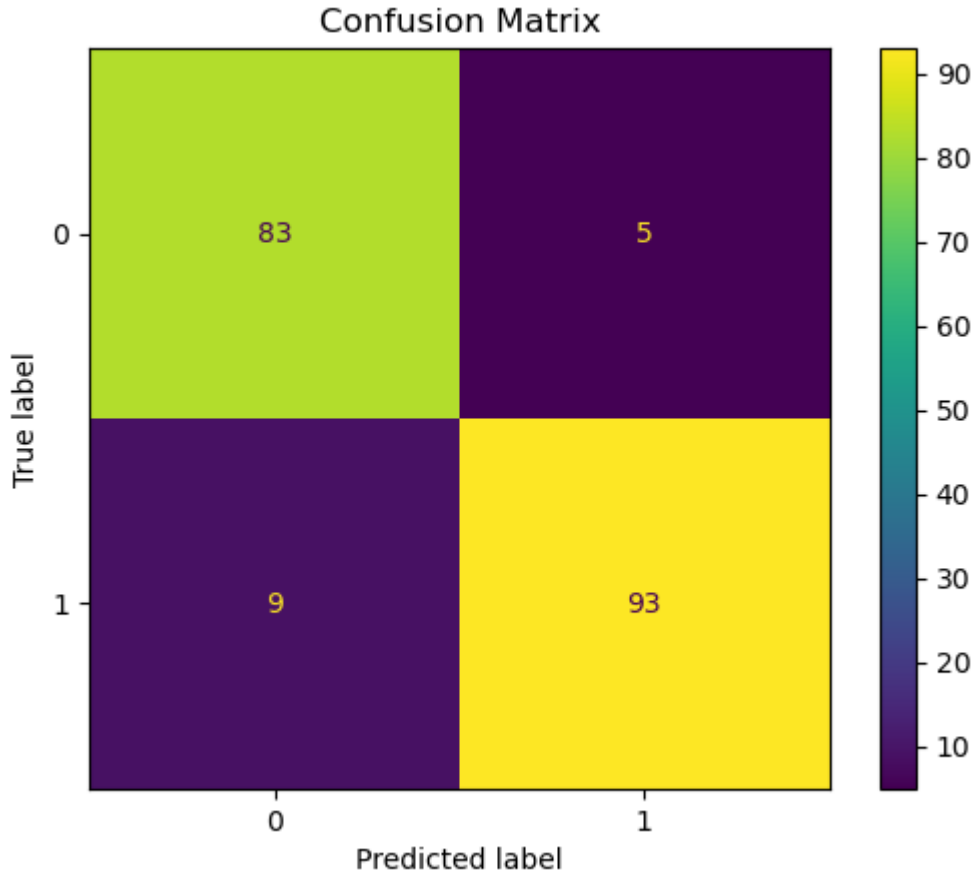
In [31]:
```
#Using Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier()
model_rf.fit(x_train,y_train)
```

Out[31]:
```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
y_pred1 = model_rf.predict(x_test)
```

In [33]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred1)
plt.title('Confusion Matrix')
plt.show()
print(f'Accuracy: {accuracy_score(y_test,y_pred1)}')
print(classification_report(y_test,y_pred1))
```




```
Accuracy: 0.9263157894736842
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        88
           1       0.95      0.91      0.93       102

    accuracy                           0.93       190
   macro avg       0.93      0.93      0.93       190
weighted avg       0.93      0.93      0.93       190
```

In [ ]:
```

```