In [1]:
```
# imprting the dataset
import pandas as pd
data = pd.read_csv('creditcard.csv')
#displays the first 6 rows
data.head()
```

Out[1]:

	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
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
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
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
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

5 rows × 31 columns

In [2]:
```
#shape of the dataset
data.shape
```

Out[2]:
```
(284807, 31)
```

In [4]:
```
#Rows and COlumns
print("Number of Rows:", data.shape[0]) #for rows
print('Number of Columns:',data.shape[1])# for columns
```

```
Number of Rows: 284807
Number of Columns: 31
```

In [5]:
```
#informations about the dataset
data.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 [6]:
```
#checking for null values
data.isnull().sum()
```

Out[6]:
```
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 [7]:
```
#displaying the firdt 6 rows for analysing the 'Amoount ' column
data.head()
```

Out[7]:

	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
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
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
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
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

5 rows × 31 columns

In [8]:
```
# standardize for Amount column
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [9]:
```
#this again displays the accurate entires (afer standardizing) in the "Amount" column
data.head()
```

Out[9]:

	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
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
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
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
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

5 rows × 31 columns

In [10]:
```
# droping time column
data = data.drop(['Time'],axis=1)
data.shape
```

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

In [11]:
```
#checling djuuplicated values
data.duplicated().any()
```

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

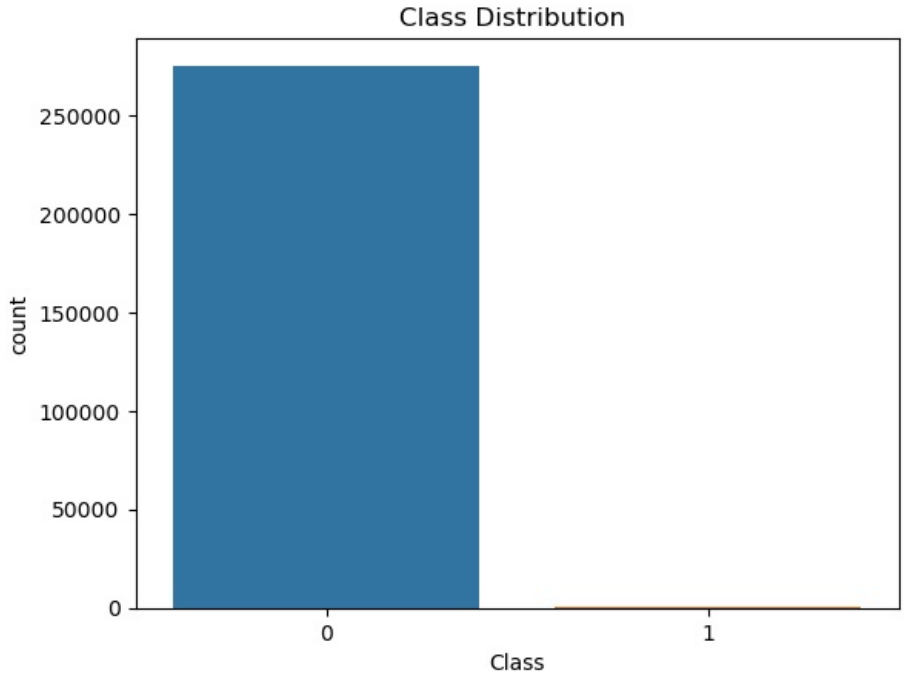
In [12]:
```
#drop duplicate values
data = data.drop_duplicates()
data.shape
```

Out[12]:
```
(275663, 30)
```

In [13]:
```
#checling Classes
data['Class'].value_counts()
```

Out[13]:
```
Class
0    275190
1       473
Name: count, dtype: int64
```

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualize the class distribution
sns.countplot(x='Class', data=data)
plt.title('Class Distribution ')
plt.show()
```



In [15]:
```
# assignig class 0 as 'normal' and class 1 as 'fraud'
normal = data[data['Class']==0]
fraud = data[data['Class']==1]
print("Number of Normal Transactionns:",len(normal))
print("Number of Fraud Transactionns:",len(fraud))
```

```
Number of Normal Transactionns: 275190
Number of Fraud Transactionns: 473
```

In [16]:
```
#taking 473 samples from normal claass to match fraud class
normal_sample= normal.sample(n=473)
normal_sample.shape
```

Out[16]:
```
(473, 30)
```

In [33]:
```
fraud.shape
```

Out[33]:
```
(473, 30)
```

In [17]:
```
# assigns the sample data to new_data
new_data = pd.concat([normal_sample,fraud])
```

In [18]:
```
# counting the class values to check for even distribution
new_data['Class'].value_counts()
```

Out[18]:
```
Class
0    473
1    473
Name: count, dtype: int64
```

In [23]:
```
# columns and targets
X = new_data.drop('Class',axis = 1)
y = new_data['Class']
```

In [24]:
```
# training and splitting the dataset
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_state = 42)
```

In [25]:
```
# applying logistic regression
```

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train,y_train)
```

Out[25]:

```
LogisticRegression()
```

In [26]:

```
y_pred1 =log.predict(X_test)
```

In [27]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print("LOGISTIC REGRESSION")
print('\nAccuracy:' , accuracy_score(y_test,y_pred1))
print('\nPrecision:',precision_score(y_test,y_pred1))
print('\nF1 Score:' , f1_score(y_test,y_pred1))
print('\nRecall:' , recall_score(y_test,y_pred1))
```

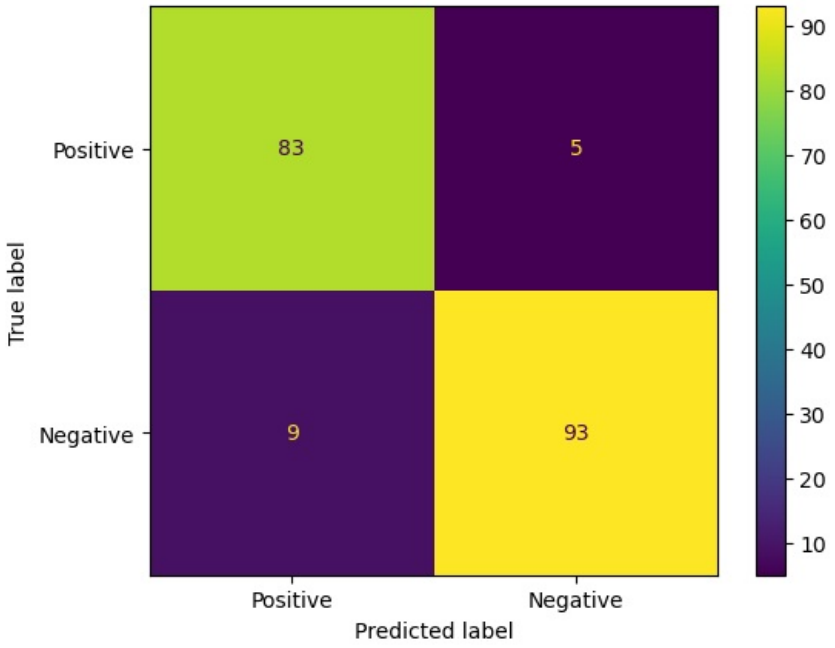
```
LOGISTIC REGRESSION

Accuracy: 0.9263157894736842

Precision: 0.9489795918367347

F1 Score: 0.93

Recall: 0.9117647058823529
```

In [28]:

```
#displays the confusion matrix for Linear Regression
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test,y_pred1)
cm
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,display_labels= ['Positive','Negative'])
display_cm.plot()
```

Out[28]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22532bcb1d0>
```



In [29]:

```
# applying random forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

Out[29]:

```
RandomForestClassifier()
```

In [30]:

```
y_predrf = rf.predict(X_test)
```

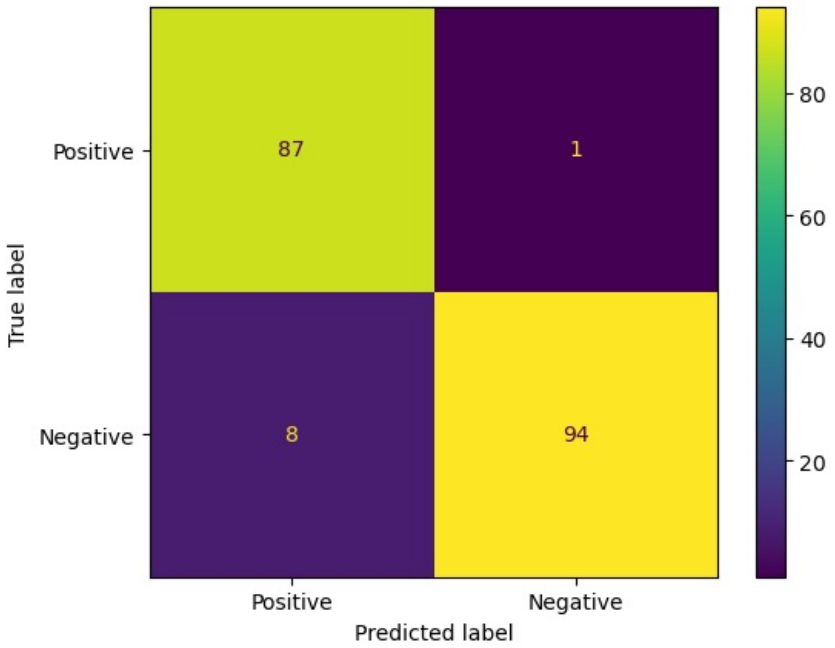
In [31]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('RANDOMFOREST CLASSIFIER:')
print('\nAccuracy:' , accuracy_score(y_test,y_predrf))
print('\nPrecision:',precision_score(y_test,y_predrf))
print('\nF1 Score:' , f1_score(y_test,y_predrf))
print('\nRecall:' , recall_score(y_test,y_predrf))
```

```
RANDOMFOREST CLASSIFIER:

Accuracy: 0.9526315789473684

Precision: 0.9894736842105263

F1 Score: 0.9543147208121827

Recall: 0.9215686274509803
```

In [32]:
```
# displays the confusion matrix Random Forest Classifier
cm = confusion_matrix(y_test,y_predrf)
cm
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,display_labels= ['Positive','Negative'])
display_cm.plot()
```

Out[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22532c6bc90>
```



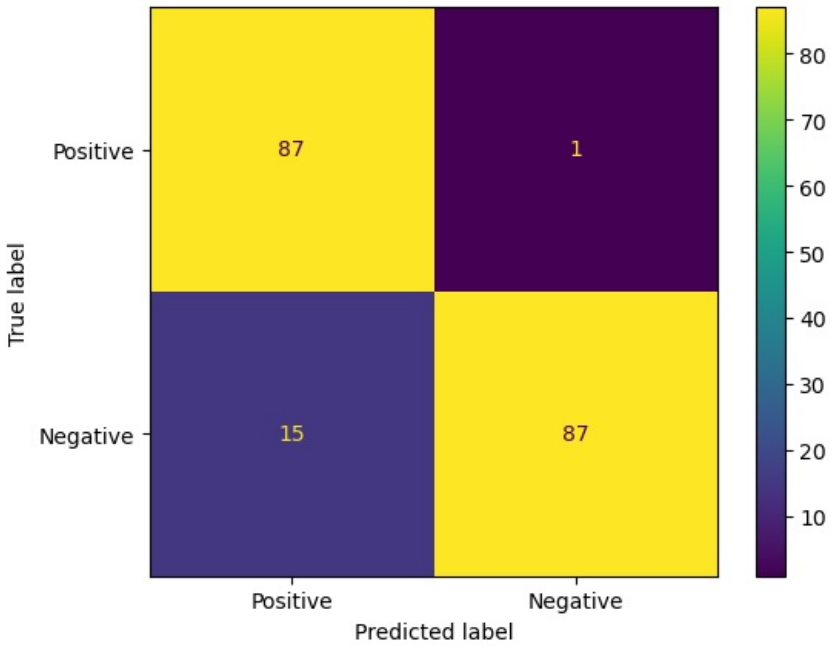
In [33]:
```
#Applying Naive bayes
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
y_pred_nv= naive_bayes.predict(X_test)
```

In [34]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
print('NAIVE BAYES:')
print('\nAccuracy:' , accuracy_score(y_test,y_pred_nv))
print('\nPrecision:',precision_score(y_test,y_pred_nv))
print('\nF1 Score:' , f1_score(y_test,y_pred_nv))
print('\nRecall:' , recall_score(y_test,y_pred_nv))
```

```
NAIVE BAYES:

Accuracy: 0.9210526315789473

Precision: 0.9887640449438202

F1 Score: 0.9214659685863874

Recall: 0.8627450980392157
```

In [70]:
```
#displays the confusion matrix for Naive Bayes
cm = confusion_matrix(y_test,y_pred_nv)
cm
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,display_labels= ['Positive','Negative'])
display_cm.plot()
```

Out[70]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1dc2c388850>
```



In [61]:
```
result = pd.DataFrame({'MODELS':['LOGISTIC REGRESSION','RANDOM FOREST','NAIVE BAYES'],
                       'ACCURACY':[accuracy_score(y_test,y_pred1),
                                   accuracy_score(y_test,y_pred2),
                                   accuracy_score(y_test,y_pred_nv)]})
```

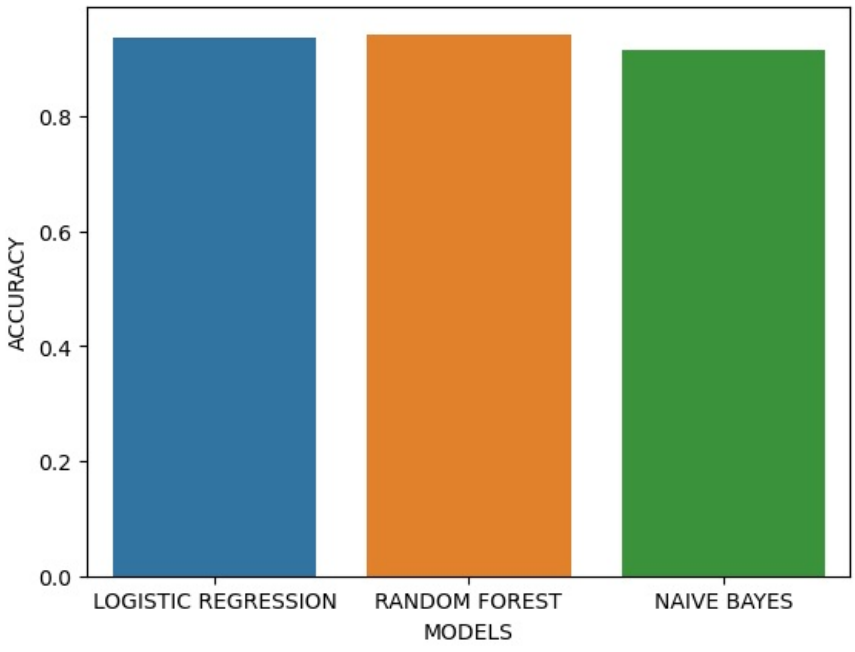
In [62]:
```
# displays the results
result
```

Out[62]:

	MODELS	ACCURACY
0	LOGISTIC REGRESSION	0.936842
1	RANDOM FOREST	0.942105
2	NAIVE BAYES	0.915789

In [63]:
```
# displays the resultant accuracy score in a bar graph
sns.barplot(x = 'MODELS',y = 'ACCURACY', data = result)
```

Out[63]:
```
<Axes: xlabel='MODELS', ylabel='ACCURACY'>
```



In [ ]: