```
Credit Card Fraud Detection
```

Step 1. Perform Exploratory Data Analysis (EDA)

Importing dataset

```
import pandas as pd
from collections import Counter
import itertools

# Load the csv file

dataframe = pd.read_csv("creditcard.csv")
dataframe.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

Check for null values in the credit card dataset.

```
dataframe.isnull().values.any()

dataframe.dropna(inplace=True)
```

```
dataframe.isnull().sum()
```

```
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
```

```
dataframe["Amount"].describe()
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
```

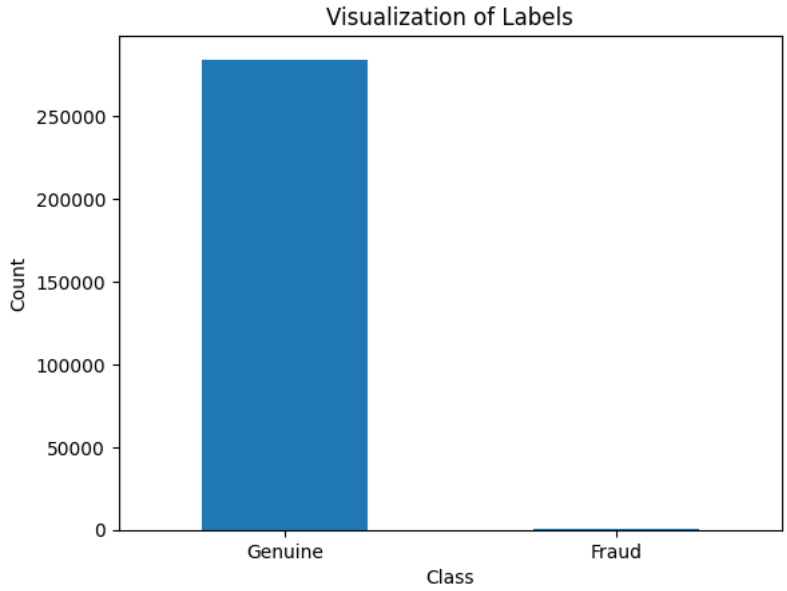
```
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

Now, let's check the number of occurrences of each class label and plot the information using matplotlib.

```
non_fraud = len(dataframe[dataframe.Class == 0])
fraud = len(dataframe[dataframe.Class == 1])
fraud_percent = (fraud / (fraud + non_fraud)) * 100

print("Number of Genuine transactions: ", non_fraud)
print("Number of Fraud transactions: ", fraud)
print("Percentage of Fraud transactions: {:.4f}".format(fraud_percent))
```

```
Number of Genuine transactions:  284315
Number of Fraud transactions:  492
Percentage of Fraud transactions: 0.1727
```

```
import matplotlib.pyplot as plt

labels = ["Genuine", "Fraud"]
count_classes = dataframe.value_counts(dataframe['Class'], sort= True)
count_classes.plot(kind = "bar", rot = 0)
plt.title("Visualization of Labels")
plt.ylabel("Count")
plt.xticks(range(2), labels)
plt.show()
```




```
import numpy as np
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
dataframe["NormalizedAmount"] = scaler.fit_transform(dataframe["Amount"].values.reshape(-1, 1))
dataframe.drop(["Amount", "Time"], inplace= True, axis= 1)

Y = dataframe["Class"]
X = dataframe.drop(["Class"], axis= 1)
```

```
from sklearn.model_selection import train_test_split

(train_X, test_X, train_Y, test_Y) = train_test_split(X, Y, test_size= 0.3, random_state= 42)

print("Shape of train_X: ", train_X.shape)
print("Shape of test_X: ", test_X.shape)
```

```
Shape of train_X:  (199364, 29)
Shape of test_X:  (85443, 29)
```

Step-2 :Step 2: Apply Machine Learning Algorithms to Credit Card Dataset

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
#Decision Tree
decision_tree = DecisionTreeClassifier()

# Random Forest
random_forest = RandomForestClassifier(n_estimators= 100)
```

Step-3 : Train and Evaluate our Models on the Dataset

```
decision_tree=DecisionTreeClassifier()
decision_tree.fit(train_X, train_Y)
predictions_dt = decision_tree.predict(test_X)
decision_tree_score = decision_tree.score(test_X, test_Y) * 100
random_forest.fit(train_X, train_Y)
predictions_rf = random_forest.predict(test_X)
random_forest_score = random_forest.score(test_X, test_Y) * 10
print("Random Forest Score: ", random_forest_score)
print("Decision Tree Score: ", decision_tree_score)
```

```
Random Forest Score:  99.95903701883127
Decision Tree Score:  99.92392589211521
```

The Random Forest classifier has slightly an edge over the Decision Tree classifier.

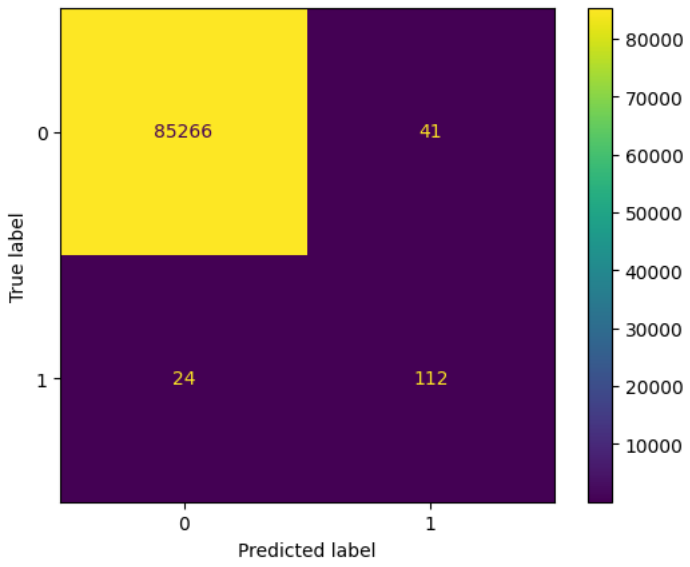
```
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, recall_score, f1_score


def metrics(actuals, predictions):
    print("Accuracy: {:.5f}".format(accuracy_score(actuals, predictions)))
    print("Precision: {:.5f}".format(precision_score(actuals, predictions)))
    print("Recall: {:.5f}".format(recall_score(actuals, predictions)))
    print("F1-score: {:.5f}".format(f1_score(actuals, predictions)))
```

Let's visualize the confusion matrix and the evaluation metrics of our Decision Tree model.

```
confusion_matrix_dt = confusion_matrix(test_Y, predictions_dt.round())
print("Confusion Matrix - Decision Tree")
print(confusion_matrix_dt)
```

```
Confusion Matrix - Decision Tree
[[85266    41]
 [   24   112]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_dt)
disp = disp.plot()
plt.show()
```



```
print("Evaluation of Decision Tree Model")
print()
metrics(test_Y, predictions_dt.round())
```

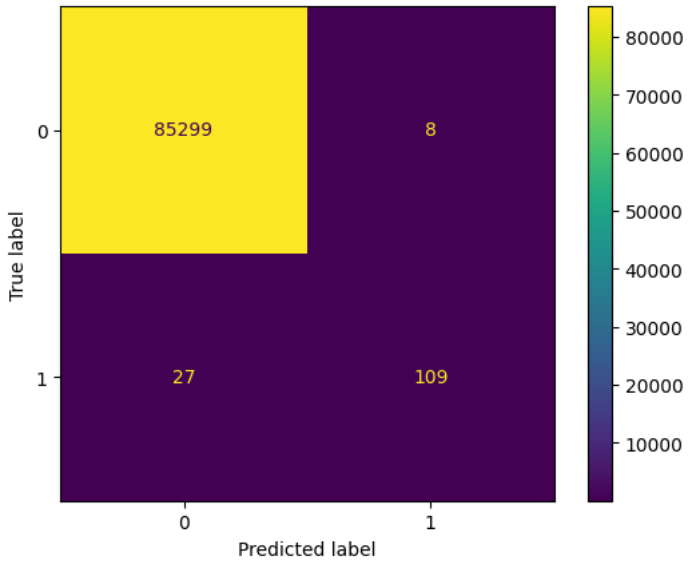
```
Evaluation of Decision Tree Model

Accuracy: 0.99924
Precision: 0.73203
Recall: 0.82353
F1-score: 0.77509
```

Let's visualize the confusion matrix and the evaluation metrics of our Random Forest model.

```
confusion_matrix_rf = confusion_matrix(test_Y, predictions_rf.round())
print("Confusion Matrix - Random Forest")
print(confusion_matrix_rf)
```

```
Confusion Matrix - Random Forest
[[85299     8]
 [   27   109]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_rf)

disp = disp.plot()

plt.show()
```




```
print("Evaluation of Random Forest Model")
print()
metrics(test_Y, predictions_rf.round())
```

```
Evaluation of Random Forest Model

Accuracy: 0.99959
Precision: 0.93162
Recall: 0.80147
F1-score: 0.86166
```

```
from imblearn.over_sampling import SMOTE

X_resampled, Y_resampled = SMOTE().fit_resample(X, Y)
print("Resampled shape of X: ", X_resampled.shape)
print("Resampled shape of Y: ", Y_resampled.shape)

value_counts = Counter(Y_resampled)
print(value_counts)

(train_X, test_X, train_Y, test_Y) = train_test_split(X_resampled, Y_resampled, test_size= 0.3, random_state= 42)
```

```
Resampled shape of X:  (568630, 29)
Resampled shape of Y:  (568630,)
```

```
Counter({0: 284315, 1: 284315})
```

As the Random Forest algorithm performed better than the Decision Tree algorithm, we will apply the Random Forest algorithm to our resampled data.

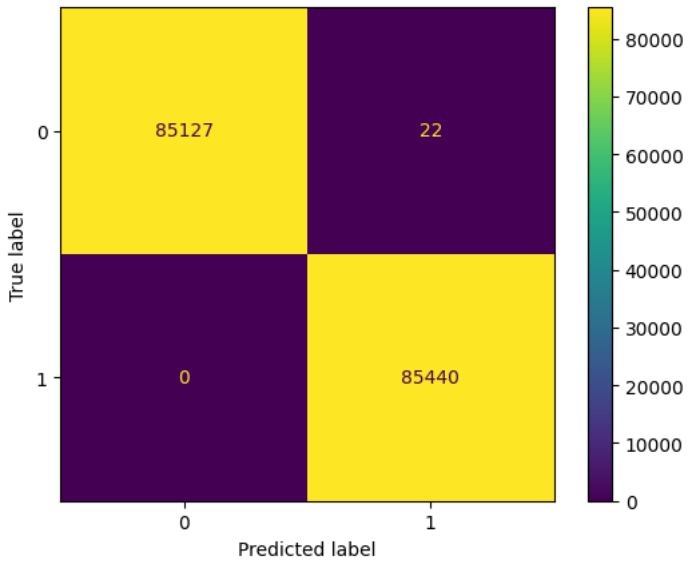
```
rf_resampled = RandomForestClassifier(n_estimators = 100)
rf_resampled.fit(train_X, train_Y)

predictions_resampled = rf_resampled.predict(test_X)
random_forest_score_resampled = rf_resampled.score(test_X, test_Y) * 100
```

visualizing the predictions of our model and plot the confusion matrix.

```
cm_resampled = confusion_matrix(test_Y, predictions_resampled .round())
print("Confusion Matrix - Random Forest")
print(cm_resampled)
```

```
Confusion Matrix - Random Forest
[[85127    22]
 [    0 85440]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

disp = ConfusionMatrixDisplay(confusion_matrix=cm_resampled)

disp = disp.plot()

plt.show()
```




```
print("Evaluation of Random Forest Model")
print()

metrics(test_Y, predictions_resampled.round())
```

```
Evaluation of Random Forest Model

Accuracy: 0.99989
Precision: 0.99977
Recall: 1.00000
F1-score: 0.99989
```

Now, it is clearly evident that our model performed much better than our previous Random Forest classifier without oversampling.

conclusion : In this python machine learning project, I built a binary classifier using the Random Forest algorithm to detect credit card fraud transactions. Through this project, I understood and applied techniques to address the class imbalance issues and achieved an accuracy of more than 99%.