# Credit Card Fraud Detection - Detailed Code Explanation

## Import Libraries

```python  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV  
from sklearn.preprocessing import StandardScaler  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix  
from imblearn.over\_sampling import SMOTE  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
```  
- \*\*pandas\*\*: Library for data manipulation and analysis.  
- \*\*scikit-learn\*\*: Used for machine learning operations like splitting data, scaling features, creating models, tuning hyperparameters, and evaluating model performance.  
- \*\*imblearn\*\*: Specifically, \*\*SMOTE\*\* (Synthetic Minority Over-sampling Technique) to address class imbalance in the dataset.  
- \*\*matplotlib.pyplot\*\* and \*\*seaborn\*\*: For creating visualizations, such as feature importance plots.  
- \*\*numpy\*\*: Provides efficient numerical operations.

## Step 1: Load the Dataset

```python  
data = pd.read\_csv('creditcard.csv')  
```  
Loads the dataset from a CSV file into a pandas DataFrame. The dataset contains information about credit card transactions with a target column `Class` where `0` indicates a legitimate transaction and `1` indicates fraud.

## Step 2: Check for Missing Values and Data Types

```python  
print("Checking for missing values and data types in the dataset:")  
print(data.isnull().sum())  
print(data.dtypes)  
```  
Checks if there are missing values and displays the data types of each column to ensure data cleanliness. Missing or incorrect data types can lead to issues during model training.

## Step 3: Ensure Columns Are Numeric and Drop NaNs

```python  
data = data.apply(pd.to\_numeric, errors='coerce')  
data = data.dropna()  
```  
- \*\*`pd.to\_numeric`\*\*: Converts columns to numeric data types; if there are non-numeric values, it assigns `NaN`.  
- \*\*`dropna`\*\*: Removes rows with `NaN` values. This ensures the dataset only contains valid numeric data for model training.

## Step 4: Separate Features and Labels

```python  
X = data.drop('Class', axis=1)  
y = data['Class']  
```  
- \*\*`X`\*\*: Contains all features, excluding the target column `Class`.  
- \*\*`y`\*\*: Contains only the target column `Class`, which will be used for supervised learning.

## Step 5: Address Class Imbalance Using SMOTE

```python  
smote = SMOTE(random\_state=42)  
X\_resampled, y\_resampled = smote.fit\_resample(X, y)  
```  
- \*\*SMOTE (Synthetic Minority Over-sampling Technique)\*\*: Generates synthetic samples for the minority class (fraudulent transactions) to balance the dataset. Class imbalance can negatively affect the model, making it biased towards the majority class.

## Step 6: Split the Data

```python  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.3, random\_state=42)  
```  
- \*\*train\_test\_split\*\*: Splits the data into training and testing sets. 30% of the data is set aside for testing, while the remaining 70% is used for training.

## Step 7: Feature Scaling

```python  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
```  
- \*\*StandardScaler\*\*: Standardizes the feature values by scaling them to have a mean of 0 and standard deviation of 1. This ensures consistent feature ranges, which helps improve the model's performance.

## Step 8: Initialize Random Forest Model

```python  
rf = RandomForestClassifier(random\_state=42)  
```  
- Initializes a Random Forest Classifier, which is an ensemble learning method that combines multiple decision trees to achieve better accuracy and robustness.

## Step 9: Hyperparameter Tuning Using RandomizedSearchCV

```python  
param\_dist = {  
 'n\_estimators': [50, 100],   
 'max\_depth': [10, 20, None],   
 'min\_samples\_split': [2, 5],  
 'min\_samples\_leaf': [1, 2]  
}  
random\_search = RandomizedSearchCV(  
 estimator=rf,   
 param\_distributions=param\_dist,  
 n\_iter=10,   
 cv=2,   
 scoring='f1',  
 n\_jobs=-1,   
 random\_state=42  
)  
random\_search.fit(X\_train, y\_train)  
```

## Retrieve the Best Model and Evaluate

```python  
best\_rf = random\_search.best\_estimator\_  
y\_pred\_best\_rf = best\_rf.predict(X\_test)  
```

## Final Model Evaluation

```python  
print("Optimized Random Forest Performance:")  
print(classification\_report(y\_test, y\_pred\_best\_rf))  
print("Accuracy:", accuracy\_score(y\_test, y\_pred\_best\_rf))  
print("Confusion Matrix:  
", confusion\_matrix(y\_test, y\_pred\_best\_rf))  
```

## Feature Importance Plot

```python  
feature\_importances = best\_rf.feature\_importances\_  
features = X.columns  
importance\_df = pd.DataFrame({'feature': features, 'importance': feature\_importances})  
importance\_df = importance\_df.sort\_values(by='importance', ascending=False)  
  
plt.figure(figsize=(10, 8))  
sns.barplot(x='importance', y='feature', data=importance\_df)  
plt.title("Feature Importance in the Optimized Random Forest Model")  
plt.xlabel("Importance")  
plt.ylabel("Feature")  
plt.show()  
```