Importing required libraries

In Gradient Boosting algorithm the next tree is grown keeping in mind the errors in the previous tree.

In [4]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score, confusion_matrix, accuracy_score

import matplotlib.pyplot as plt

from pylab import rcParams
rcParams['figure.figsize'] = 8, 8
```

In [5]:
```
# Reading in the dataset
data=pd.read_csv("creditcard.csv")
data.head()
```

Out[5]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.114697	0.796303	-0.149553	-0.823011	0.878763	-0.553152	0.939259	-0.108502	0.111137
1	-0.039318	0.495784	-0.810884	0.546693	1.986257	4.386342	-1.344891	-1.743736	-0.563103
2	2.275706	-1.531508	-1.021969	-1.602152	-1.220329	-0.462376	-1.196485	-0.147058	-0.950224
3	1.940137	-0.357671	-1.210551	0.382523	0.050823	-0.171322	-0.109124	-0.002115	0.869258
4	1.081395	-0.502615	1.075887	-0.543359	-1.472946	-1.065484	-0.443231	-0.143374	1.659826

5 rows × 30 columns

In [6]:
```
# Checking the shape of our data
data.shape
```

Out[6]:
```
(56962, 30)
```

In [7]:
```
# Checking the distribution of two classes in the target variable
data.Target.value_counts()
```

Out[7]:
```
0    56864
1       98
Name: Target, dtype: int64
```

In [8]:
```
# Creating the dataset with all independent variables
X = data.iloc[:,:-1]

# Creating the dataset with the dependent variable
Y = data.iloc[:,-1]
```

Building the Model

In [9]:
```
# Splitting the dataset into the Training set and Test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, rando
```

In [10]:
```
print("The shape of train dataset :")
print(X_train.shape)

print("\n The shape of test dataset :")
print(X_test.shape)
```

```
The shape of train dataset :
(45569, 29)

 The shape of test dataset :
(11393, 29)
```

In [11]:
```
print("Distribution of classes of dependent variable in train :")
print(Y_train.value_counts())

print("\n Distribution of classes of dependent variable in train :")
print(Y_test.value_counts())
```

```
Distribution of classes of dependent variable in train :
0    45491
1       78
Name: Target, dtype: int64

 Distribution of classes of dependent variable in train :
0    11373
1       20
Name: Target, dtype: int64
```

In [12]:
```
# Create a rf_classifier object with number of trees set to 50
gbdt_classifier = GradientBoostingClassifier(n_estimators=50,random_state=0)

# Fit the object to train dataset
gbdt_classifier.fit(X_train, Y_train)
```

Out[12]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=50,
              n_iter_no_change=None, presort='auto', random_state=0,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

In [13]:
```
train_preds = gbdt_classifier.predict(X_train)
test_preds = gbdt_classifier.predict(X_test)
```

In [14]:
```
roc_auc_score(Y_train,train_preds)
```

Out[14]:
```
0.8973809415105495
```

In [15]:
```
# Calculate roc_auc score
roc_auc_score(Y_test,test_preds)
```

Out[15]:
```
0.7996922535830476
```

Variable Importance Ranking

In [16]:
```
features = X_train.columns
importances = gbdt_classifier.feature_importances_
indices = np.argsort(importances)
```

In [17]:
```
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='black', align='center
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')

plt.show()
```