```
import pandas as pd
import zipfile
with zipfile.ZipFile("/content/creditcard.csv.zip", 'r') as zip_ref:
    zip_ref.extractall("/content")
df = pd.read_csv("/content/creditcard.csv.zip")
print(df.shape)
df.head()
```

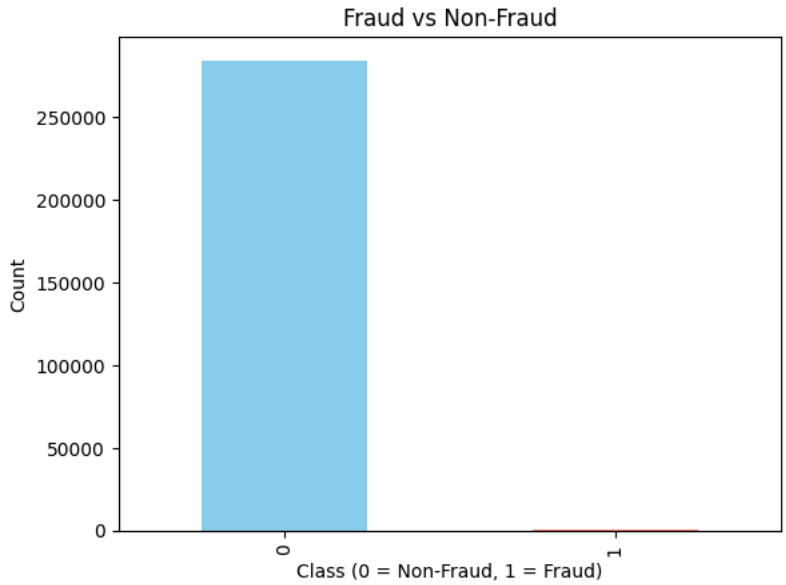
```
(284807, 31)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
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.
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.
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.
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.
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.

5 rows × 31 columns

```
df.info()
df.describe()
print(df['Class'].value_counts())
print("Fraud ratio:", df['Class'].value_counts(normalize=True))
import matplotlib.pyplot as plt
colors = ['skyblue', 'salmon']
df['Class'].value_counts().plot(
    kind='bar',
    title='Fraud vs Non-Fraud',
    color=colors
)
plt.xlabel("Class (0 = Non-Fraud, 1 = Fraud)")
plt.ylabel("Count")
plt.show()
```

```
<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
Class
0    284315
1       492
Name: count, dtype: int64
Fraud ratio: Class
0    0.998273
1    0.001727
Name: proportion, dtype: float64
```




```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['normAmount'] = scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['normTime'] = scaler.fit_transform(df['Time'].values.reshape(-1,1))
df = df.drop(['Time','Amount'], axis=1)
```

```
from sklearn.model_selection import train_test_split
X = df.drop('Class', axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
```

```
X_res, y_res = smote.fit_resample(X_train, y_train)
print("Before SMOTE:", y_train.value_counts())
print("After SMOTE:", y_res.value_counts())
```

```
Before SMOTE: Class
0    227451
1       394
Name: count, dtype: int64
After SMOTE: Class
0    227451
1    227451
Name: count, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
lr = LogisticRegression(max_iter=1000)
lr.fit(X_res, y_res)
y_pred = lr.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, lr.predict_proba(X_test)[:,1]))
```

```
[[55406  1458]
 [    8    90]]
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.97      0.99     56962

ROC-AUC: 0.9698482164390798
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=50, random_state=42) # Reduced estimators for faster training
rf.fit(X_res, y_res)
y_pred_rf = rf.predict(X_test)
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
print("ROC-AUC:", roc_auc_score(y_test, rf.predict_proba(X_test)[:,1]))
```

```
[[56849    15]
 [   17    81]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.84      0.83      0.84        98

    accuracy                           1.00     56962
   macro avg       0.92      0.91      0.92     56962
weighted avg       1.00      1.00      1.00     56962

ROC-AUC: 0.9648747494918057
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
model = models.Sequential([
    layers.Dense(32, activation='relu', input_dim=X_res.shape[1]),
    layers.Dropout(0.5),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_res, y_res, epochs=5, batch_size=2048, validation_split=0.2)
y_pred_nn = (model.predict(X_test) > 0.5).astype(int)
print(confusion_matrix(y_test, y_pred_nn))
print(classification_report(y_test, y_pred_nn))
```

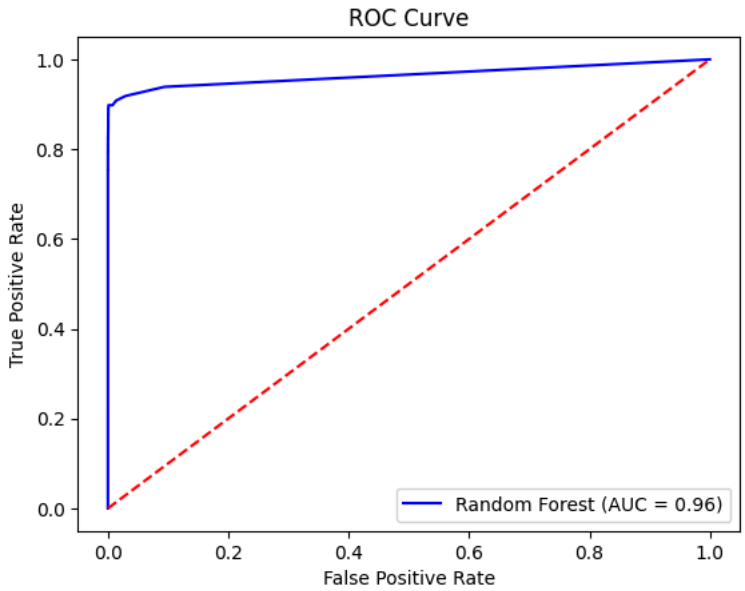
```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
178/178 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.6755 - loss: 0.5543 - val_accuracy: 0.8795 - val_loss: 0.2045
Epoch 2/5
178/178 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9472 - loss: 0.1459 - val_accuracy: 0.9108 - val_loss: 0.1674
Epoch 3/5
178/178 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9568 - loss: 0.1088 - val_accuracy: 0.9240 - val_loss: 0.1408
Epoch 4/5
178/178 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9629 - loss: 0.0916 - val_accuracy: 0.9364 - val_loss: 0.1240
Epoch 5/5
178/178 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9681 - loss: 0.0795 - val_accuracy: 0.9544 - val_loss: 0.1063
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 2s 987us/step
[[56464   400]
```

```
[   11    87]]
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56864
           1       0.18      0.89      0.30        98

    accuracy                           0.99     56962
   macro avg       0.59      0.94      0.65     56962
weighted avg       1.00      0.99      1.00     56962
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
fpr, tpr, _ = roc_curve(y_test, rf.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='blue', label="Random Forest (AUC = %.2f)" % roc_auc)
plt.plot([0,1],[0,1],'--', color='red')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

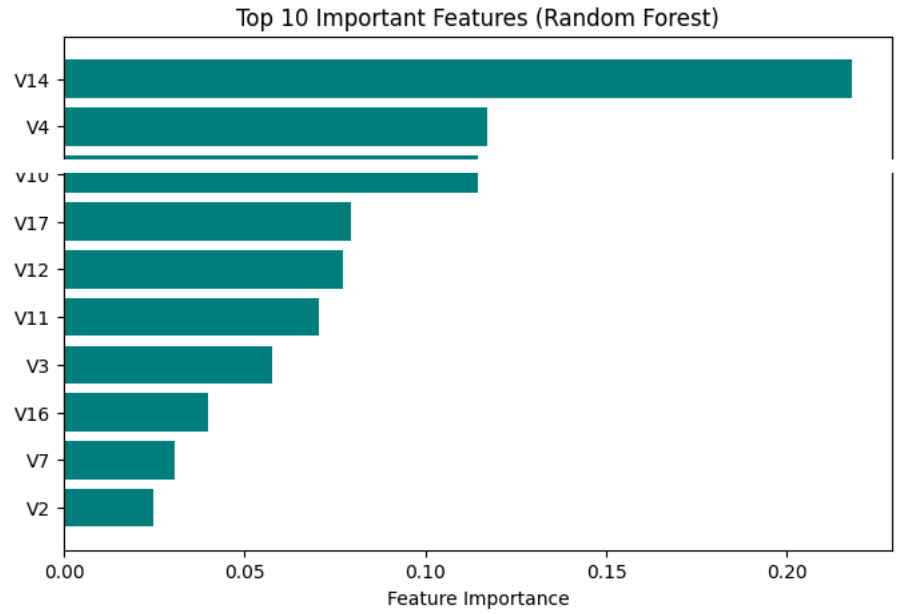



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import pandas as pd
def evaluate_model(name, y_true, y_pred, y_proba):
    return {
        "Model": name,
        "Accuracy": accuracy_score(y_true, y_pred),
        "Precision": precision_score(y_true, y_pred),
        "Recall": recall_score(y_true, y_pred),
        "F1-score": f1_score(y_true, y_pred),
        "ROC-AUC": roc_auc_score(y_true, y_proba)
    }
results = []
results.append(evaluate_model("Logistic Regression", y_test, y_pred, lr.predict_proba(X_test)[:,1]))
results.append(evaluate_model("Random Forest", y_test, y_pred_rf, rf.predict_proba(X_test)[:,1]))
results.append(evaluate_model("Neural Network", y_test, y_pred_nn, model.predict(X_test).ravel()))
metrics_df = pd.DataFrame(results)
print(metrics_df)
```

```
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step
                Model  Accuracy  Precision    Recall  F1-score   ROC-AUC
0  Logistic Regression  0.974264   0.058140  0.918367  0.109356  0.969848
1        Random Forest  0.999438   0.843750  0.826531  0.835052  0.964875
2       Neural Network  0.992785   0.178645  0.887755  0.297436  0.978072
```

```
import numpy as np
import matplotlib.pyplot as plt
importances = rf.feature_importances_
indices = np.argsort(importances)[-10:]
plt.figure(figsize=(8,5))
plt.barh(range(len(indices)), importances[indices], align='center', color='teal')
plt.yticks(range(len(indices)), [X.columns[i] for i in indices])
```

```
plt.xlabel("Feature Importance")
plt.title("Top 10 Important Features (Random Forest)")
plt.show()
```




```
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr.predict_proba(X_test)[:,1])
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf.predict_proba(X_test)[:,1])
fpr_nn, tpr_nn, _ = roc_curve(y_test, model.predict(X_test).ravel())
plt.figure(figsize=(8,6))
plt.plot(fpr_lr, tpr_lr, label="Logistic Regression (AUC=%.2f)" % auc(fpr_lr, tpr_lr), color='blue')
plt.plot(fpr_rf, tpr_rf, label="Random Forest (AUC=%.2f)" % auc(fpr_rf, tpr_rf), color='green')
plt.plot(fpr_nn, tpr_nn, label="Neural Network (AUC=%.2f)" % auc(fpr_nn, tpr_nn), color='purple')
plt.plot([0,1],[0,1],'--', color='red')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.show()
```

```
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
```