Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, RocCurveDisplay
from imblearn.over_sampling import SMOTE

# Load dataset
df = pd.read_csv("creditcard.csv")
df.head()
```

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

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
```

```
[5 rows x 31 columns]
```

```
df.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
```

```
df.describe()
```

```
                Time            V1            V2            V3
V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
```

```
mean    94813.859575  1.175161e-15  3.384974e-16 -1.379537e-15
2.094852e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00
1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -
5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -
8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -
1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00
7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00
1.687534e+01

                 V5            V6            V7            V8
V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean   1.021879e-15  1.494498e-15 -5.620335e-16  1.149614e-16 -
2.414189e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01
1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.628620e-16 -3.576577e-16  2.618565e-16  4.473914e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean   5.109395e-16  1.686100e-15 -3.661401e-16 -1.227452e-16
88.349619
```

```
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

```
df['Class'].value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

Data Cleaning

```
# Check missing values
df.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
```

```
# Drop duplicates if any
df.drop_duplicates(inplace=True)
```

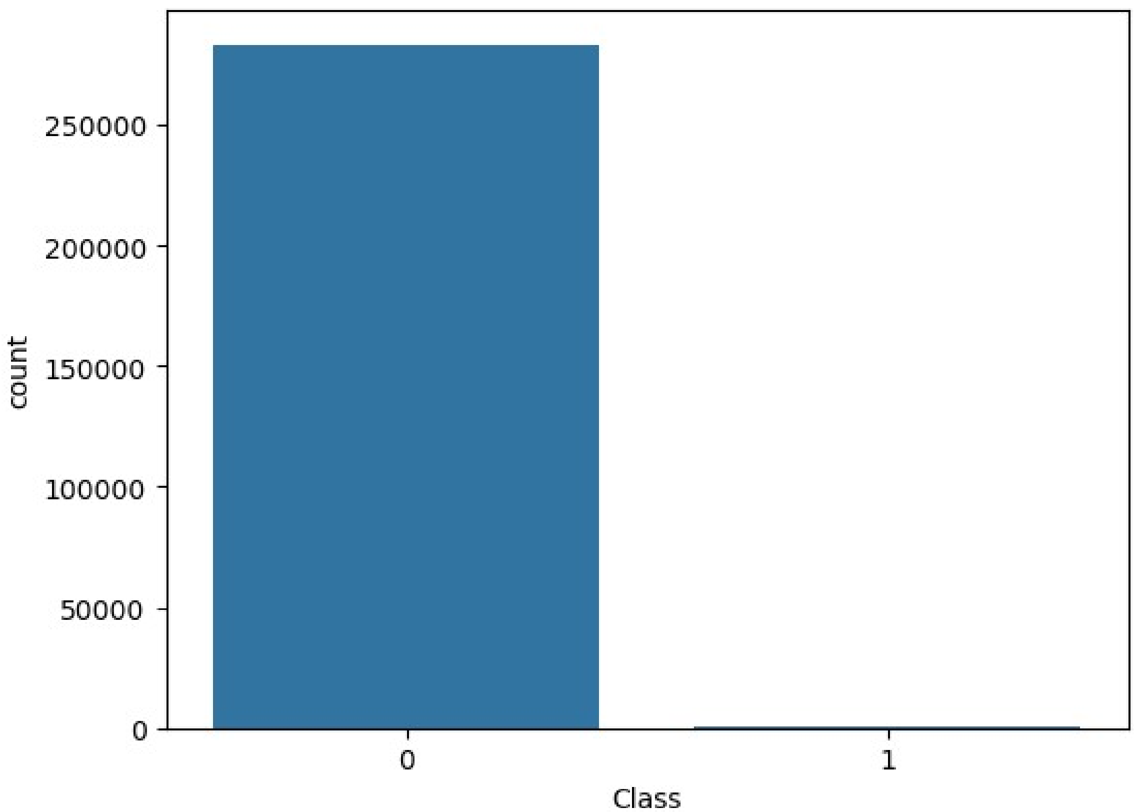
```
(df.isnull().sum() / len(df)) * 100
```

```
Time      0.0
V1        0.0
V2        0.0
V3        0.0
V4        0.0
V5        0.0
V6        0.0
V7        0.0
V8        0.0
V9        0.0
V10       0.0
V11       0.0
V12       0.0
V13       0.0
V14       0.0
V15       0.0
V16       0.0
V17       0.0
V18       0.0
V19       0.0
V20       0.0
V21       0.0
V22       0.0
V23       0.0
V24       0.0
```

```
V25        0.0
V26        0.0
V27        0.0
V28        0.0
Amount     0.0
Class      0.0
dtype: float64
```

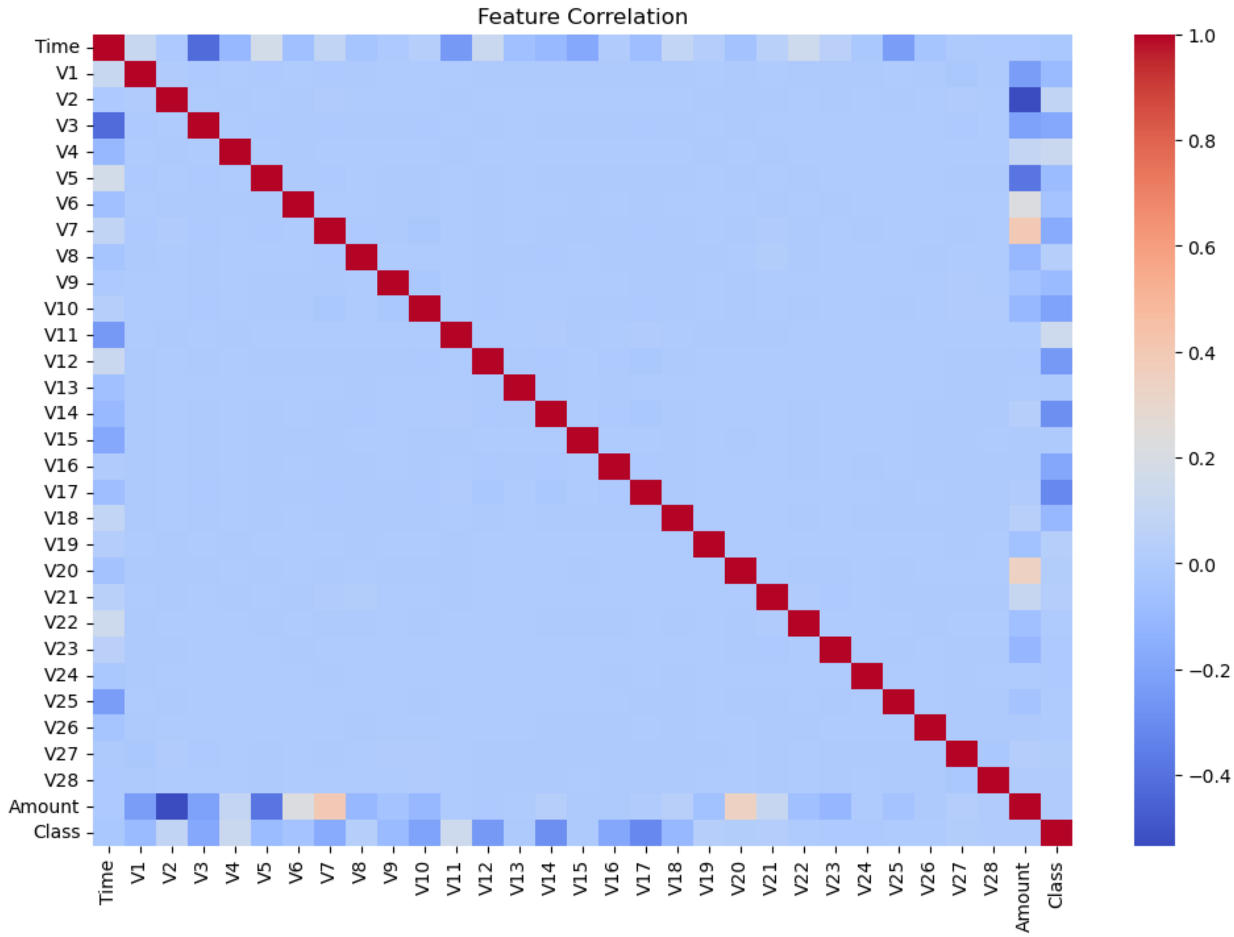
Exploratory Data Analysis (EDA)

```
#Class Distribution
sns.countplot(data=df,x='Class')
plt.show()
```




```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=False,cmap='coolwarm')
plt.title("Feature Correlation")

Text(0.5, 1.0, 'Feature Correlation')
```



Handling Class Imbalance

```
#split data x and y
x=df.drop('Class',axis=1)
y=df['Class']

# Split dataset
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=42,stratify=y)

#smote use for balancing data as over bound
smote=SMOTE(random_state=42)
x_train_res,y_train_res=smote.fit_resample(x_train,y_train)

y_train_res.value_counts()

Class
0    226602
1    226602
Name: count, dtype: int64
```

Feature Scaling

```
#scalling the function
scaler=StandardScaler()
x_train_res_scaled=scaler.fit_transform(x_train_res)
x_test_scaled=scaler.transform(x_test)
#fit_transform() -- Learn scaling parameters & apply them
#transform()-- Apply same scaling as training (no new fitting),if we
fit in test data then model fail on real word data
```

Model Building

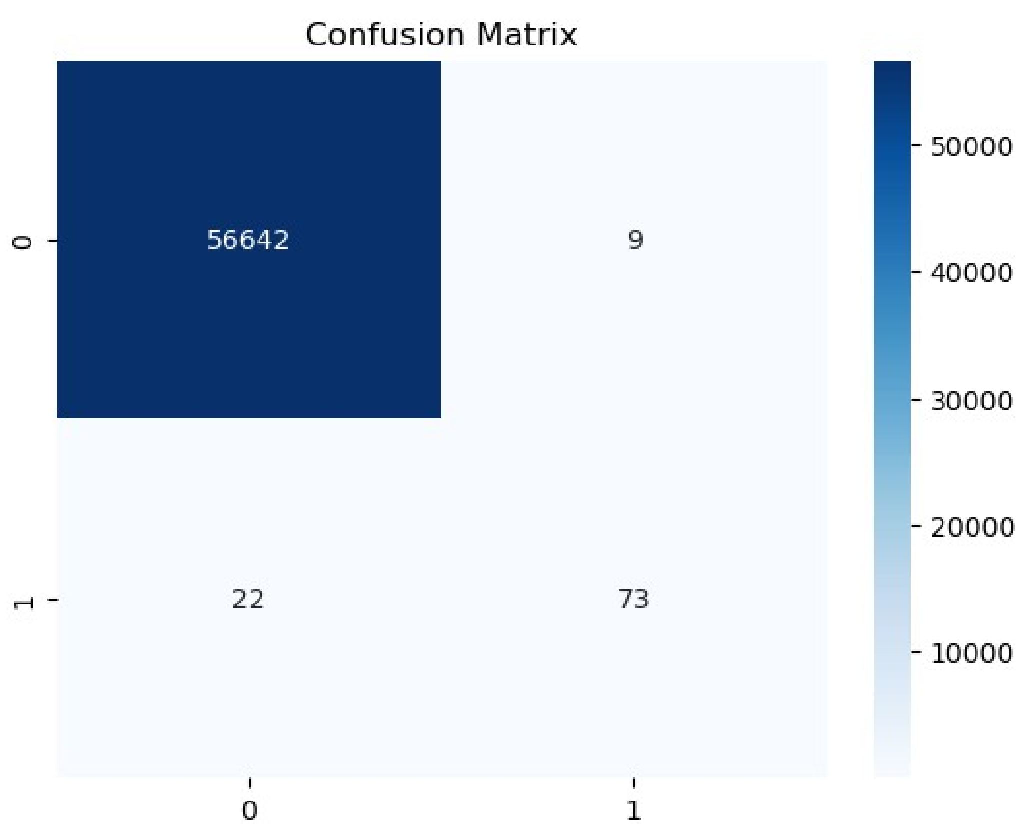
```
rf=RandomForestClassifier( n_estimators=100,random_state=42)
rf.fit(x_train_res_scaled,y_train_res)
```

```
RandomForestClassifier(random_state=42)
```

```
y_pred = rf.predict(x_test_scaled)
y_prob = rf.predict_proba(x_test_scaled)[:,1]
#fraud predicted value and prbability value stored in variable
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```



in above w show True negative values is coming 56642,false postive 9,false negative 22 and true postive is 73

```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.89      0.77      0.82        95

    accuracy                           1.00     56746
   macro avg       0.94      0.88      0.91     56746
weighted avg       1.00      1.00      1.00     56746
```

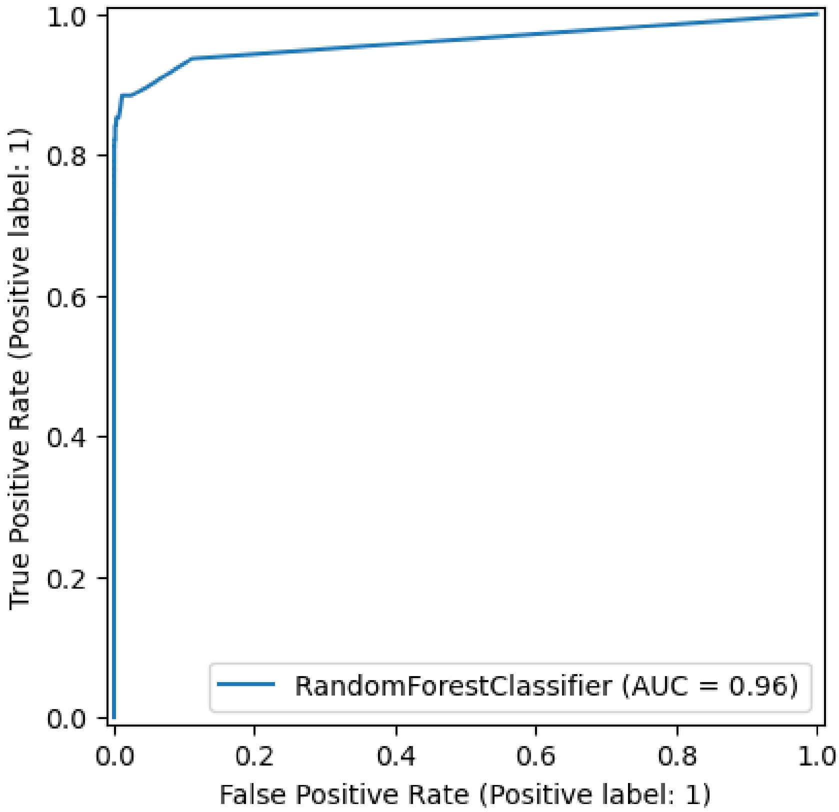
in above undestand predicted frauds that were correct 89% and 77% of actual fraud detect

```
# ROC-AUC
roc_auc_score(y_test,y_prob)
```

```
0.9608938941942773
```

```
RocCurveDisplay.from_estimator(rf,x_test_scaled,y_test)
```

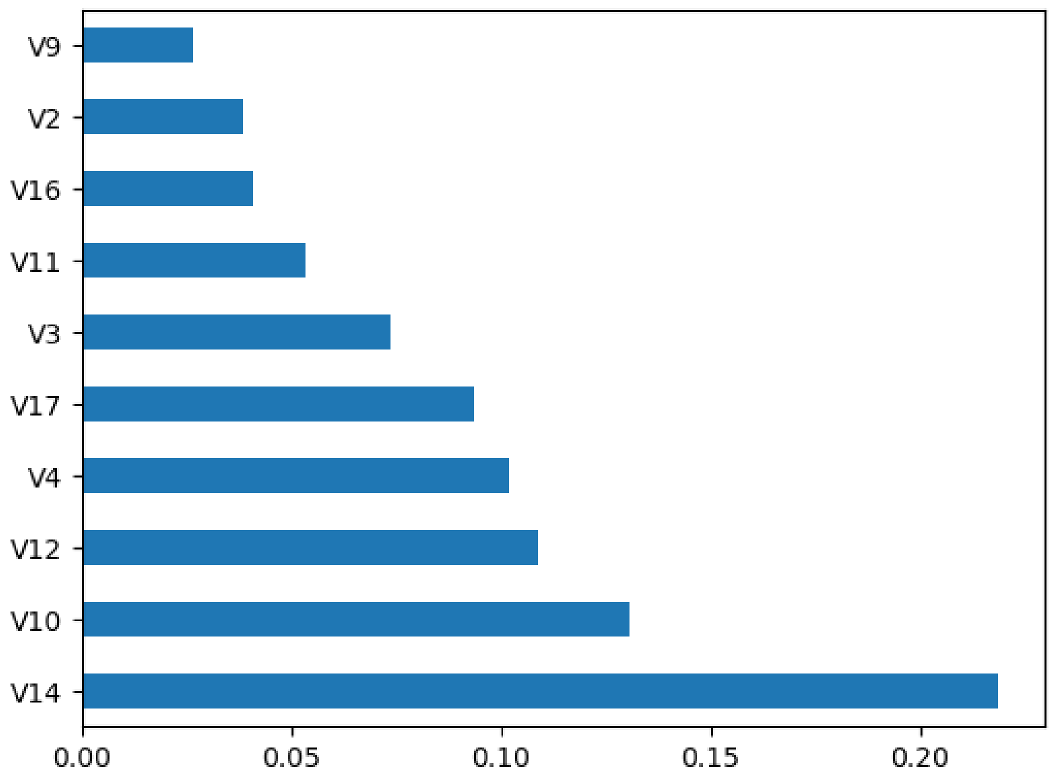
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2532494c440>
```



its show how model can seprate the two classes, 96% chance that model will assign a higher fraud probability to an actual fraudulent transaction than to a non-fraudulent one.

```
feat_imp=pd.Series(rf.feature_importances_,index=x.columns)
feat_imp.nlargest(10).plot(kind='barh')
```

```
<Axes: >
```



in above show that v14 is main fraud indicator, V10–V12–V4–V17 is a Important predictors and on remainig features Model doesn't rely on them means V14, V10, V12, V4, V17 these are top fraud signal features.