In [1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 500)
```

In [2]:

```
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

In [3]:

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

Out[3]:

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

In [4]:

```
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, train_size=0.8, test_size=0.2,
```

In [5]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
X_train.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7
201788	134039.0	2.023734	-0.429219	-0.691061	-0.201461	-0.162486	0.283718	-0.674694
179369	124044.0	-0.145286	0.736735	0.543226	0.892662	0.350846	0.089253	0.626708
73138	54997.0	-3.015846	-1.920606	1.229574	0.721577	1.089918	-0.195727	-0.462586
208679	137226.0	1.851980	-1.007445	-1.499762	-0.220770	-0.568376	-1.232633	0.248573
206534	136246.0	2.237844	-0.551513	-1.426515	-0.924369	-0.401734	-1.438232	-0.119942

In [6]:

```
X_test['Amount'] = scaler.transform(X_test[['Amount']])
X_test.head()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7
49089	43906.0	1.229452	-0.235478	-0.627166	0.419877	1.797014	4.069574	-0.896223
154704	102638.0	2.016893	-0.088751	-2.989257	-0.142575	2.675427	3.332289	-0.652336
67247	52429.0	0.535093	-1.469185	0.868279	0.385462	-1.439135	0.368118	-0.499370
251657	155444.0	2.128486	-0.117215	-1.513910	0.166456	0.359070	-0.540072	0.116023
201903	134084.0	0.558593	1.587908	-2.368767	5.124413	2.171788	-0.500419	1.059829

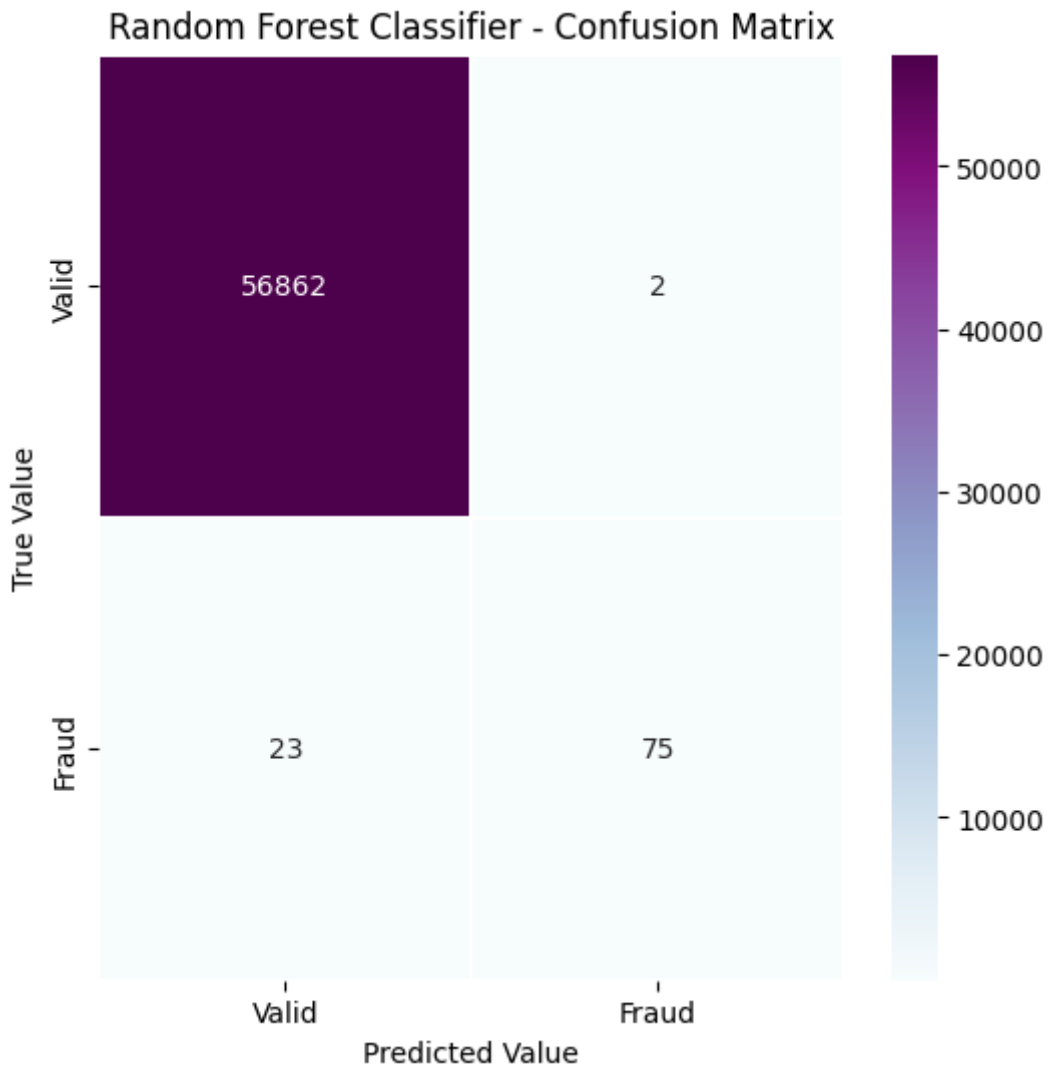
In [7]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state =
from sklearn.metrics import (confusion_matrix, roc_curve, classification_report, precisi
```

In [15]:

```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier( random_state = 42 )
rfc.fit(X_train, y_train)
Y_pred=rfc.predict(X_test)

print("Model Accuracy:", round(accuracy_score(y_test, Y_pred),2))
print("Model Precision:", round(precision_score(y_test, Y_pred),2))
print("Model F1-Score:", round(f1_score(y_test, Y_pred),2))
print("Model ROC:", round(roc_auc_score(y_test, Y_pred),2) , '\n')
conf_matrix=confusion_matrix(y_test, Y_pred)
labels= ['Valid', 'Fraud']
plt.figure(figsize=(6, 6))

sns.heatmap(pd.DataFrame(conf_matrix), xticklabels= labels, yticklabels= labels,
            linewidths= 0.05 ,annot=True, fmt="d" , cmap='BuPu')


plt.title("Random Forest Classifier - Confusion Matrix")
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```

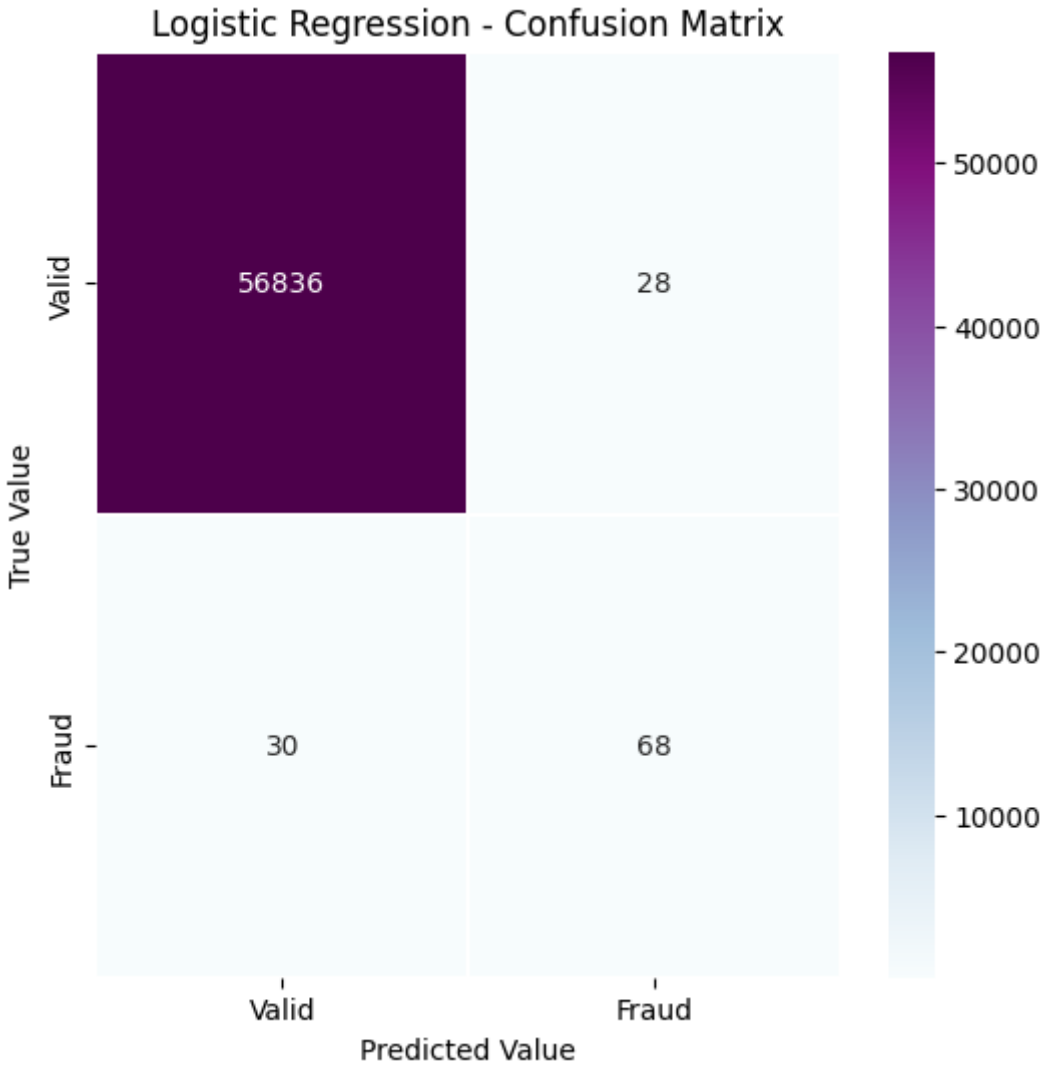
```
Model Accuracy: 1.0
Model Precision: 0.97
Model F1-Score: 0.86
Model ROC: 0.88
```


Random Forest Classifier - Confusion Matrix
56862
2
23
75
Valid
Fraud
True Value
Valid
Fraud
Predicted Value
50000
40000
30000
20000
10000

In [18]:

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state = 42)
logreg.fit(X_train, y_train)
Y_pred1 = logreg.predict(X_test)

print("Model Accuracy:", round(accuracy_score(y_test, Y_pred1),2))
print("Model Precision:", (precision_score(y_test, Y_pred1),2))
print("Model Recall:", round(recall_score(y_test, Y_pred1),2))
print("Model F1-Score:", round(f1_score(y_test, Y_pred1),2) , '\n')


conf_matrix1 = confusion_matrix(y_test, Y_pred1)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix1),annot=True, fmt='d',
            linewidths= 0.05 ,cmap='BuPu',xticklabels= labels, yticklabels= labels)


plt.title('Logistic Regression - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```

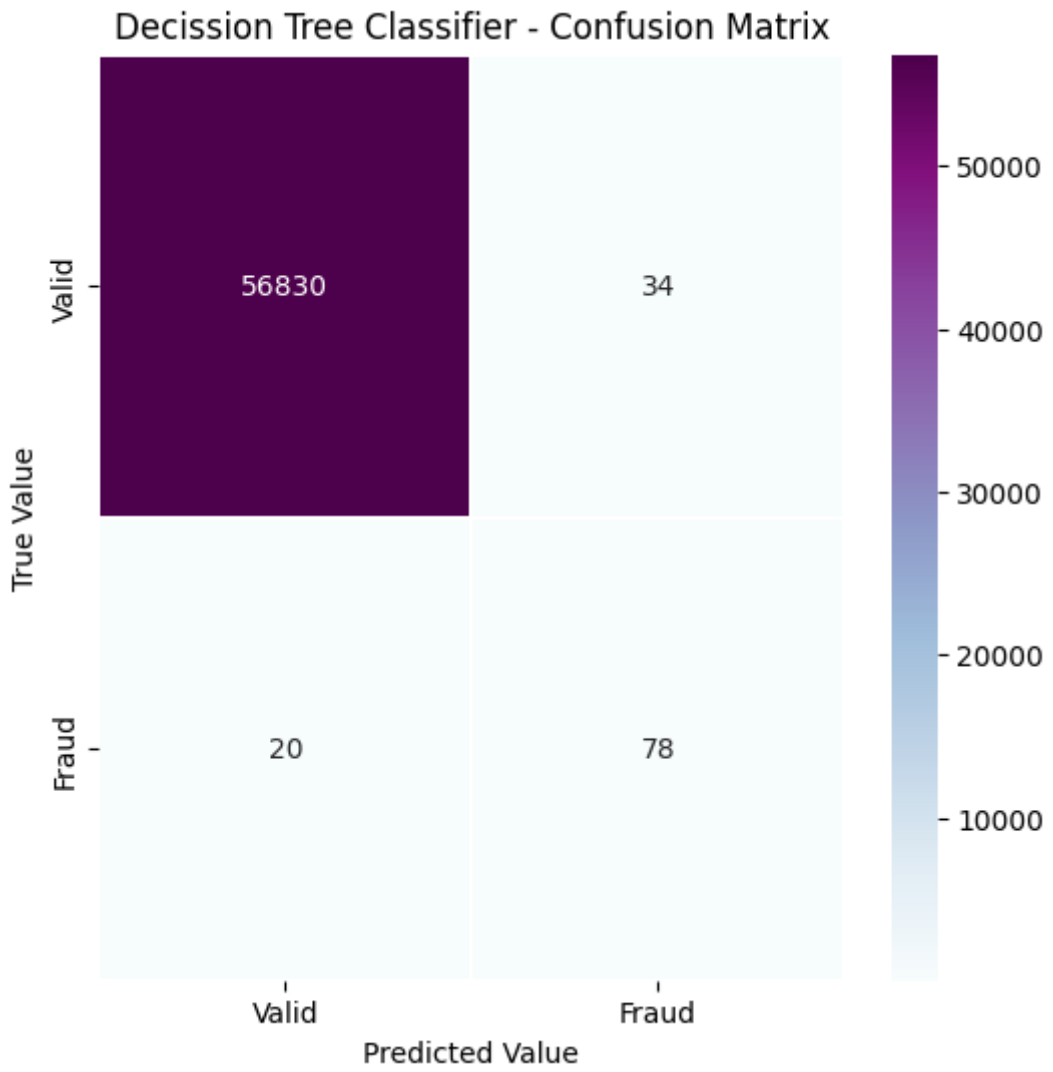
```
Model Accuracy: 1.0
Model Precision: 0.71
Model Recall: 0.69
Model F1-Score: 0.7
```


Logistic Regression - Confusion Matrix
56836
28
30
68
Valid
Fraud
True Value
Valid
Fraud
Predicted Value
50000
40000
30000
20000
10000

In [9]:

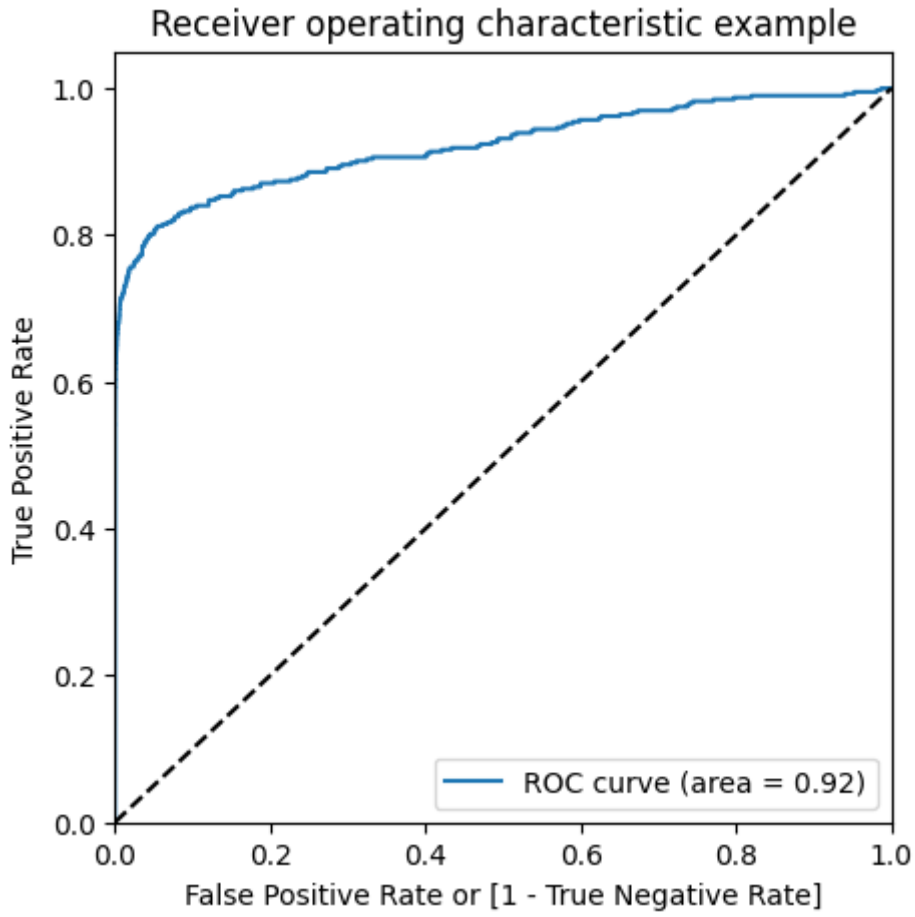
```
# DECISSION TREE CLASSIFIER MODEL

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(random_state = 42)
dtc.fit(X_train,y_train)
root_node = np.argmax(dtc.tree_.feature == -2)
print("Root node :", root_node)

Y_pred2 = dtc.predict(X_test)
conf_matrix2 = confusion_matrix(y_test , Y_pred2)

print("Model Accuracy:", round(accuracy_score(y_test, Y_pred2),2))
print("Model Precision:", round(precision_score(y_test, Y_pred2),2))
print("Model Recall:", round(recall_score(y_test, Y_pred2),2))
print("Model F1-Score:", round(f1_score(y_test, Y_pred2),2) , '\n')

conf_matrix2 = confusion_matrix(y_test, Y_pred2)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix2),annot=True, fmt='d',linewidths= 0.05 ,cmap='BuPu'
            xticklabels= labels, yticklabels= labels)


plt.title('Decission Tree Classifier - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```

```
Root node : 7
Model Accuracy: 1.0
Model Precision: 0.7
Model Recall: 0.8
Model F1-Score: 0.74
```


Decission Tree Classifier - Confusion Matrix
56830
34
20
78
Valid
Fraud
True Value
Valid
Fraud
Predicted Value
50000
40000
30000
20000
10000

In [23]:

```
logistic_imb = LogisticRegression(C=0.01)
logistic_imb_model = logistic_imb.fit(X_train, y_train)
y_train_pred = logistic_imb_model.predict(X_train)

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
y_train_pred_proba = logistic_imb_model.predict_proba(X_train)[:,1]
draw_roc(y_train, y_train_pred_proba)
```



In [24]:

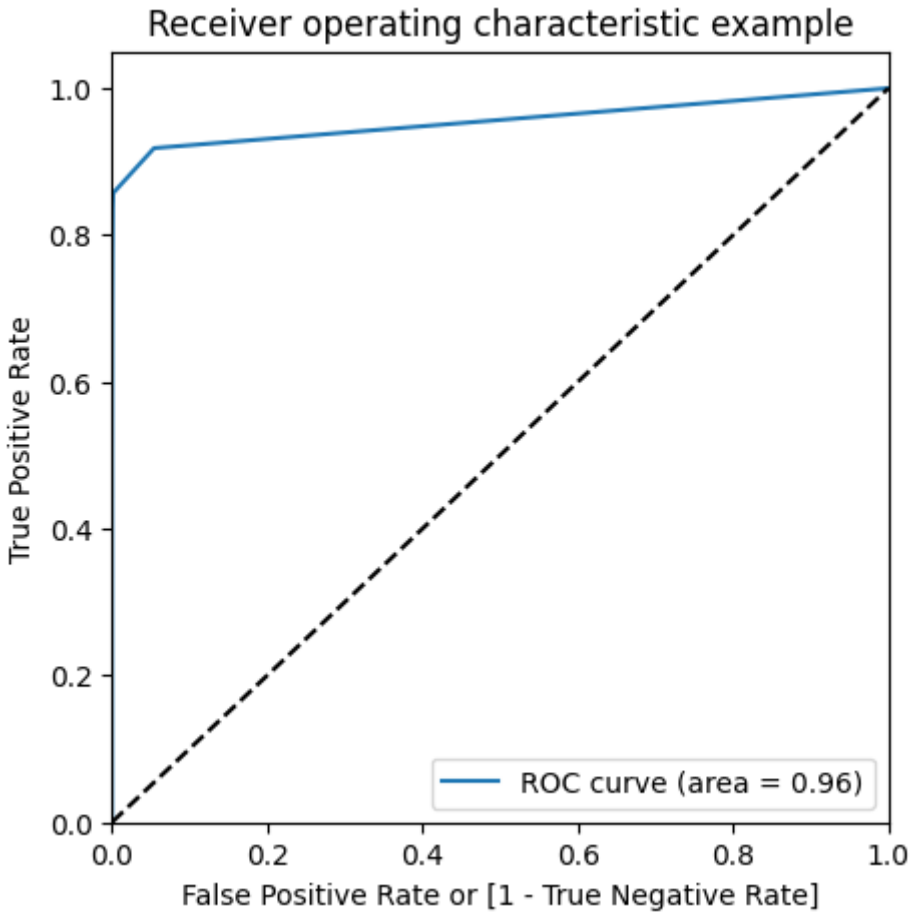
```
from sklearn.tree import DecisionTreeClassifier
dt_imb_model = DecisionTreeClassifier(criterion = "gini",
                                      random_state = 100,
                                      max_depth=5,
                                      min_samples_leaf=100,
                                      min_samples_split=100)

dt_imb_model.fit(X_train, y_train)
y_test_pred = dt_imb_model.predict(X_test)
y_test_pred_proba = dt_imb_model.predict_proba(X_test)[:,1]
auc = metrics.roc_auc_score(y_test, y_test_pred_proba)
auc
```

Out[24]:

```
0.9550532491415248
```

In [26]:

```
draw_roc(y_test, y_test_pred_proba)
```



In [8]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='ball_tree',n_neighbors = 5,metric='euclidean')
knn.fit(X_train, y_train)
```

Out[8]:

```
KNeighborsClassifier
KNeighborsClassifier(algorithm='ball_tree', metric='euclidean')
```

In [9]:

```
y_pred = knn.predict(X_test)
y_pred[5000:6000]
```

Out[9]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [10]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[56864     0]
 [   93     5]]
```

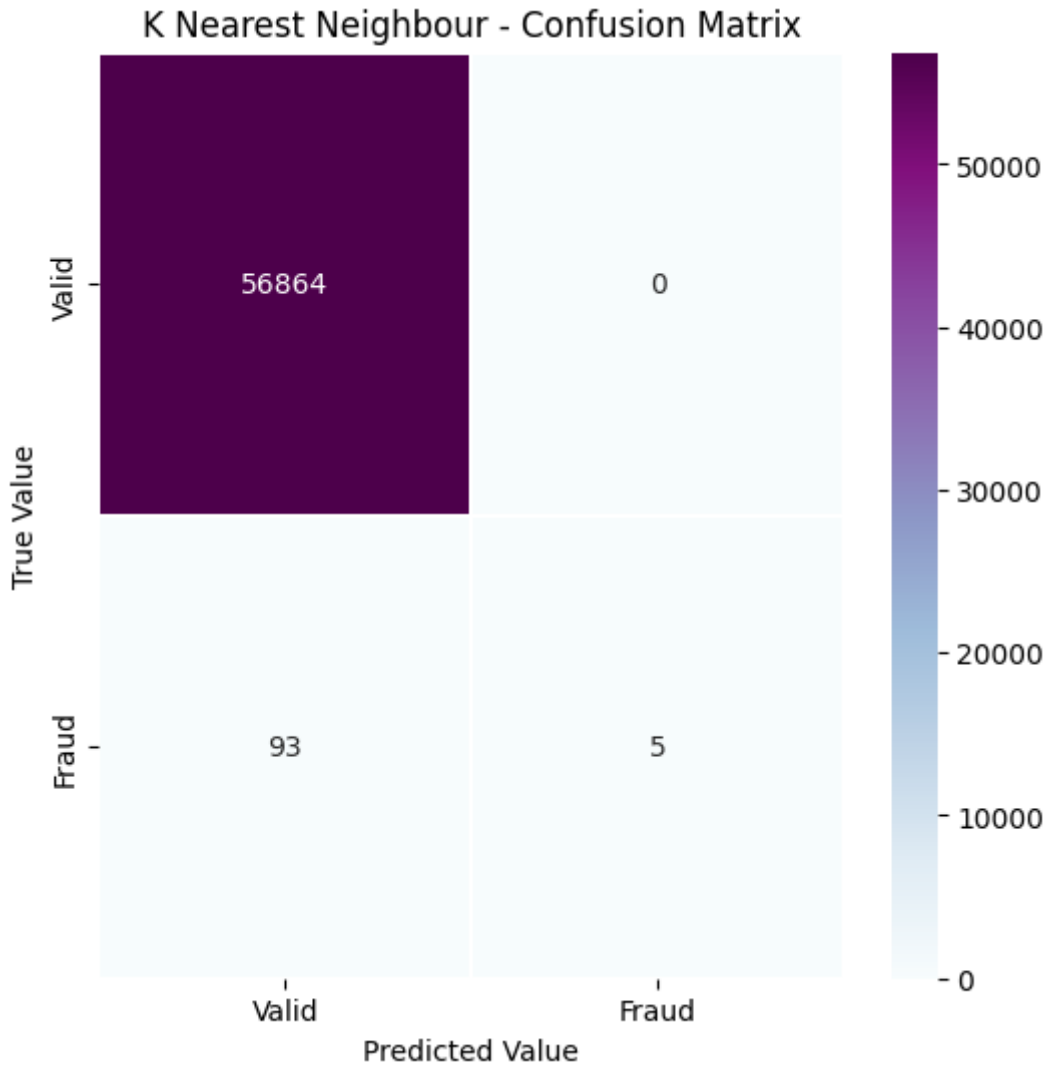
In [11]:

```
acc_knn =accuracy_score(y_test, y_pred)*100
print("The accuracy is",acc_knn,"%")
```

```
The accuracy is 99.83673326077034 %
```

In [12]:

```
Y_pred1 = knn.predict(X_test)
conf_matrix1 = confusion_matrix(y_test, Y_pred1)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix1),annot=True, fmt='d',
            linewidths= 0.05 ,cmap='BuPu',xticklabels= labels, yticklabels= labels)

plt.title('K Nearest Neighbour - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```

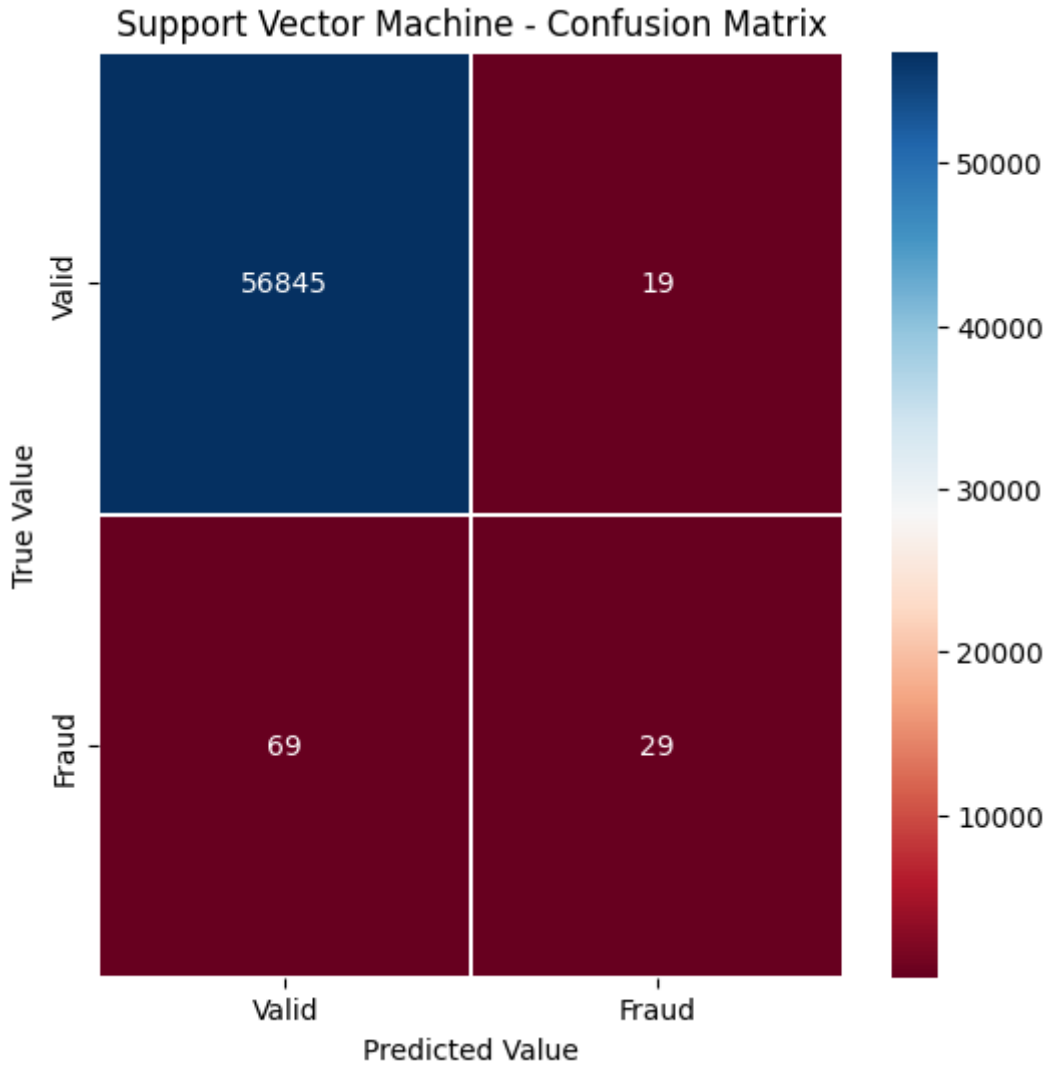


In [*]:

```
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Create an SVM model
model = svm.SVC(kernel='linear', C=1.0, random_state=42)
# Train the SVM model
model.fit(X_train, y_train)
# Evaluate the SVM model on the testing set
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test,y_pred))
```

In [17]:

```
y_pred = model.predict(X_test)
conf_matrix1 = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix1),annot=True, fmt='d',
            linewidths= 0.05 ,cmap='RdBu',xticklabels= labels, yticklabels= labels)

plt.title('Support Vector Machine - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```



In [ ]: