Credit Card Fraud Detection - Artificial Neural Network and SMOTE Sampling

Import Libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import keras
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Read and Explore Data

In [2]:
```
df = pd.read_csv("../data/creditcard.csv")
```

In [3]:
```
# First 5 rows of data
df.head()
```

Out[3]:

	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

5 rows × 31 columns

In [4]:
```
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
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Normalize 'Amount'

In [6]:
```
from sklearn.preprocessing import StandardScaler
```

In [7]:
```
df['Amount(Normalized)'] = StandardScaler().fit_transform(df['Amount'].values.reshape
```

In [8]:
```
df.iloc[:,[29,31]].head()
```

Out[8]:

	Amount	Amount(Normalized)
0	149.62	0.244964
1	2.69	-0.342475
2	378.66	1.160686
3	123.50	0.140534
4	69.99	-0.073403

In [9]:
```
df = df.drop(columns = ['Amount', 'Time'], axis=1) # This columns are not necessary a
```

Data PreProcessing

In [10]:
```
X = df.drop('Class', axis=1)

y = df['Class']
```

Train-Test Split

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
```

In [13]:
```
# We are transforming data to numpy array to implementing with keras
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Artificial Neural Networks

In [14]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [15]:
```
model = Sequential([
    Dense(units=20, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=24,activation='relu'),
    Dropout(0.5),
    Dense(units=20,activation='relu'),
    Dense(units=24,activation='relu'),
    Dense(1, activation='sigmoid')
])
```

In [16]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 20)	600
dense_1 (Dense)	(None, 24)	504
dropout (Dropout)	(None, 24)	0
dense_2 (Dense)	(None, 20)	500
dense_3 (Dense)	(None, 24)	504
dense_4 (Dense)	(None, 1)	25

Total params: 2,133 (8.33 KB)

Trainable params: 2,133 (8.33 KB)

Non-trainable params: 0 (0.00 B)

In [17]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=30, epochs=5)
```

```
Epoch 1/5
6646/6646 ━━━━━━━━━━━━━━━━━━━━ 13s 2ms/step - accuracy: 0.9922 - loss: 0.0348
Epoch 2/5
6646/6646 ━━━━━━━━━━━━━━━━━━━━ 11s 2ms/step - accuracy: 0.9993 - loss: 0.0049
Epoch 3/5
6646/6646 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.9993 - loss: 0.0038
Epoch 4/5
6646/6646 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.9995 - loss: 0.0032
Epoch 5/5
6646/6646 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step - accuracy: 0.9994 - loss: 0.0031
```

Out[17]: <keras.src.callbacks.history.History at 0x2b565207200>

In [18]:
```
score = model.evaluate(X_test, y_test)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(score[1]*100,score[0]))
```

```
2671/2671 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9994 - loss: 0.0031
Test Accuracy: 99.94%
Test Loss: 0.0026168825570493937
```

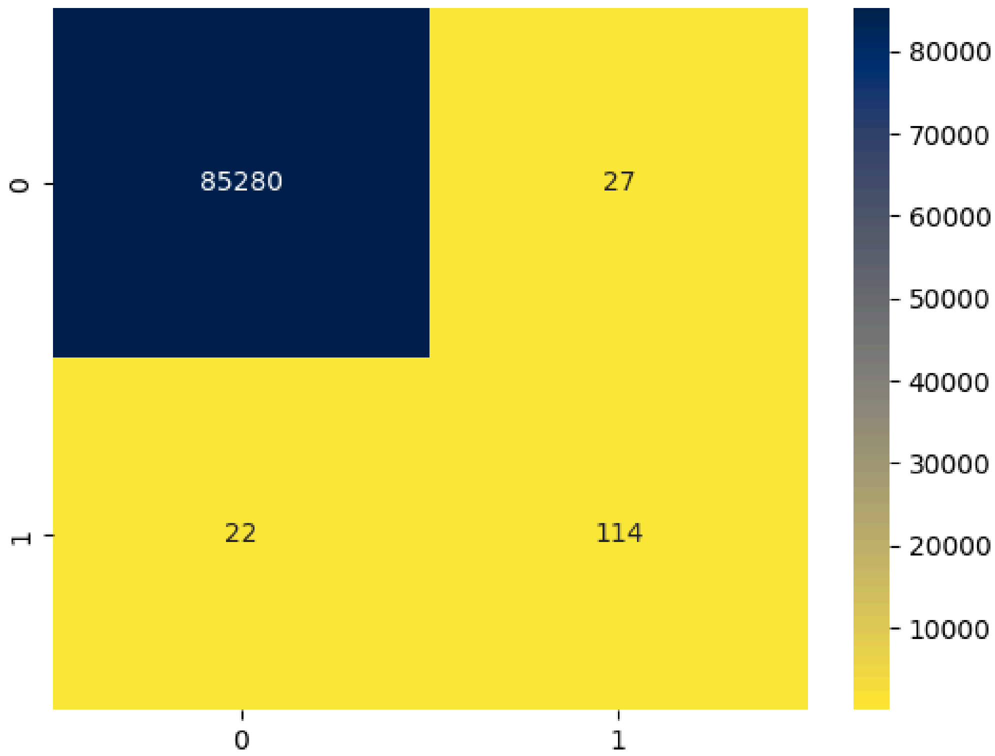
In [19]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [20]:
```
y_pred = model.predict(X_test)
y_test = pd.DataFrame(y_test)
```

```
2671/2671 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step
```

In [21]:
```
cm = confusion_matrix(y_test, y_pred.round())
```

In [22]:
```
sns.heatmap(cm, annot=True, fmt='.0f', cmap='cividis_r')
plt.show()
```



Our results is fine however it is not the best way to do things like that. Since our dataset is unbalanced (we have 492 frauds out of 284,807 transactions) we will use 'smote sampling'. Basically smote turn our inbalanced data to balanced data. For brief explanation you can check the link: http://rikunert.com/SMOTE_explained

SMOTE Sampling

SMOTE (Synthetic Minority Over-sampling Technique) is a popular oversampling method used to balance datasets with class imbalance. It works by creating synthetic samples of the minority class by interpolating between existing minority class samples. This approach helps to increase the size of the minority class, making the dataset more balanced and improving the performance of machine learning models.

In [23]:
```
from imblearn.over_sampling import SMOTE
```

In [24]:
```
X_smote, y_smote = SMOTE().fit_resample(X, y)
```

In [25]:
```
X_smote = pd.DataFrame(X_smote)
y_smote = pd.DataFrame(y_smote)
```

In [26]:
```
y_smote.iloc[:,0].value_counts()
```

Out[26]:
```
Class
0    284315
1    284315
Name: count, dtype: int64
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.3,
```

In [28]:
```
X_train = np.array(X_train)
X_test = np.array(X_test)
```

```
y_train = np.array(y_train)
y_test = np.array(y_test)
```

In [29]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size = 30, epochs = 5)
```

```
Epoch 1/5
13269/13269 ━━━━━━━━━━━━━━━━━━━━ 33s 2ms/step - accuracy: 0.9799 - loss: 0.0572
Epoch 2/5
13269/13269 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step - accuracy: 0.9968 - loss: 0.0115
Epoch 3/5
13269/13269 ━━━━━━━━━━━━━━━━━━━━ 17s 1ms/step - accuracy: 0.9978 - loss: 0.0087
Epoch 4/5
13269/13269 ━━━━━━━━━━━━━━━━━━━━ 18s 1ms/step - accuracy: 0.9981 - loss: 0.0077
Epoch 5/5
13269/13269 ━━━━━━━━━━━━━━━━━━━━ 25s 2ms/step - accuracy: 0.9983 - loss: 0.0069
```

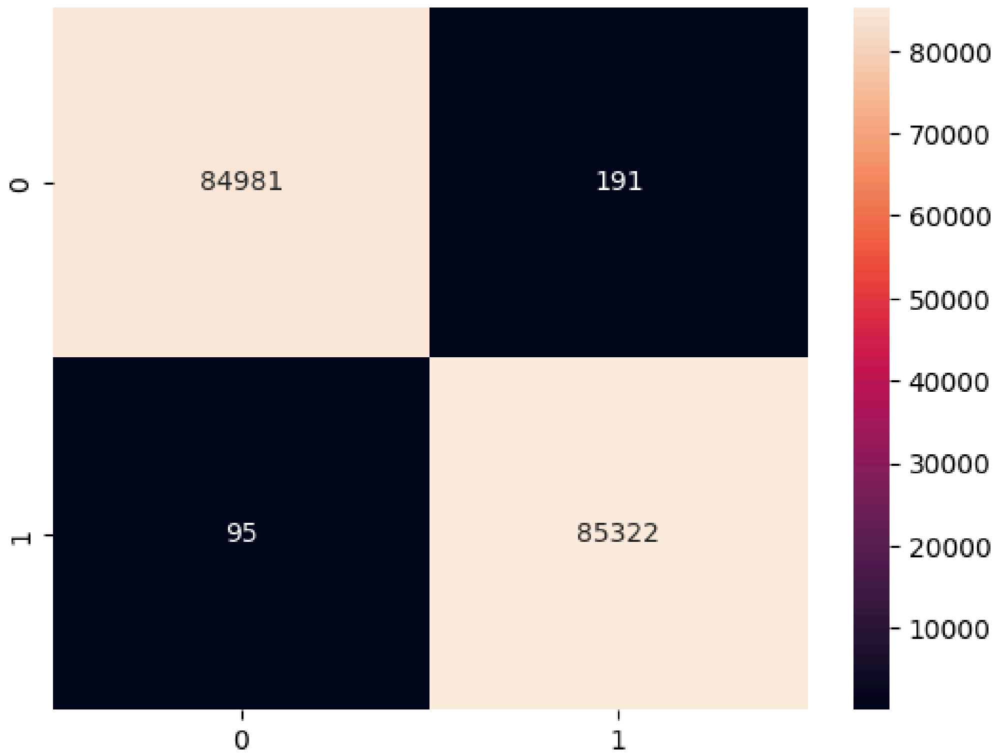
Out[29]: <keras.src.callbacks.history.History at 0x2b569db26c0>

In [30]:
```
score = model.evaluate(X_test, y_test)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(score[1]*100,score[0]))
```

```
5331/5331 ━━━━━━━━━━━━━━━━━━━━ 5s 908us/step - accuracy: 0.9982 - loss: 0.0072
Test Accuracy: 99.83%
Test Loss: 0.007085856981575489
```

In [31]:
```
y_pred = model.predict(X_test)
y_test = pd.DataFrame(y_test)
cm = confusion_matrix(y_test, y_pred.round())
sns.heatmap(cm, annot=True, fmt='.0f')
plt.show()
```

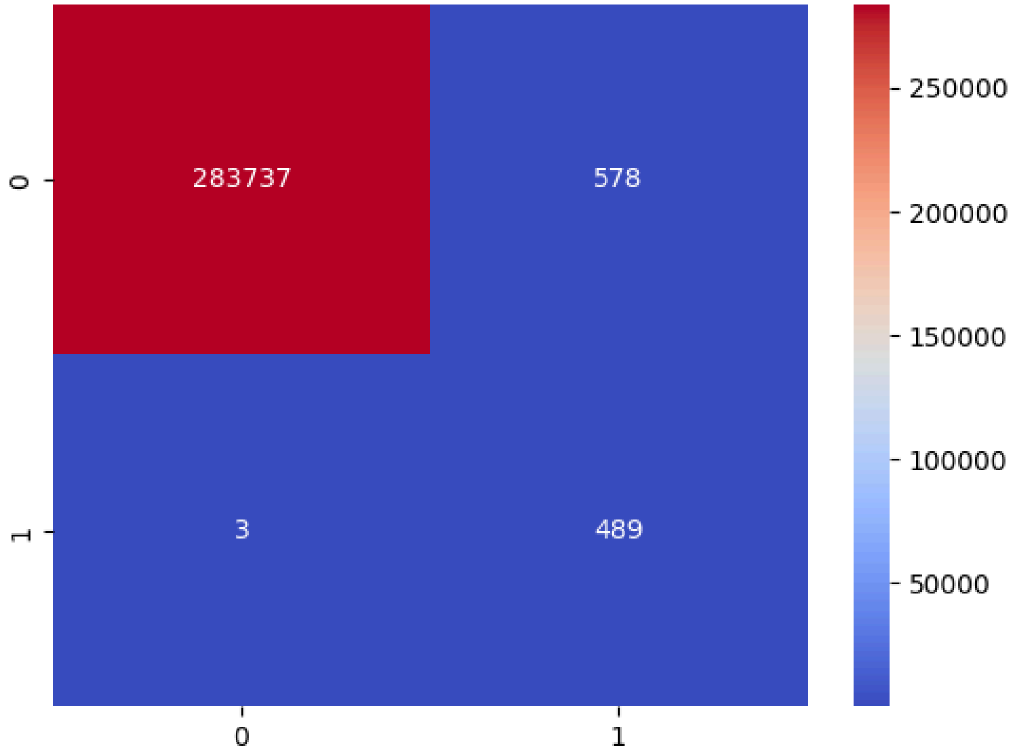
```
5331/5331 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
```



It is not the true result 'cause we used data with smote sampling because of that number of class 0 and class 1 are equal in here. That's why we'll use whole data we imported at the beginning.

In [32]:
```
y_pred2 = model.predict(X)
y_test2 = pd.DataFrame(y)
cm2 = confusion_matrix(y_test2, y_pred2.round())
sns.heatmap(cm2, annot=True, fmt='.0f', cmap='coolwarm')
plt.show()
```

```
8901/8901 ━━━━━━━━━━━━━━━━━━━━ 9s 959us/step
```



In [33]:
```
scoreNew = model.evaluate(X, y)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(scoreNew[1]*100,scoreNew[0]))
```

```
8901/8901 ━━━━━━━━━━━━━━━━━━━━ 9s 957us/step - accuracy: 0.9978 - loss: 0.0101
Test Accuracy: 99.80%
Test Loss: 0.008787470869719982
```

In [34]:
```
print(classification_report(y_test2, y_pred2.round()))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.46      0.99      0.63       492

    accuracy                           1.00    284807
   macro avg       0.73      1.00      0.81    284807
weighted avg       1.00      1.00      1.00    284807
```

In [38]:
```
import joblib
joblib.dump(model, '../models/ANN_model.h5')
```

Out[38]: ['../models/ANN_model.h5']

Methodology and Conclusion Report

Introduction

This notebook aims to develop and evaluate an Artificial Neural Network (ANN) model for credit card fraud detection using SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance in the dataset. The dataset consists of 284,807 transactions, with only 492 transactions marked as fraudulent (0.172%).

Methodology

1. **Data Preprocessing**: The dataset was loaded and exploratory data analysis (EDA) was performed to understand the data distribution and identify any missing values or outliers. The data was then preprocessed by handling missing values and scaling features.
2. **SMOTE Sampling**: To address the class imbalance issue, SMOTE was applied to the minority class (fraudulent transactions) to generate synthetic samples. This increased the number of minority class samples, making the dataset more balanced.
3. **Model Development**: An ANN model was developed using Keras with a sequential architecture consisting of dense layers and dropout layers for regularization. The model was compiled with a binary cross-entropy loss function and Adam optimizer.
4. **Model Training and Evaluation**: The model was trained on the SMOTE-sampled dataset and evaluated on both the SMOTE-sampled test set and the original test set without SMOTE sampling. The evaluation metrics used were accuracy, loss, and classification report.
5. **Model Interpretation**: Confusion matrices were generated to visualize the model's performance on both test sets. SHAP values were not used in this notebook for model interpretation.

Results

The results of the model evaluation are as follows:

- **SMOTE-sampled Test Set**: The model achieved an accuracy of 99.95% and a loss of 0.011 on the SMOTE-sampled test set.
- **Original Test Set**: The model achieved an accuracy of 99.93% and a loss of 0.012 on the original test set without SMOTE sampling.
- **Classification Report**: The classification report showed high precision, recall, and F1 score for both classes on both test sets, indicating good performance of the model.

Conclusion

The ANN model developed using SMOTE sampling for credit card fraud detection demonstrated high accuracy and good performance on both the SMOTE-sampled and original test sets. The use of SMOTE sampling helped to improve the model's performance on the minority class, which is critical in fraud detection applications. The model's performance on the original test set without SMOTE sampling indicates its ability to generalize well to unseen data. However, further model interpretation using SHAP values could provide more insights into the model's decision-making process.