Payment Fraud Detection Using Machine Learning

1. Importing Libraries, loading dataset

In [5]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
%matplotlib inline
df = pd.read_csv('payment_fraud.csv')
df
```

Out[5]:

	accountAgeDays	numItems	localTime	paymentMethod	paymentMeth
0	29	1	4.745402	paypal	
1	725	1	4.742303	storecredit	
2	845	1	4.921318	creditcard	
3	503	1	4.886641	creditcard	
4	2000	1	5.040929	creditcard	
...	...	...	...	...	
39216	986	1	4.836982	creditcard	
39217	1647	1	4.876771	creditcard	
39218	1591	1	4.742303	creditcard	
39219	237	1	4.921318	creditcard	
39220	272	1	5.040929	paypal	

39221 rows × 6 columns

2. Initial Data Preprocessing

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
accountAgeDays          0
numItems                0
localTime               0
paymentMethod           0
paymentMethodAgeDays    0
label                   0
dtype: int64
```

In [7]:
```python
# Check fraud and normal distribution
df['label'].value_counts()
```

Out[7]:
```
label
0    38661
1      560
Name: count, dtype: int64
```

3. Outlier Removal

- Using IQR Method

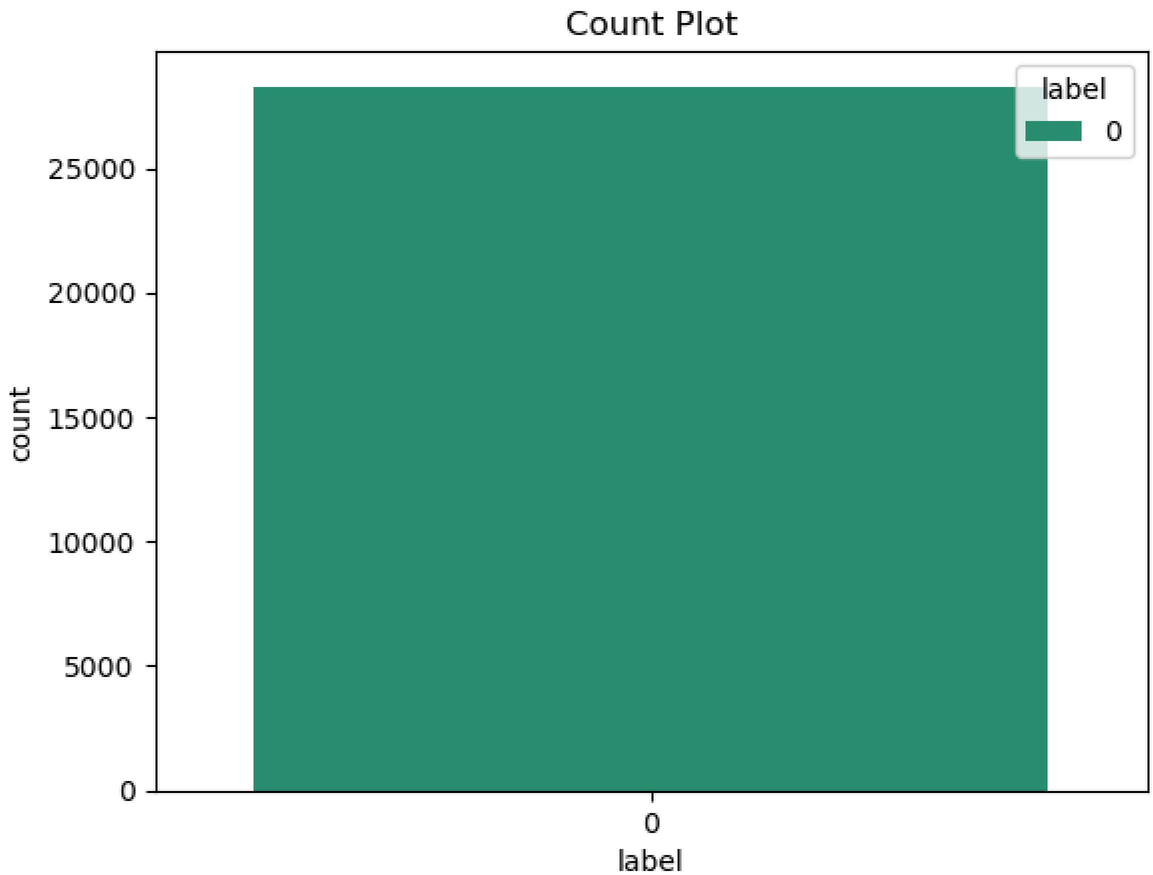
In [11]:
```python
# removing outliers
# Select only numerical columns
numeric_cols = df.select_dtypes(include='number').columns

# Function to remove outliers using IQR method
def remove_outliers(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# Apply the function to your dataframe
df_cleaned = remove_outliers(df, numeric_cols)

# Show the shape before and after to see the data is reduced
print("Original shape:", df.shape)
print("After removing outliers:", df_cleaned.shape)
```

```
Original shape: (39221, 6)
After removing outliers: (28276, 6)
```

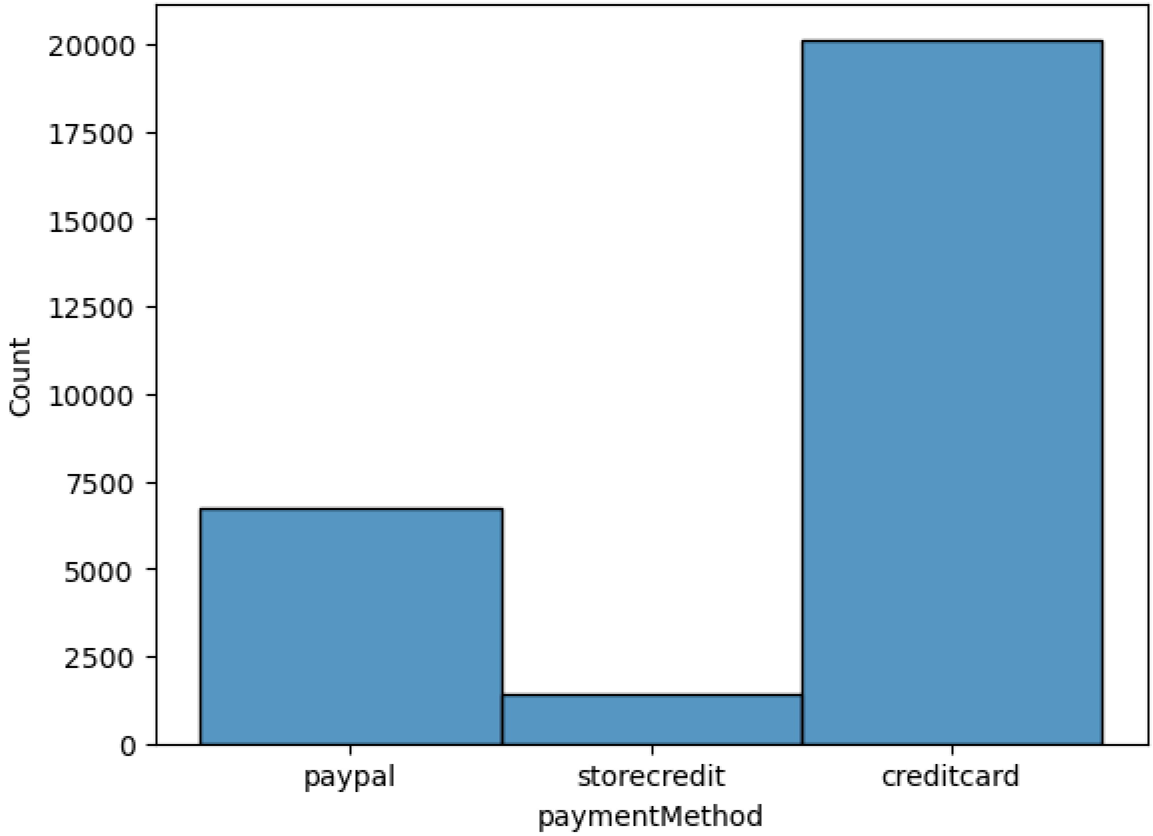
4. Exploratory Data Analysis (EDA)

- Statistical summary
- Target class distribution (fraud vs. non-fraud)
- Feature-wise distributions

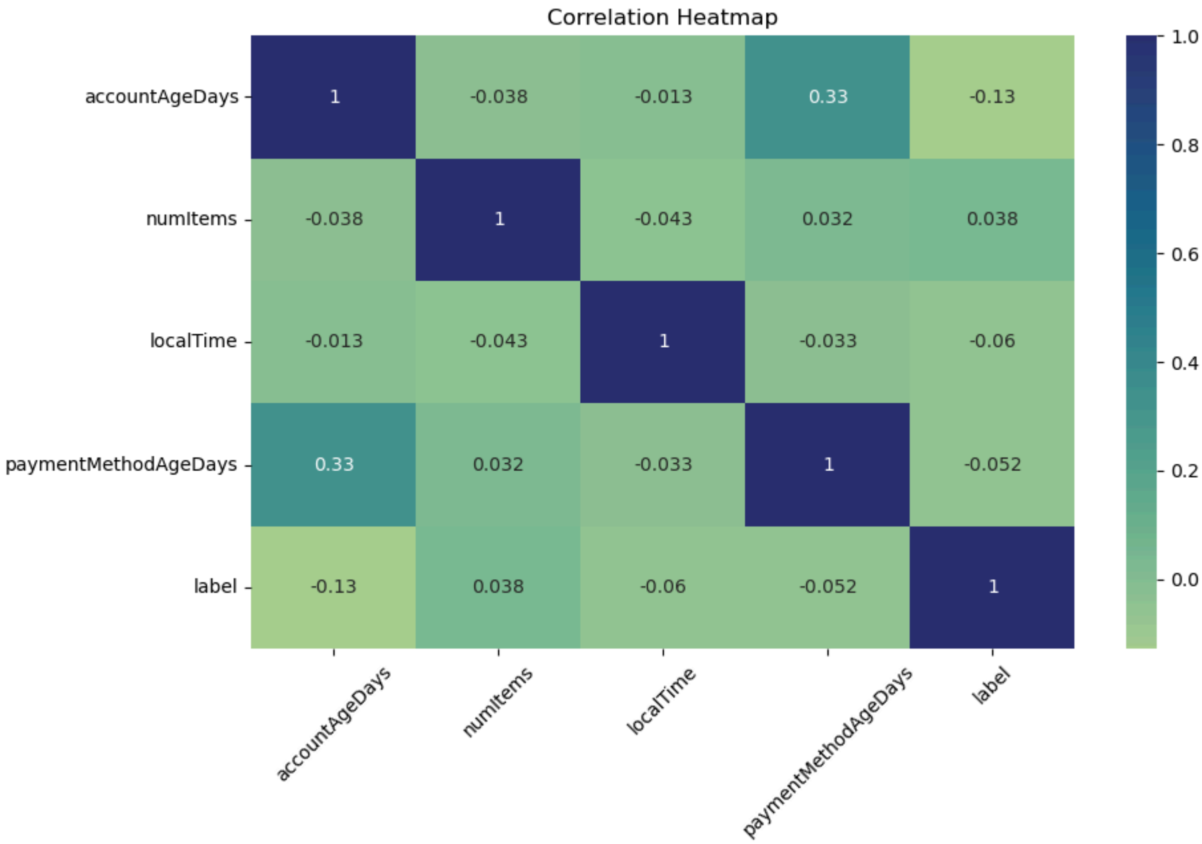
In [13]:
```python
# Univariant
# Shows Fraud vs. Non-Fraud sample counts.
sns.countplot(data=df_cleaned, x='label', palette ='Dark2', hue='label')
plt.title('Count Plot')
plt.show()
```



In [15]:
```
# univariant
# Shows: Frequency of different payment methods.
sns.histplot(x='paymentMethod', data=df_cleaned, stat="count")
plt.show()
```



In [23]:
```
# Select only numeric columns
# Shows: Correlation between numeric variables.

# here the values indicate corr between the variables +1 positive, -1 negati
numeric_df = df.select_dtypes(include='number')
correlation=numeric_df.corr()
plt.figure(figsize=(10,6))
sns.heatmap(correlation, annot=True, cmap='crest') # display the data values
plt.title('Correlation Heatmap')
plt.xticks(rotation=45)
plt.show()
```

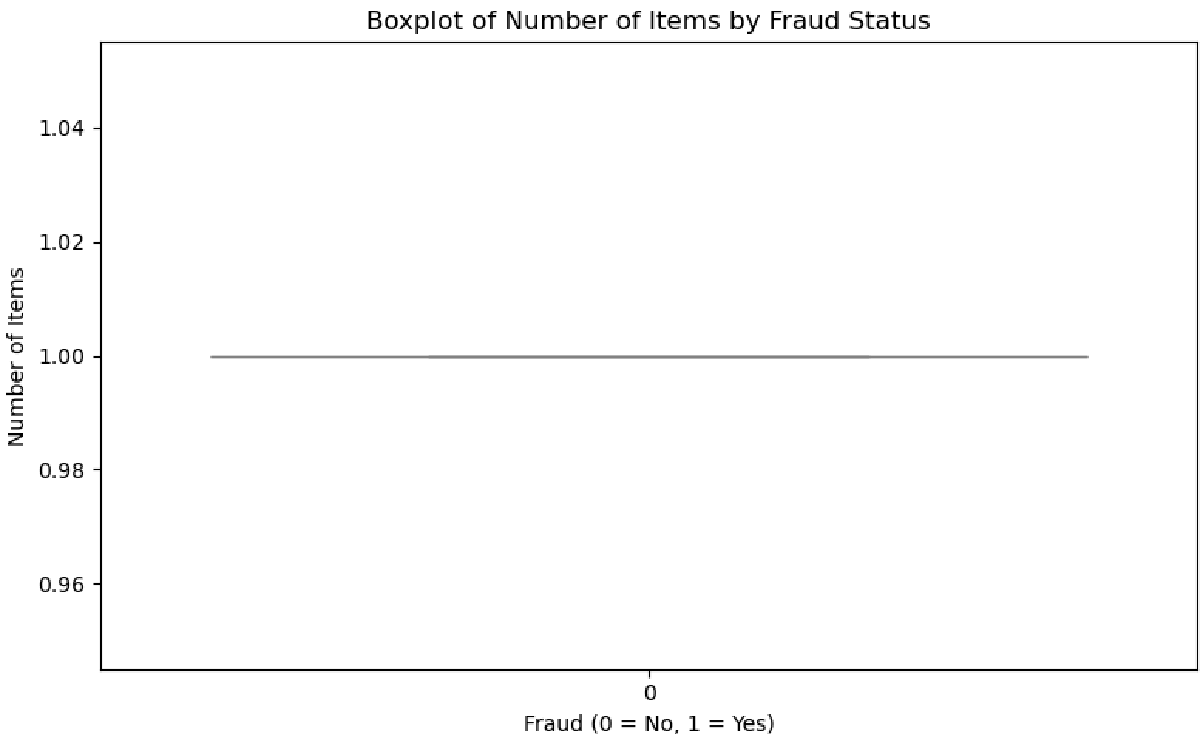


Graphical EDA

- Boxplot of Number of Items by Fraud Status
- Boxplot of Account Age by Fraud Status and Payment Method
- Violin Plot of Payment Method Age by Fraud Status

In [27]:
```
# box plot Shows: Number of items per transaction based on fraud status.
plt.figure(figsize=(8, 5))
sns.boxplot(data=df_cleaned, x='label', y='numItems', palette='coolwarm')

plt.title("Boxplot of Number of Items by Fraud Status")
plt.xlabel("Fraud (0 = No, 1 = Yes)")
plt.ylabel("Number of Items")
plt.tight_layout()
plt.show()
```

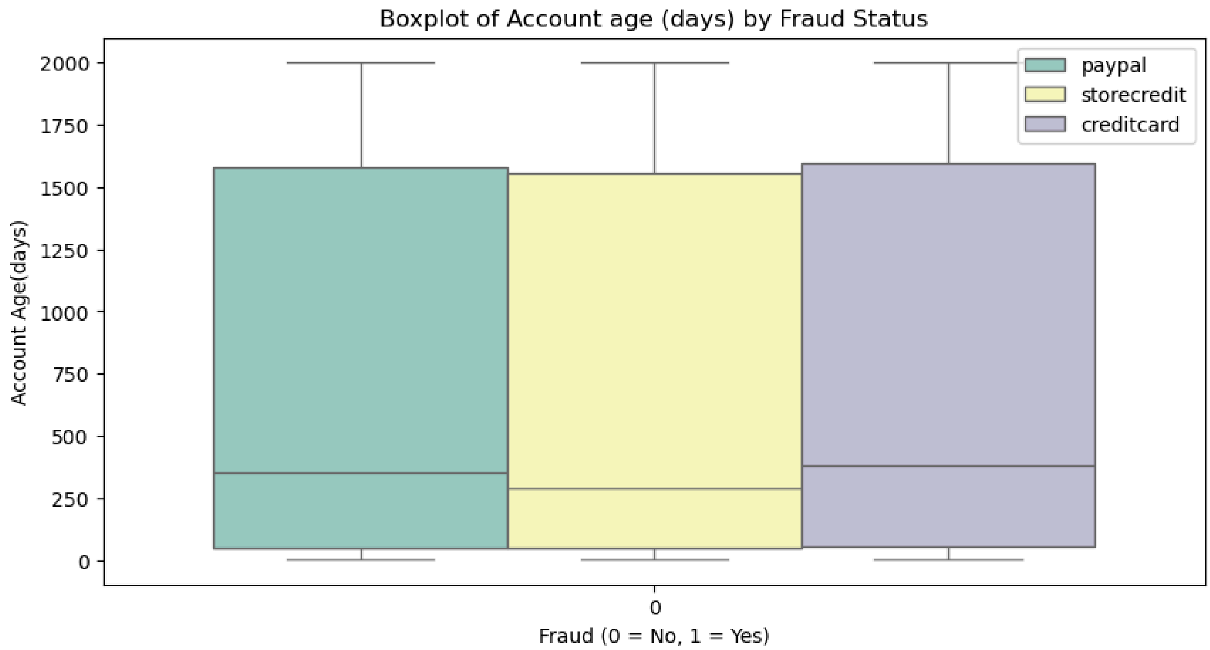
```
C:\Users\ATHARVA\AppData\Local\Temp\ipykernel_11844\1502099584.py:3: FutureW
arning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.boxplot(data=df_cleaned, x='label', y='numItems', palette='coolwarm')
<Figure size 800x500 with 0 Axes>
```



In [29]:
```
'''After removing outliers, the number of items in both fraud and non-fraud
So the boxplot shows a single line at 1.0 for both categories.
This means that, without the outliers, there's no variation in item count, a
this line is a median and represents the whole data'''
```

Out[29]:
```
"After removing outliers, the number of items in both fraud and non-fraud t
ransactions became almost the same — mostly 1.\nSo the boxplot shows a sing
le line at 1.0 for both categories.\nThis means that, without the outliers,
there's no variation in item count, and it’s not a useful feature for disti
nguishing fraud.\nthis line is a median and represents the whole data"
```

In [39]:
```
# Boxplot of amount by fraud status
plt.figure(figsize=(10, 5))
sns.boxplot(data=df_cleaned, x='label', y='accountAgeDays', hue='paymentMeth
plt.title("Boxplot of Account age (days) by Fraud Status")
plt.xlabel("Fraud (0 = No, 1 = Yes)")
plt.ylabel("Account Age(days)")
plt.legend(loc='upper right') # it was shown inside the graph for saving spa
plt.show()
```



In [35]:
```
''' This boxplot was supposed to show both fraud and non-fraud groups, but a
So this plot compares account age across different payment methods for only
```

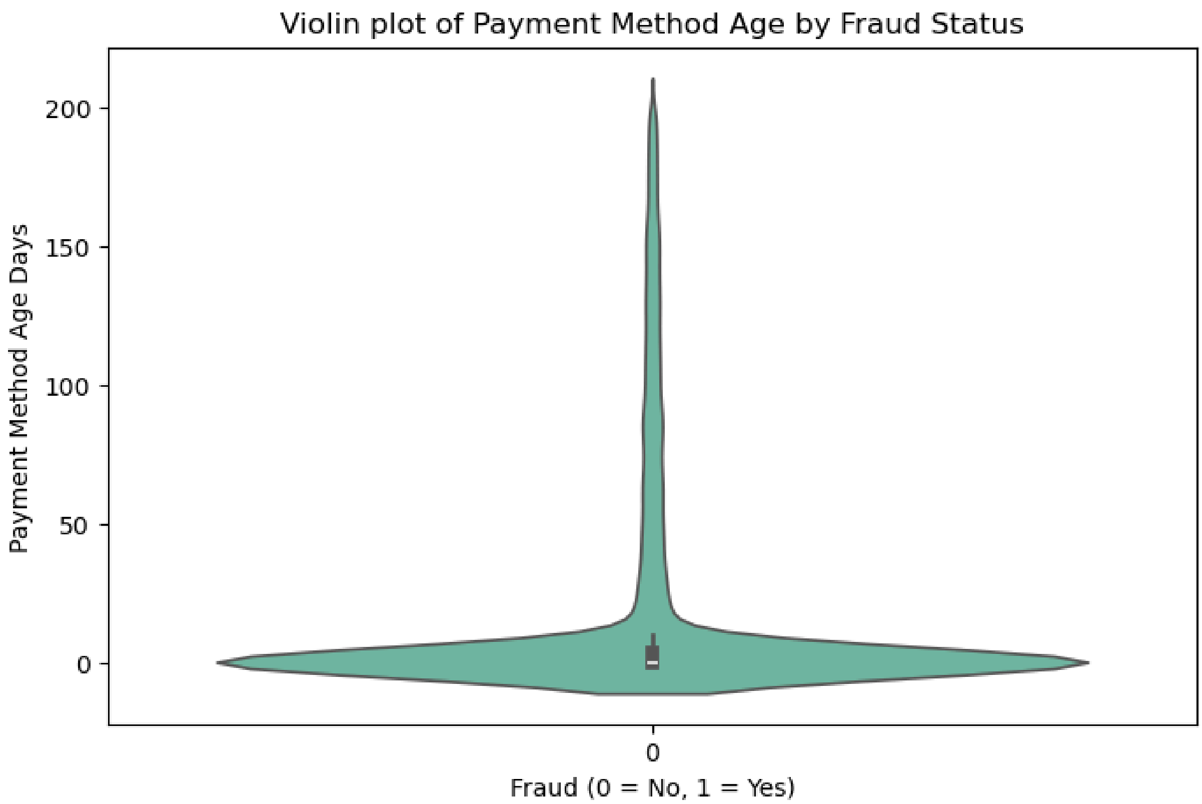
Out[35]:
```
' This boxplot was supposed to show both fraud and non-fraud groups, but af
ter removing outliers, only the non-fraud data remains visible.\nSo this pl
ot compares account age across different payment methods for only the non-f
raud transactions.'
```

In [1]:
```
'''This boxplot compares the account age (in days) for fraud and non-fraud t
It shows whether fraud is more common in newer or older accounts, and whethe
The boxes are split by color using the hue parameter, which represents diffe
After removing outliers the fraud cases are disappeared this
suggest that they often involve accounts with very low account age(newer acc
```

Out[1]:
```
'This boxplot compares the account age (in days) for fraud and non-fraud tr
ansactions, grouped by payment method.\nIt shows whether fraud is more comm
on in newer or older accounts, and whether that depends on which payment me
thod was used.\nThe boxes are split by color using the hue parameter, which
represents different payment methods like PayPal, Store Credit, and Credit
Card.\nAfter removing outliers the fraud cases are disappeared this \nsugge
st that they often involve accounts with very low account age(newer account
s), which were treated as outliers."'
```

In [43]:
```
# voilin plot

plt.figure(figsize=(8, 5))
sns.violinplot(data=df_cleaned, x='label', y='paymentMethodAgeDays', palette
plt.title("Violin plot of Payment Method Age by Fraud Status")
plt.xlabel("Fraud (0 = No, 1 = Yes)")
plt.ylabel("Payment Method Age Days")
plt.show()
```

```
C:\Users\ATHARVA\AppData\Local\Temp\ipykernel_11844\493599876.py:4: FutureWa
rning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.violinplot(data=df_cleaned, x='label', y='paymentMethodAgeDays', palet
te='Set2')
```



In [45]:
```
''' This violin plot compares how long payment methods had been used in frau
We see that most values are clustered near the bottom — around 0 to 50 days.
This means that fraud tends to happen more with newer payment methods.
The fraud side (label = 1) is almost flat or missing, showing very few frauc
```

Out[45]:
```
' This violin plot compares how long payment methods had been used in fraud
vs. non-fraud transactions.\nWe see that most values are clustered near the
bottom — around 0 to 50 days.\nThis means that fraud tends to happen more w
ith newer payment methods.\nThe fraud side (label = 1) is almost flat or mi
ssing, showing very few fraud cases left after cleaning.'
```

5. Encoding Categorical Variables

- Label Encoding: paymentMethod

In [47]:
```
# label Encoding Converts categorical payment method to numeric values.
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df_cleaned['paymentMethod'] = le.fit_transform(df_cleaned.paymentMethod.valu
df_cleaned.head()
```

Out[47]:

	accountAgeDays	numItems	localTime	paymentMethod	paymentMethodA
0	29	1	4.745402	1	28.
1	725	1	4.742303	2	0.
2	845	1	4.921318	0	0.
3	503	1	4.886641	0	0.
4	2000	1	5.040929	0	0.

In [49]:
```
df_cleaned.paymentMethod.value_counts()
```

Out[49]:
```
paymentMethod
0    20126
1     6750
2     1400
Name: count, dtype: int64
```

In [51]:
```
# One-hot Encoding create a column for each category

one_hot = pd.get_dummies(df_cleaned['paymentMethod'])
one_hot
```

Out[51]:

	0	1	2
0	False	True	False
1	False	False	True
2	True	False	False
3	True	False	False
4	True	False	False
...	...	...	...
39212	True	False	False
39213	True	False	False
39216	True	False	False
39218	True	False	False
39220	False	True	False

28276 rows × 3 columns

In [53]:
```
df_cleaned.head(10)
```

Out[53]:

	accountAgeDays	numItems	localTime	paymentMethod	paymentMethodA
0	29	1	4.745402	1	28.
1	725	1	4.742303	2	0.
2	845	1	4.921318	0	0.
3	503	1	4.886641	0	0.
4	2000	1	5.040929	0	0.
5	119	1	4.962055	1	0.
6	2000	1	4.921349	1	0.
7	371	1	4.876771	0	0.
8	2000	1	4.748314	0	0.
9	4	1	4.461622	0	0.

In [31]:

```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\atharva\anaconda3\li
b\site-packages (1.6.1)
Collecting scikit-learn
  Using cached scikit_learn-1.7.1-cp312-cp312-win_amd64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.22.0 in c:\users\atharva\anaconda3\l
ib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.8.0 in c:\users\atharva\anaconda3\li
b\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\atharva\anaconda3\l
ib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\atharva\anac
onda3\lib\site-packages (from scikit-learn) (3.5.0)
Using cached scikit_learn-1.7.1-cp312-cp312-win_amd64.whl (8.7 MB)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.6.1
    Uninstalling scikit-learn-1.6.1:
      Successfully uninstalled scikit-learn-1.6.1
Successfully installed scikit-learn-1.7.1
Note: you may need to restart the kernel to use updated packages.
```

```
  WARNING: Failed to remove contents in a temporary directory 'C:\Users\ATHA
RVA\anaconda3\Lib\site-packages\~=learn'.
  You can safely remove it manually.
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
sklearn-compat 0.1.3 requires scikit-learn<1.7,>=1.2, but you have scikit-le
arn 1.7.1 which is incompatible.
```

6. Feature Scaling

- Standardization
- Normalization

In [32]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

In [55]:
```
df_cleaned.head()
df_cleaned.describe().round(2)
```

Out[55]:

	accountAgeDays	numItems	localTime	paymentMethod	paymentMeth
count	28276.00	28276.0	28276.00	28276.00	
mean	754.78	1.0	4.85	0.34	
std	784.98	0.0	0.16	0.57	
min	2.00	1.0	4.46	0.00	
25%	51.00	1.0	4.75	0.00	
50%	365.00	1.0	4.89	0.00	
75%	1589.00	1.0	4.96	1.00	
max	2000.00	1.0	5.04	2.00	

In [57]:
```
# normalization

new_df = pd.DataFrame(df_cleaned,columns = ['paymentMethod', 'paymentMethodA
new_df.head(5)
```

Out[57]:

	paymentMethod	paymentMethodAgeDays
0	1	28.204861
1	2	0.000000
2	0	0.000000
3	0	0.000000
4	0	0.000000

In [35]:
```
scalar = MinMaxScaler() #instantiating the minmaxscalar() function
normalized_df = scalar.fit_transform(new_df)
print(normalized_df)
```

```
[[5.00000000e-01 1.41053888e-02]
 [1.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00]
 ...
 [0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.18066081e-01]
 [5.00000000e-01 3.47295058e-07]]
```

In [36]:
```
# standardization
scalar = StandardScaler() #instantiating the standardscalar() function
standardized_df = scalar.fit_transform(new_df)
print(standardized_df)
```

```
[[ 1.17536144 -0.33303221]
 [ 2.94227791 -0.43249725]
 [-0.59155504 -0.43249725]
 ...
 [-0.59155504 -0.43249725]
 [-0.59155504  0.40005321]
 [ 1.17536144 -0.4324948 ]]
```

In [37]:
```
!pip install -U scikit-learn imbalanced-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\atharva\anaconda3\li
b\site-packages (1.7.1)
Requirement already satisfied: imbalanced-learn in c:\users\atharva\anaconda
3\lib\site-packages (0.13.0)
Requirement already satisfied: numpy>=1.22.0 in c:\users\atharva\anaconda3\l
ib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.8.0 in c:\users\atharva\anaconda3\li
b\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\atharva\anaconda3\l
ib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\atharva\anac
onda3\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: sklearn-compat<1,>=0.1 in c:\users\atharva\an
aconda3\lib\site-packages (from imbalanced-learn) (0.1.3)
Collecting scikit-learn
  Using cached scikit_learn-1.6.1-cp312-cp312-win_amd64.whl.metadata (15 kB)
Using cached scikit_learn-1.6.1-cp312-cp312-win_amd64.whl (11.1 MB)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.7.1
    Uninstalling scikit-learn-1.7.1:
      Successfully uninstalled scikit-learn-1.7.1
Successfully installed scikit-learn-1.6.1
```

7. Train-Test Split

- 80-20 split
- Stratified sampling to handle class imbalance

In [61]:
```
# Train-Test Split
from sklearn.model_selection import train_test_split
X = df_cleaned.drop('label', axis=1)
```

```
y = df_cleaned['label']

# X_train, y_train: 80% of the data for training
# X_test, y_test: 20% of the data for testing
# test_size=0.2: Keep 20% of data for testing.
# stratify=y: Ensures the same ratio of fraud and non-fraud in both train ar
# random_state=42: Fixes the random split so you get the same result every t

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, str
```

In [39]:
```
#print(X_train.dtypes)

import sklearn
import imblearn

print("scikit-learn version:", sklearn.__version__)
print("imbalanced-learn version:", imblearn.__version__)
```

```
scikit-learn version: 1.6.1
imbalanced-learn version: 0.13.0
```

8. Handling Class Imbalance

- Undersampling (RandomUnderSampler)
- Oversampling (SMOTE)

In [40]:
```
# undersampling - majority class(non fraud) is reduced to minority class(fra
from imblearn.under_sampling import RandomUnderSampler

undersample = RandomUnderSampler(random_state=42)
X_under, y_under = undersample.fit_resample(X_train, y_train)
# fit_resample() does two things: Fit the resampling logic (e.g., how many s
# Resample the dataset accordingly and return the balanced X and y

print("Before:", np.bincount(y_train)) # counts the number of occurrences of
print("After:", np.bincount(y_under))# y_under is the resampled version of y
```

```
Before: [30928   448]
After: [448 448]
```

In [69]:
```
'''We want to reduce the majority class, we need to count labels, not featur
Features (X): These are the input variables used to make a prediction.
Example: account age, number of items, payment method, etc.

Label (y): This is the output variable (what we want to predict).
Example: whether a transaction is fraud or not (0 or 1).'''
```

Out[69]:
```
'We want to reduce the majority class, we need to count labels, not feature
s\nFeatures (X): These are the input variables used to make a prediction.\n
Example: account age, number of items, payment method, etc.\n\nLabel (y): T
his is the output variable (what we want to predict).\nExample: whether a t
ransaction is fraud or not (0 or 1).'
```

In [41]:
```
# Oversampling - Increases the number of samples in the minority class (frau
from imblearn.over_sampling import SMOTE # Synthetic Minority Over-sampling
```

```
# It creates fake but realistic examples of the minority class.
# Instead of copying the same rows, SMOTE looks at a point's nearest neighbo
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_train, y_train)

print("Before:", np.bincount(y_train))
print("After:", np.bincount(y_smote))
```

```
Before: [30928   448]
After: [30928 30928]
```

In [62]:

```
from sklearn.metrics import classification_report, confusion_matrix
# classifies data accurracy according to the models used


# from sklearn.datasets import make_regression # linear regression
```

In [71]:

```
'''We scale data when using distance-based or gradient-based models like KNN
Tree-based models like Random Forest don’t need scaling because they split b
```

Out[71]:

```
'We scale data when using distance-based or gradient-based models like KNN,
SVM, or logistic regression.\nTree-based models like Random Forest don’t ne
ed scaling because they split based on thresholds, not distances.'
```

Model Building & Evaluation

Model 1: Random Forest Classifier

- Training
- Confusion Matrix
- Performance Metrics

In [57]:

```
# Random Forest

from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier() # Train Random Forest model
model_rf.fit(X_smote, y_smote)  # or use X_under, y_under

# Predict
y_pred_rf = model_rf.predict(X_test)

# Classification Report
from sklearn.metrics import classification_report
print("Random Forest:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      7733
           1       1.00      1.00      1.00       112

    accuracy                           1.00      7845
   macro avg       1.00      1.00      1.00      7845
weighted avg       1.00      1.00      1.00      7845
```

Model 2: Support Vector Machine (SVM)

- Training
- Confusion Matrix
- Performance Metrics

In [60]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Scale training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_smote)  # or X_under
X_test_scaled = scaler.transform(X_test)

# Train SVM model
model_svm = SVC()
model_svm.fit(X_train_scaled, y_smote)  # or y_under

# Predict
y_pred_svm = model_svm.predict(X_test_scaled)

# Classification Report
from sklearn.metrics import classification_report
print("SVM:\n", classification_report(y_test, y_pred_svm))
```

```
SVM:
               precision    recall  f1-score   support

           0       1.00      0.86      0.93      7733
           1       0.09      1.00      0.17       112

    accuracy                           0.86      7845
   macro avg       0.55      0.93      0.55      7845
weighted avg       0.99      0.86      0.92      7845
```

Model 3: K-Nearest Neighbors (KNN)

- Training
- Confusion Matrix
- Performance Metrics

In [62]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

# Scale training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_smote)  # or X_under
X_test_scaled = scaler.transform(X_test)

# Train KNN model
model_knn = KNeighborsClassifier()
model_knn.fit(X_train_scaled, y_smote)  # or y_under

# Predict
y_pred_knn = model_knn.predict(X_test_scaled)

# Classification Report
from sklearn.metrics import classification_report
print("KNN:\n", classification_report(y_test, y_pred_knn))



# 0 is class 0 (non-fraud)

# 1 is class 1 (fraud)

# accuracy = overall correct predictions

# macro avg = average across classes (treat all classes equally)

# weighted avg = average considering how many samples are in each class
```

```
KNN:
               precision    recall  f1-score   support

           0       1.00      0.99      1.00      7733
           1       0.65      0.95      0.77       112

    accuracy                           0.99      7845
   macro avg       0.83      0.97      0.88      7845
weighted avg       0.99      0.99      0.99      7845
```

Precision: How precise your fraud predictions are

Recall: How many real frauds you caught

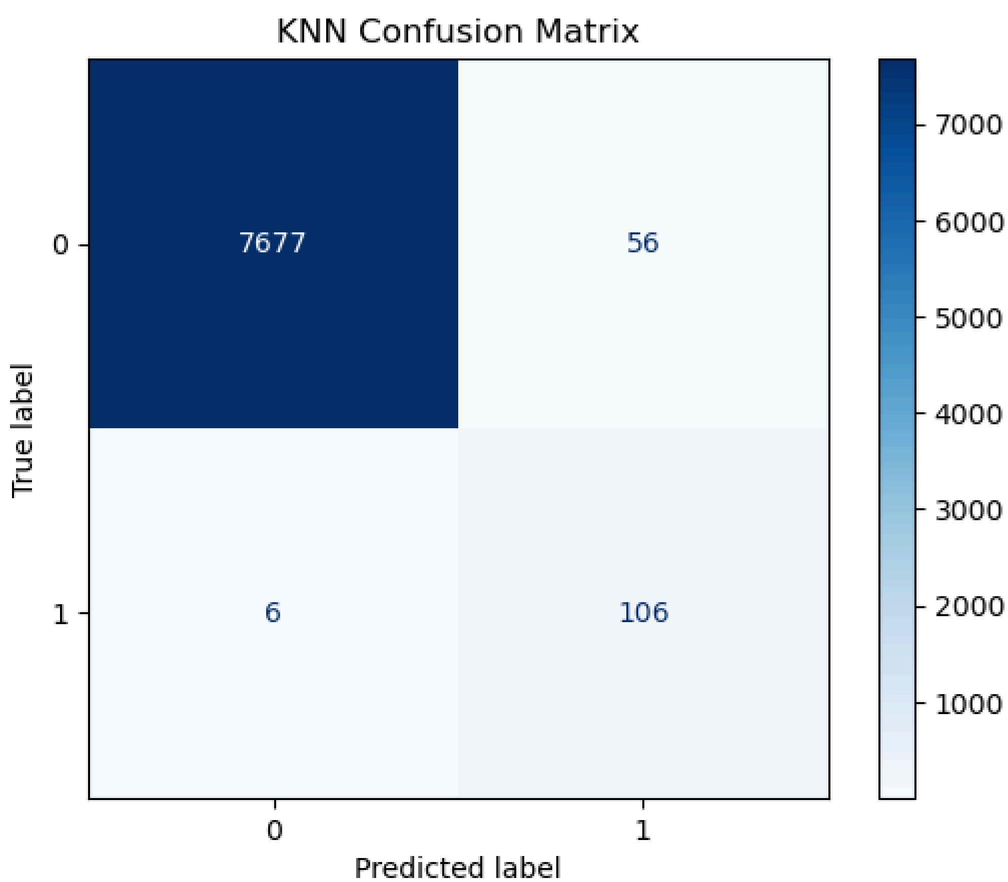
F1-score: Balance between precision and recall

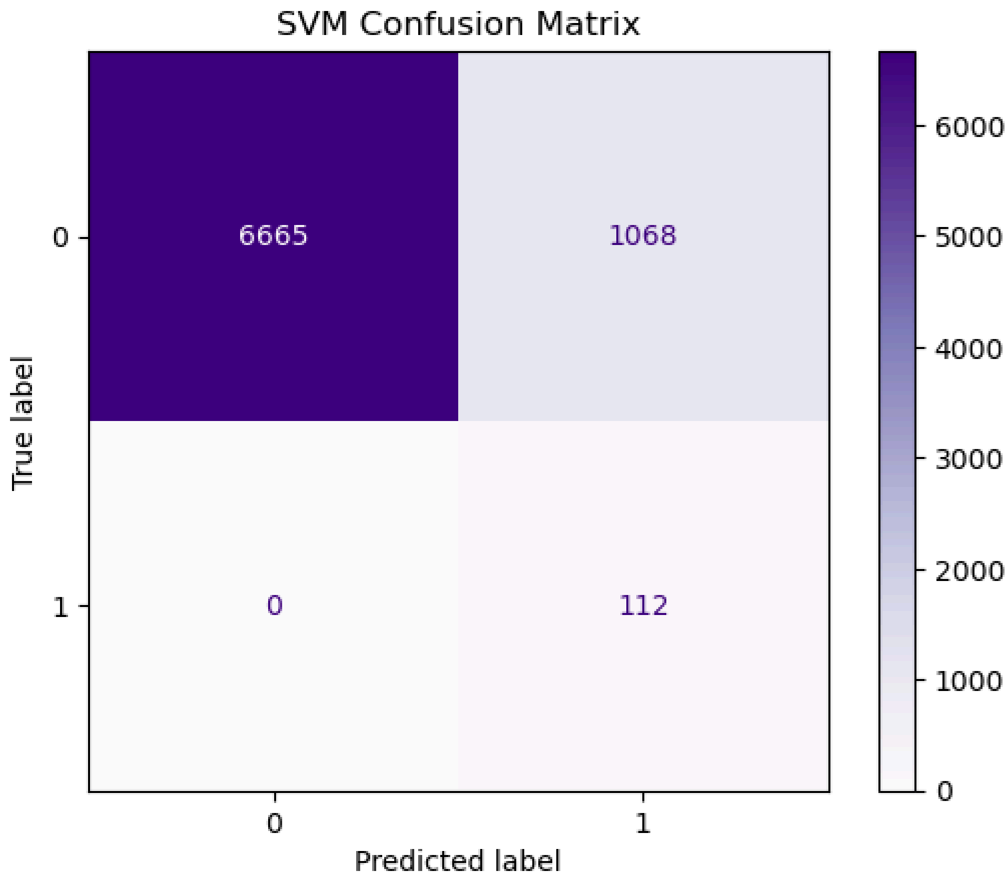
Accuracy: Overall performance

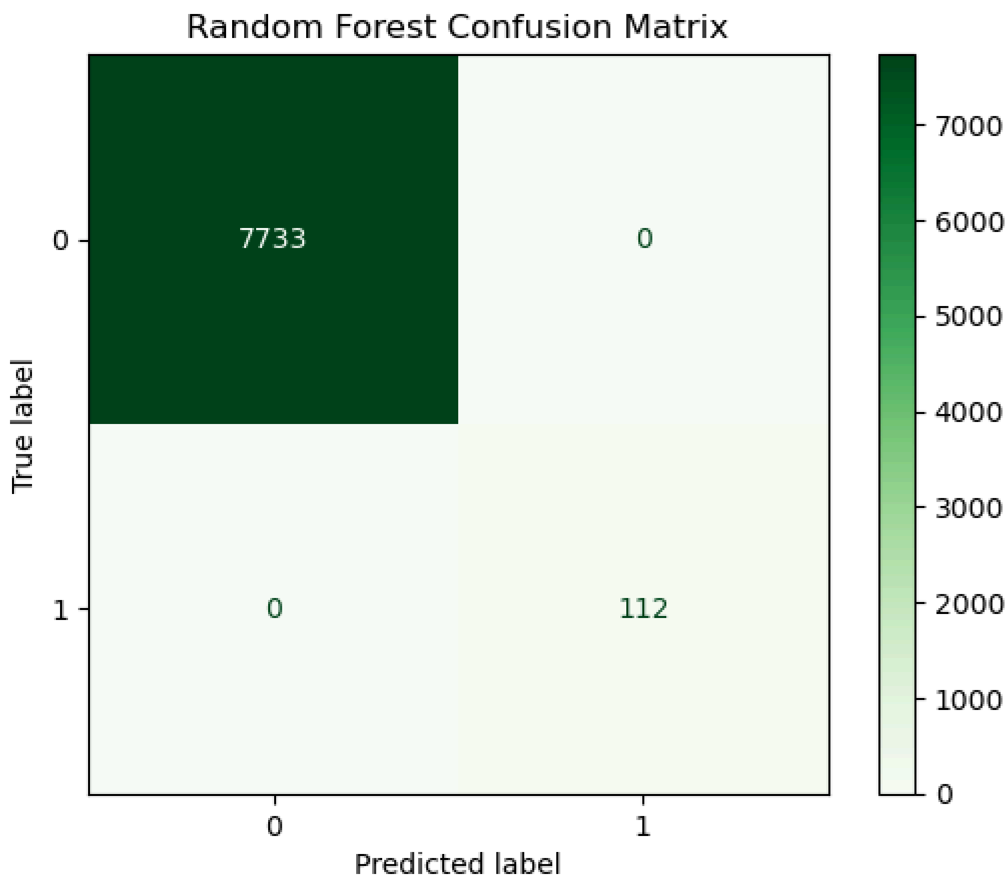
Plotting confusion matrix

In [64]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```python
# For KNN
cm_knn = confusion_matrix(y_test, y_pred_knn)
disp_knn = ConfusionMatrixDisplay(confusion_matrix=cm_knn)
disp_knn.plot(cmap='Blues')
plt.title("KNN Confusion Matrix")
plt.show()

# For SVM
cm_svm = confusion_matrix(y_test, y_pred_svm)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm)
disp_svm.plot(cmap='Purples')
plt.title("SVM Confusion Matrix")
plt.show()

# For Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf)
disp_rf.plot(cmap='Greens')
plt.title("Random Forest Confusion Matrix")
plt.show()

# the graph below tells that how many values are true positive and true nega
```




SVM Confusion Matrix
6665
1068
0
112
True label
Predicted label
0
1
0
1
6000
5000
4000
3000
2000
1000
0


Random Forest Confusion Matrix
7733
0
0
112
True label
Predicted label
0
1
0
1
7000
6000
5000
4000
3000
2000
1000
0

In [79]:
```
# TN    True Negatives (non-fraud correctly predicted)
# TP    True Positives (fraud correctly predicted)
# FP    False Positives (non-fraud wrongly marked as fraud)
# FN    False Negatives (fraud missed as non-fraud)


#                  Predicted No (0)     Predicted Yes (1)

# Actual No (0)   True Negative (TN)    False Positive (FP)
# Actual Yes (1)  False Negative (FN)  True Positive (TP)


# Based on the confusion matrices, KNN is the most balanced model — it detec
# SVM over-predicts fraud, and Random Forest misses all frauds."
```

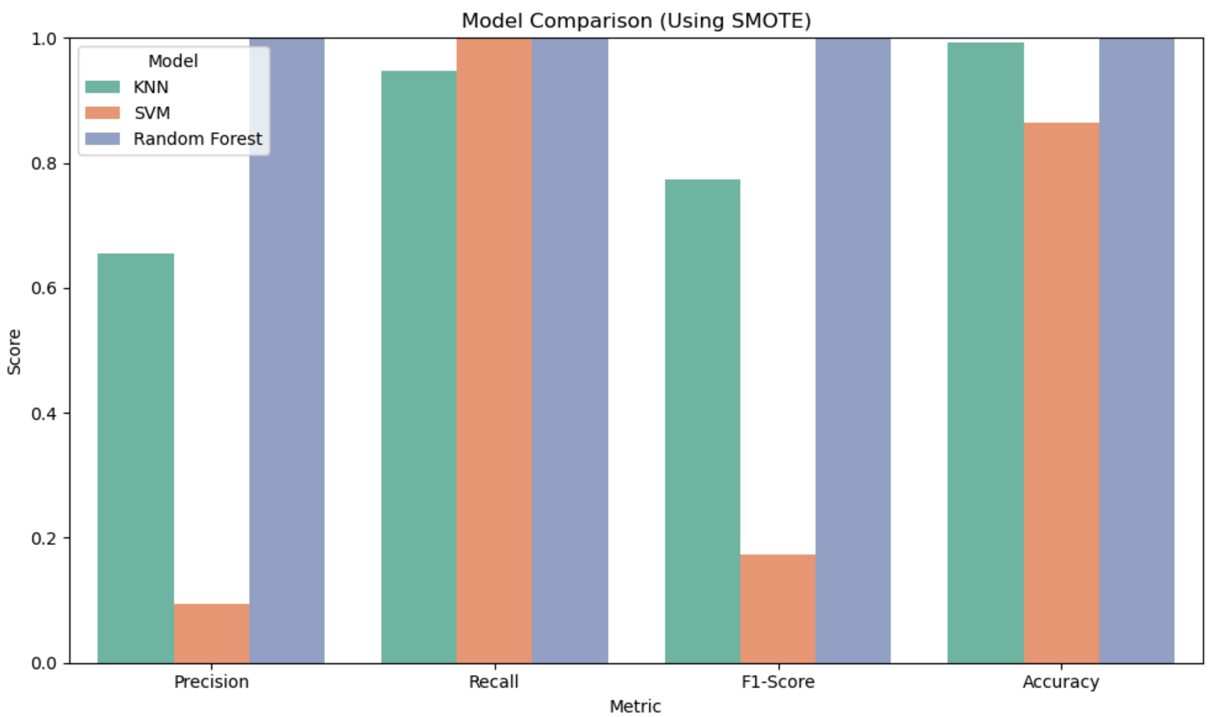
Model Graphs

Model Comparison & Metrics Visualization

- Accuracy
- Precision
- Recall
- F1 Score

In [86]:
```python
# Metric---> Type of performance measurement (Precision, Recall, Accuracy, e
# Score----> The actual result or value for that metric (e.g., 0.89 = 89%)

from sklearn.metrics import precision_score, recall_score, f1_score, accurac
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Metrics for each model (using predictions on original y_test)
metrics = {
    'KNN': {
        'Precision': precision_score(y_test, y_pred_knn),
        'Recall': recall_score(y_test, y_pred_knn),
        'F1-Score': f1_score(y_test, y_pred_knn),
        'Accuracy': accuracy_score(y_test, y_pred_knn)
    },
    'SVM': {
        'Precision': precision_score(y_test, y_pred_svm),
        'Recall': recall_score(y_test, y_pred_svm),
        'F1-Score': f1_score(y_test, y_pred_svm),
        'Accuracy': accuracy_score(y_test, y_pred_svm)
    },
    'Random Forest': {
        'Precision': precision_score(y_test, y_pred_rf),
        'Recall': recall_score(y_test, y_pred_rf),
        'F1-Score': f1_score(y_test, y_pred_rf),
        'Accuracy': accuracy_score(y_test, y_pred_rf)
    }
}
```

```python
# Convert to DataFrame
# columns are models (KNN, SVM, etc.) ; rows are metric names
# .T transposes the DataFrame
# After transpose, the row labels (KNN, SVM, etc.) are in the index reset_ir
# Melts (unpivots) the DataFrame from wide to long format ; Keeps the 'index
# unpivot column metrics to row metrics

df_metrics = pd.DataFrame(metrics).T.reset_index().melt(id_vars='index')
df_metrics.columns = ['Model', 'Metric', 'Score'] # Renames the columns from

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=df_metrics, x='Metric', y='Score', hue='Model', palette='Se
plt.title("Model Comparison (Using SMOTE)")
plt.ylim(0, 1)
plt.ylabel("Score")
plt.legend(title='Model')
plt.tight_layout()
plt.show()
```



In [88]:
```python
# This helps you visually evaluate which model performs best on which metric
# For fraud detection, we focus more on F1-Score and Recall because it's imp
# So, the model with the highest bars in those metrics would be the most eff

# Accuracy--> Overall correct predictions
# Precision--> Of all fraud predictions, how many were right
# Recall--> Of all actual frauds, how many were caught
# F1-Score--> Overall fraud detection effectiveness
```

In [ ]: