1. Import Libraries

In [1]:
```
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.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

import joblib
```

2. Load Dataset

In [2]:
```
df = pd.read_csv('creditcard.csv')
```

3. Explore Dataset

In [3]:
```
df.shape
```

Out[3]:
```
(284807, 31)
```

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.describe()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84807(
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.21348
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.19435
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.32167
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.08629
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.23580
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.27345
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.00072

8 rows × 31 columns

In [6]:

```
df.isnull().sum()
```

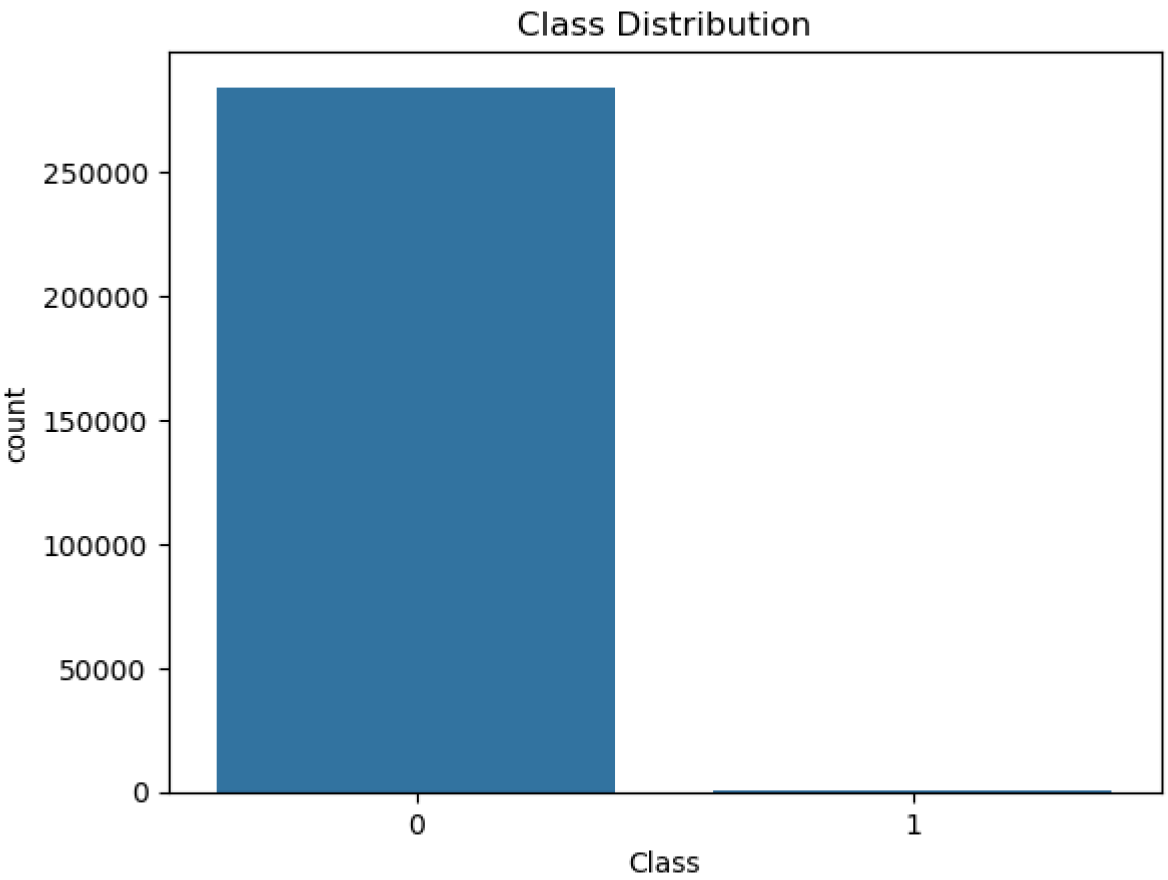
Out[6]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [7]:
```
df['Class'].value_counts()
```

Out[7]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

4. Visualize Calss Distribution

In [8]:
```
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.show()
```



5. As Time column is not useful so drop it

In [9]:
```
df = df.drop(['Time'], axis = 1)
df.head()
```

Out[9]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.1
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.1
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.9
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.1
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.1

5 rows × 30 columns

6. Scaled `Amount` to normalize its range and improve model performance

In [10]:

```
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
```

In [11]:

```
df.head()
```

Out[11]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.1
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.1
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.9
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.1
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.1

5 rows × 30 columns

7. Split features and target

In [12]:
```
X = df.drop(['Class'], axis = 1)
y = df['Class']
```

In [13]:
```
X.head()
```

Out[13]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.251412	-0.018307	0.2
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.069083	-0.225775	-0.6
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.524980	0.247998	0.7
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.208038	-0.108300	0.0
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.408542	-0.009431	0.7

5 rows × 29 columns

In [14]:
```
print('Sr','Class')
y.head()
```

```
Sr Class
```

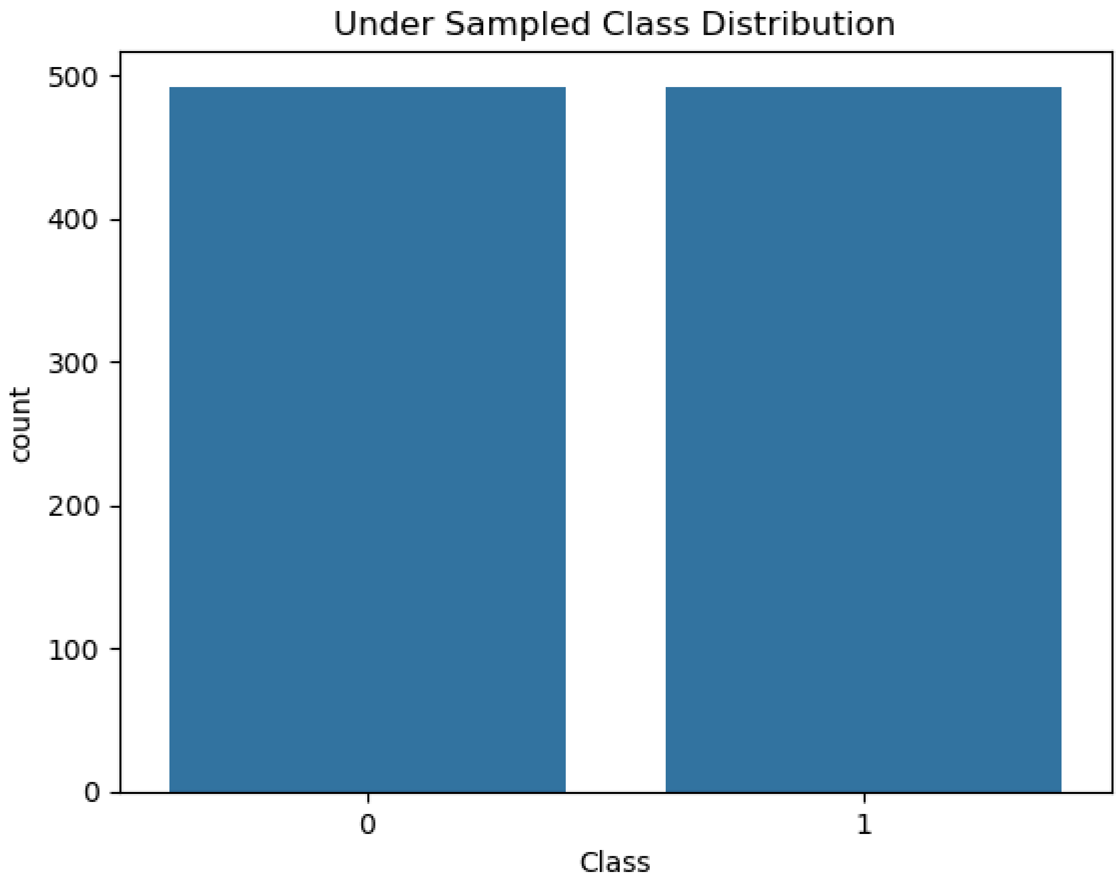
Out[14]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

8. Under Sampling as our dataset is imbalance (One class `Fraudlent` that is `1` is significantly less than other `Normal` that is `0`)

In [15]:
```
rus = RandomUnderSampler(random_state=42)
X_under, y_under = rus.fit_resample(X, y)
print("Under Sampled Data Shape:", X_under.shape)
```

```
sns.countplot(x=y_under)
plt.title("Under Sampled Class Distribution")
plt.show()
```

```
Under Sampled Data Shape: (984, 29)
```



In [16]:
```
y_under.value_counts()
```

Out[16]:
```
Class
0    492
1    492
Name: count, dtype: int64
```

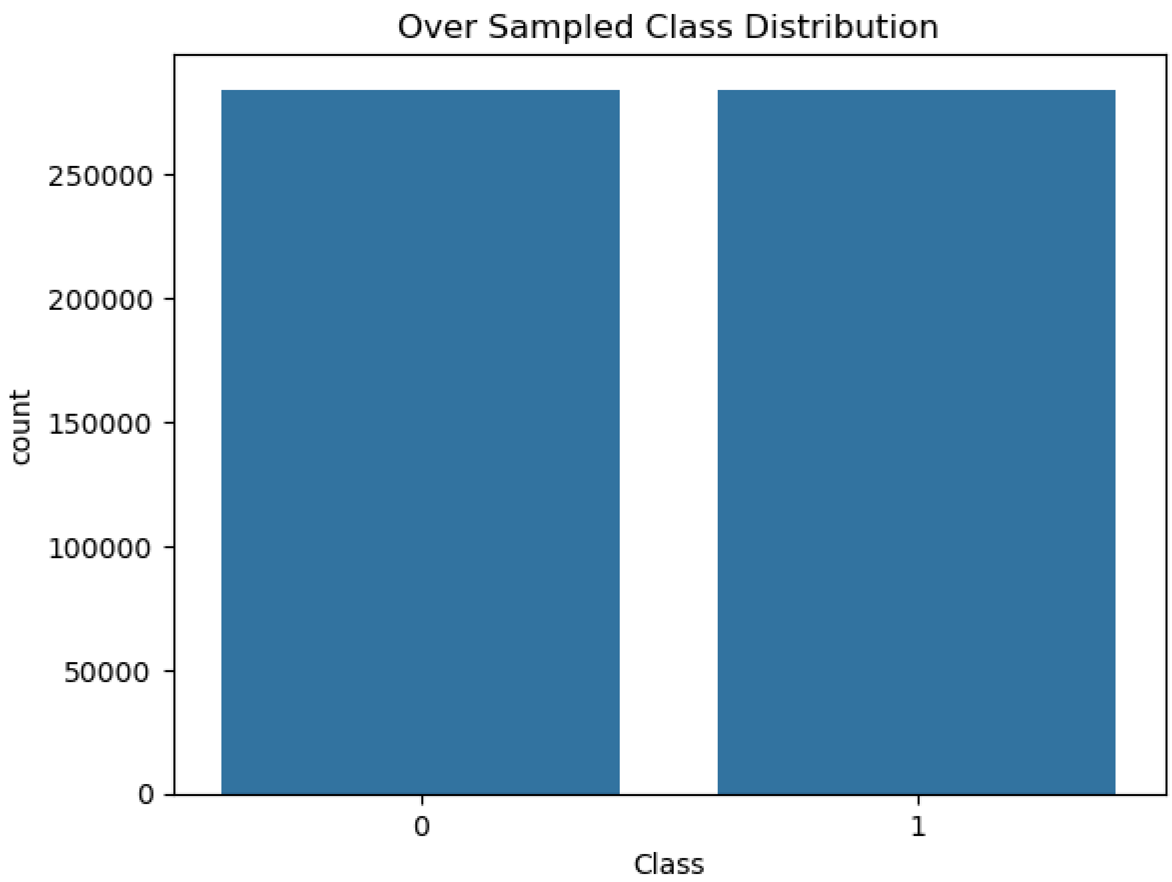
9. As in under sampling we have lost many important data points, so it is not good to do undersampling when we have huge

difference in numbers of class.

Here we will use Over sampling whichwill generate the data points for the minority class. We use SMUT for this

In [17]:
```
smote = SMOTE(random_state=42)
X_over, y_over = smote.fit_resample(X, y)
print("Over Sampled Data Shape:", X_over.shape)
sns.countplot(x=y_over)
plt.title("Over Sampled Class Distribution")
plt.show()
```

```
Over Sampled Data Shape: (568630, 29)
```



In [18]:
```
y_over.value_counts()
```

Out[18]:
```
Class
0    284315
1    284315
Name: count, dtype: int64
```

10. Train-Test Split (on oversampled data)

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size=0.2, random_state=42, stratify=y_over)
```

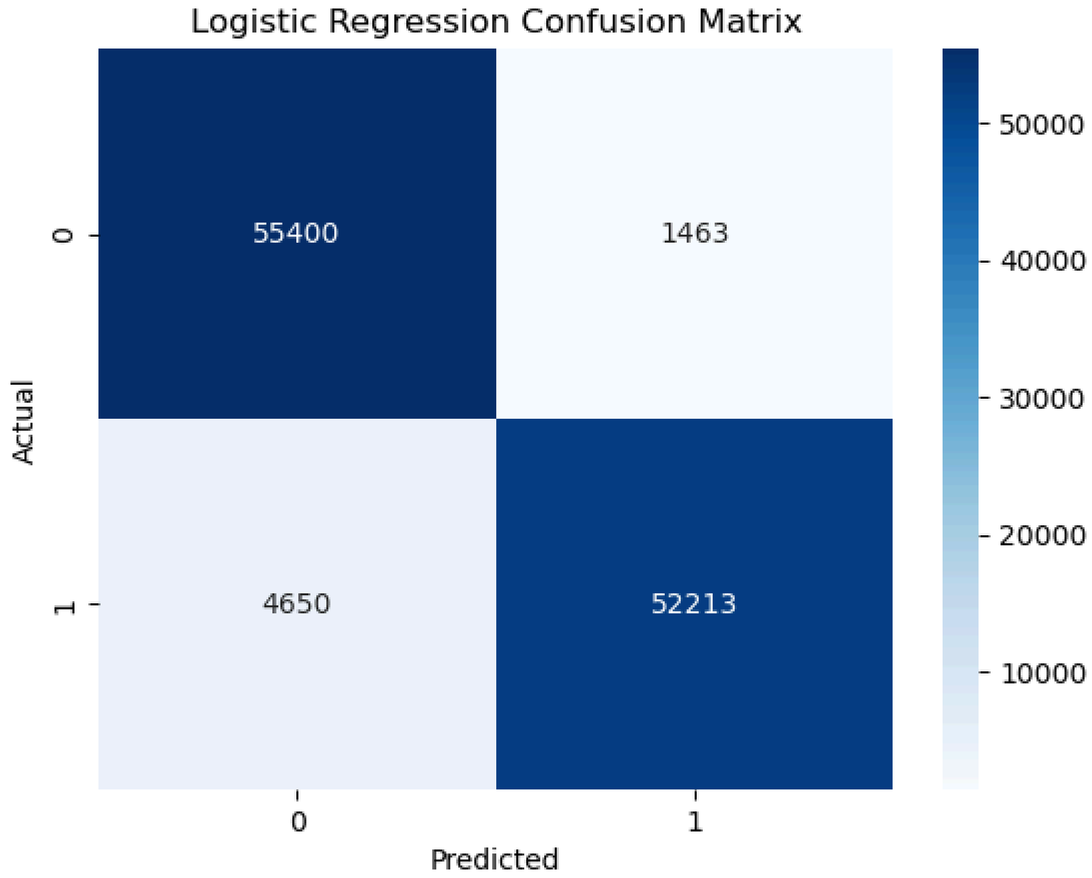
11. Define Function to Evaluate Models

In [20]:
```
def evaluate_model(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\n{name} Results:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    return accuracy_score(y_test, y_pred)
```

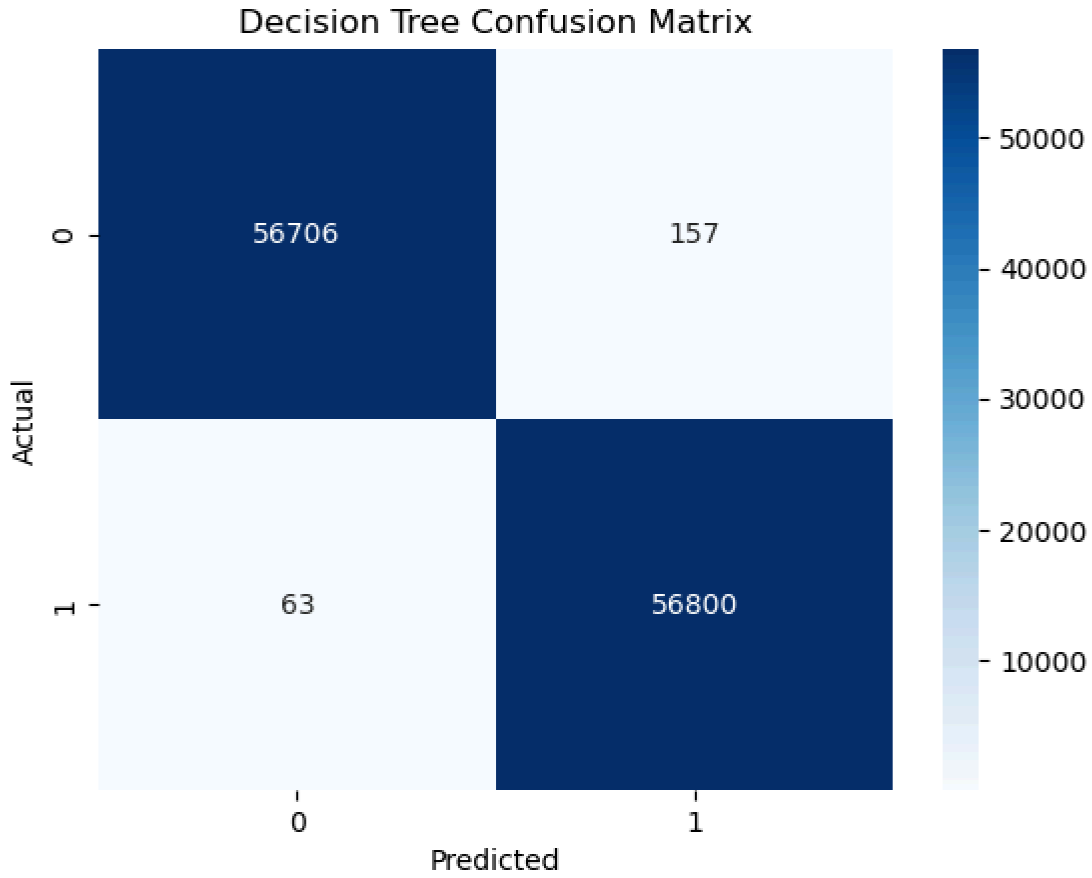
12. Train and Evaluate Models

In [21]:
```
accuracy_scores = {}

# Logistic Regression
lr = LogisticRegression(max_iter=1000)
accuracy_scores['Logistic Regression'] = evaluate_model(lr, "Logistic Regression")

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
accuracy_scores['Decision Tree'] = evaluate_model(dt, "Decision Tree")

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
accuracy_scores['Random Forest'] = evaluate_model(rf, "Random Forest")
```

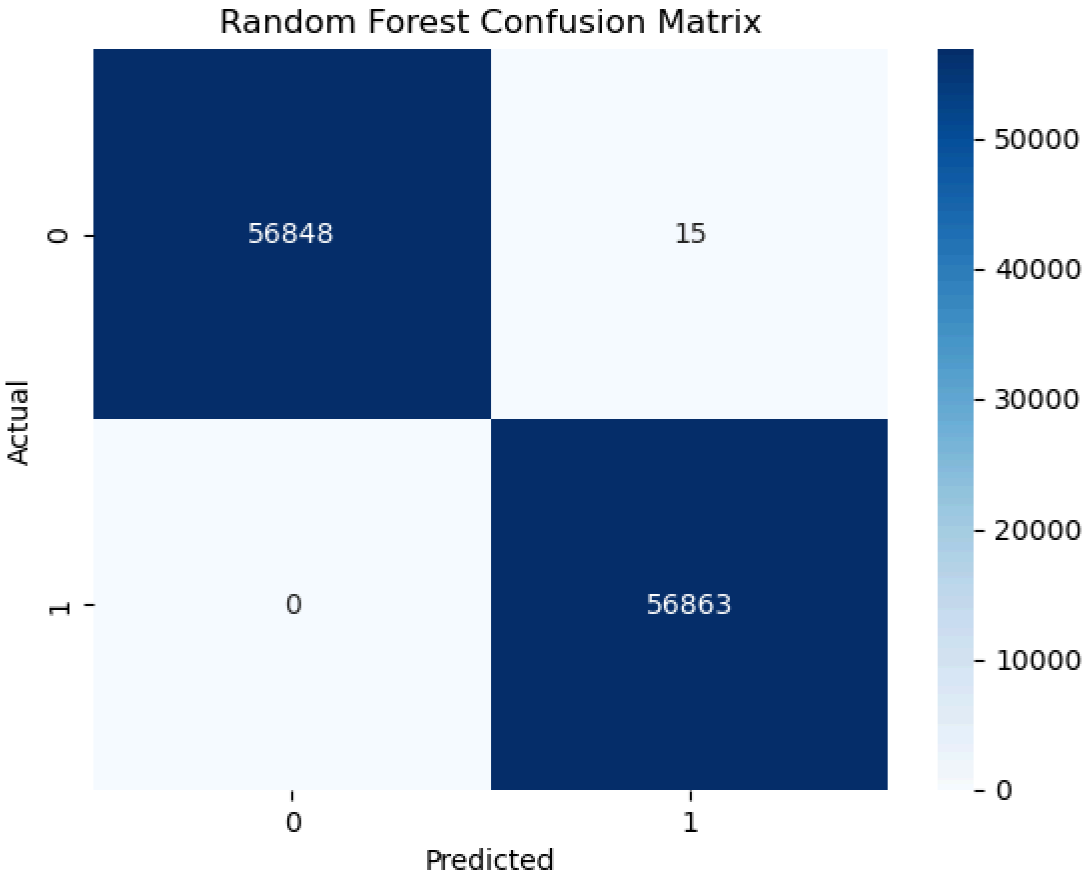
```
Logistic Regression Results:
Accuracy: 0.9462479995779329
Precision: 0.9727438706311946
Recall: 0.9182245045108418
F1 Score: 0.9446982512959227
```




```
Decision Tree Results:
Accuracy: 0.9980655259131597
Precision: 0.9972435345962744
Recall: 0.9988920739320823
F1 Score: 0.9980671235283781
```

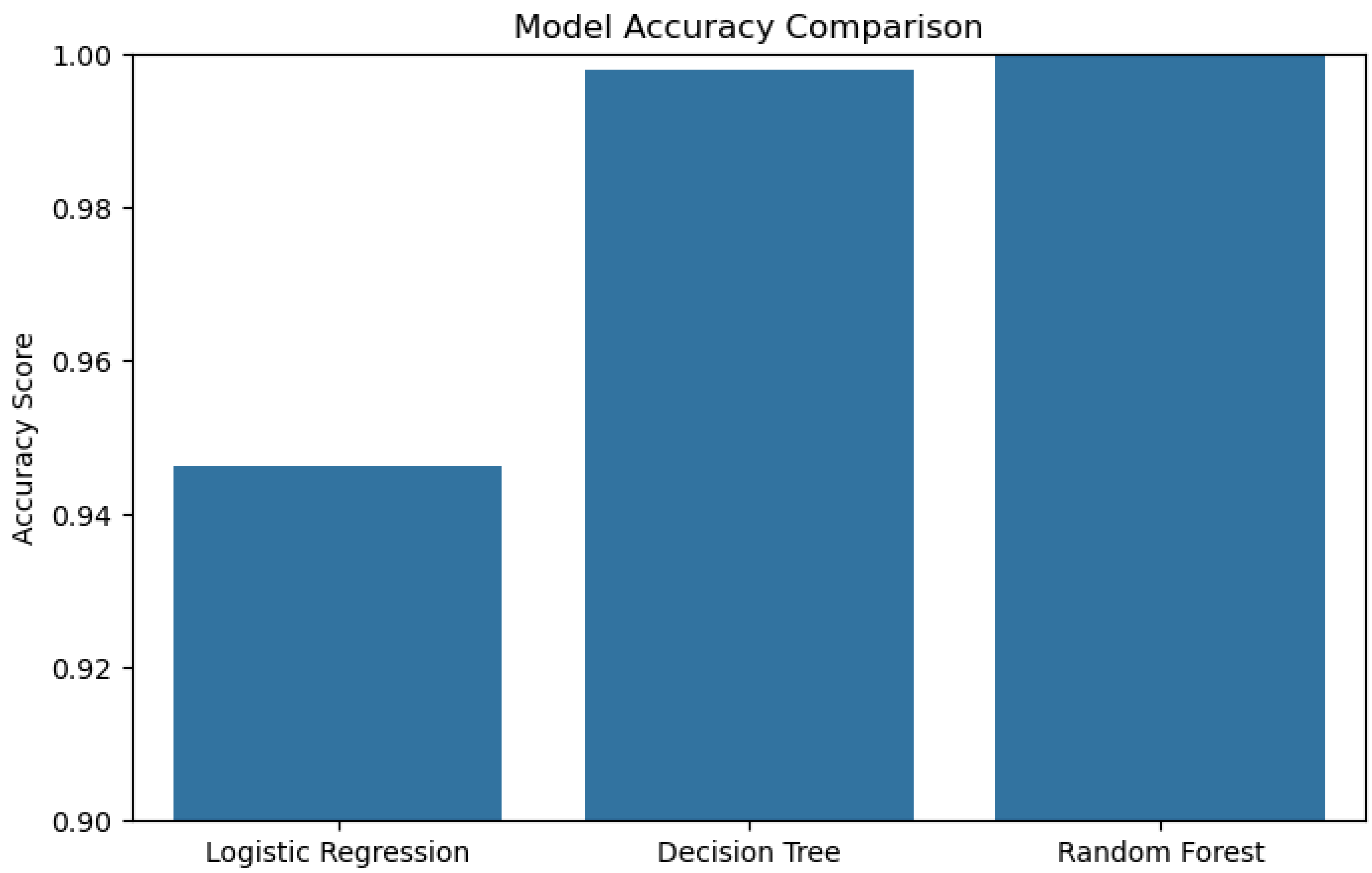



```
Random Forest Results:
Accuracy: 0.9998681040395336
Precision: 0.9997362776468933
Recall: 1.0
F1 Score: 0.9998681214337838
```



13. Compare Models

In [23]:
```
plt.figure(figsize=(8,5))
sns.barplot(x=list(accuracy_scores.keys()), y=list(accuracy_scores.values()))
plt.ylabel('Accuracy Score')
plt.title('Model Accuracy Comparison')
plt.ylim(0.9, 1.0)
plt.show()
```



14. Save the Best Model

In [24]:
```
best_model_name = max(accuracy_scores, key=accuracy_scores.get)
print("Best Model:", best_model_name)

if best_model_name == 'Logistic Regression':
    joblib.dump(lr, 'fraud_model.pkl')
    model = lr
elif best_model_name == 'Decision Tree':
    joblib.dump(dt, 'fraud_model.pkl')
    model = dt
else:
```

```python
    joblib.dump(rf, 'fraud_model.pkl')
    model = rf
```

```
Best Model: Random Forest
```

15. Predict on New Input

In [26]:
```python
# Load model
model = joblib.load('fraud_model.pkl')


# Replace these with your own 29 feature values
my_input = [[0.3,-0.5,0.7,-0.9,0.6,0.3,0.2,-0.1,0.4,0.1,-0.3,0.5,-0.2,0.0,0.1,0.3,0.4,-0.3,0.2,0.0,0.1,0.0,0.0,0.0,0.0,0.02,0.
# Create a DataFrame with the same feature names as X
input_df = pd.DataFrame(my_input, columns=X.columns)

# Predict
prediction = model.predict(input_df)
print("Prediction:", "Fraudulent" if prediction[0] == 1 else "Normal Transaction")
```

```
Prediction: Normal Transaction
```

In [ ]:
```python
# !streamlit run app.py
```