Build a model to detect fraudulent credit card transactions. Use a dataset containing information about credit card transactions, and experiment with algorithms like Logistic Regression, Decision Trees, or Random Forests to classify transactions as fraudulent or legitimate.

Importing Libraries

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

# Machine Learning Libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Evaluation Metrics
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve, accuracy_score, precision_score, recall_score,

# For Handling Imbalanced Data
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
```

Load and Explore the Dataset

```
# Load the dataset
data_train = pd.read_csv('/content/fraudTrain.csv')
```

```
# Display first few rows
data_train.head()
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	...	36.0788	-81.1781	3495	
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	...	48.8878	-118.2105	149	n
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	...	42.1808	-112.2620	4154	
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	...	46.2306	-112.1138	1939	Pa
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	...	38.4207	-79.4629	99	ps

5 rows × 23 columns

```
# Check for missing values
print(data_train.isnull().sum())
```

```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
```

```
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

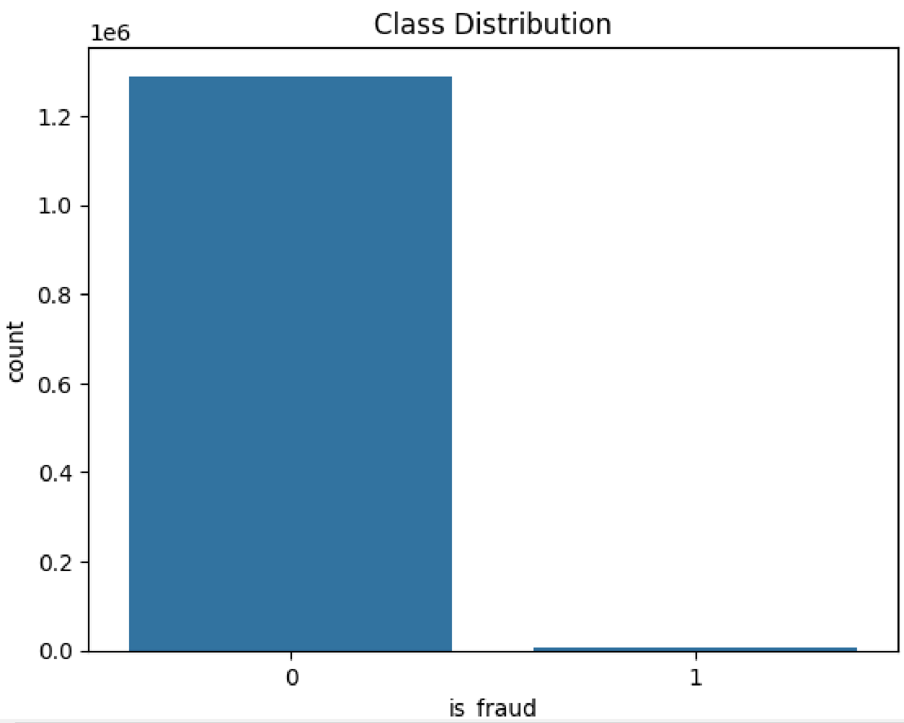
```
#Drop Rows with Missing Values
#data_train = data_train.dropna()
```

```
# Understand the distribution of classes
print(data_train['is_fraud'].value_counts())
```

```
is_fraud
0    1289169
1       7506
Name: count, dtype: int64
```

Exploratory Data Analysis

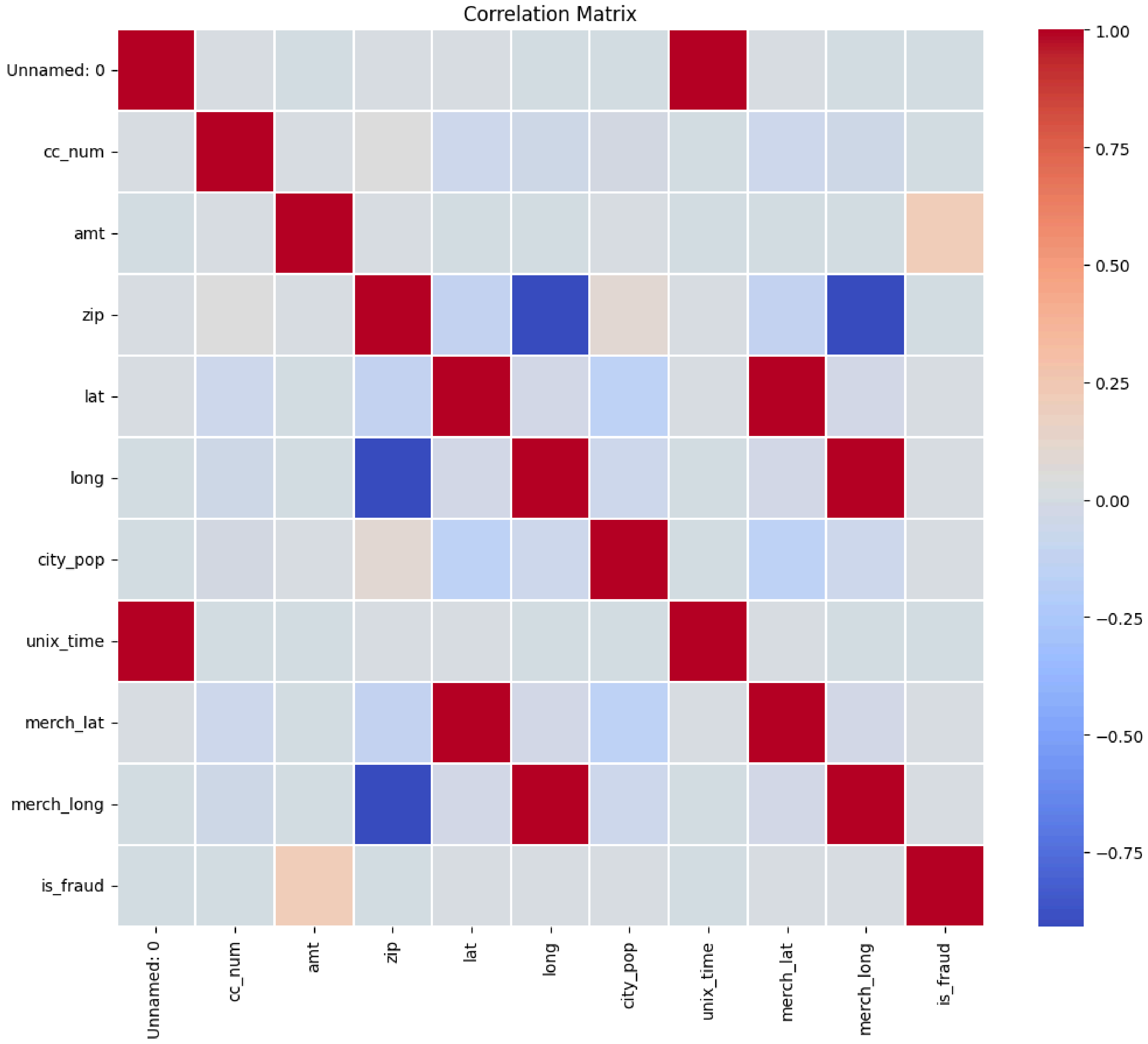
```
# Distribution of the classes
sns.countplot(x='is_fraud', data=data_train)
plt.title('Class Distribution')
plt.show()
```




```
# Statistical summary
data_train.describe()
```

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.882444e+04	1.349244e+09	3.853734e+01	-9.022646e+01	5.788652e-03
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.019564e+05	1.284128e+07	5.109788e+00	1.377109e+01	7.586269e-02
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902779e+01	-1.666712e+02	0.000000e+00
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.338751e+09	3.473357e+01	-9.689728e+01	0.000000e+00
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.349250e+09	3.936568e+01	-8.743839e+01	0.000000e+00
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.359385e+09	4.195716e+01	-8.023680e+01	0.000000e+00
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.371817e+09	6.751027e+01	-6.695090e+01	1.000000e+00

```
# Select only numeric columns for correlation matrix
numeric_data = data_train.select_dtypes(include=[np.number])

# Correlation matrix
plt.figure(figsize=(12,10))
sns.heatmap(numeric_data.corr(), cmap='coolwarm', linewidths=0.1)
plt.title('Correlation Matrix')
plt.show()
```



Data Preprocessing

Since the goal is to detect fraud based on transaction behavior, merchant name might not be the most relevant feature, as it could be specific to a few instances rather than helping to generalize a broader pattern of fraud.

```
# Features and target variable
X = data_train.drop(['first', 'last','gender','street', 'city', 'state', 'job', 'dob', 'trans_num', 'trans_date_trans_time','merchant','is_fra
y = data_train['is_fraud']
```

Handling Class Imbalance

Fraud detection datasets are typically highly imbalanced. Using techniques like SMOTE (Synthetic Minority Over-sampling Technique) can help balance the classes.

```
# Before balancing
print("Before SMOTE:", y.value_counts())
```

```
Before SMOTE: is_fraud
0    1289169
1       7506
Name: count, dtype: int64
```

```
# Initialize label encoder
label_encoder = LabelEncoder()

# Transform the 'Category' column
X['category'] = label_encoder.fit_transform(X['category'])

print(X['category'].head())
```

```
0    8
1    4
2    0
3    2
4    9
Name: category, dtype: int64
```

```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

# After balancing
print("After SMOTE:", y_res.value_counts())
```

```
After SMOTE: is_fraud
0    1289169
1    1289169
Name: count, dtype: int64
```

Feature Scaling

It's essential to scale features, especially for algorithms like Logistic Regression.

```
scaler = StandardScaler()
X_res_scaled = scaler.fit_transform(X_res)
```

Split the Dataset

```
X_train, X_test, y_train, y_test = train_test_split(X_res_scaled, y_res, test_size=0.3, random_state=42, stratify=y_res)
print(X_train.shape, X_test.shape)
```

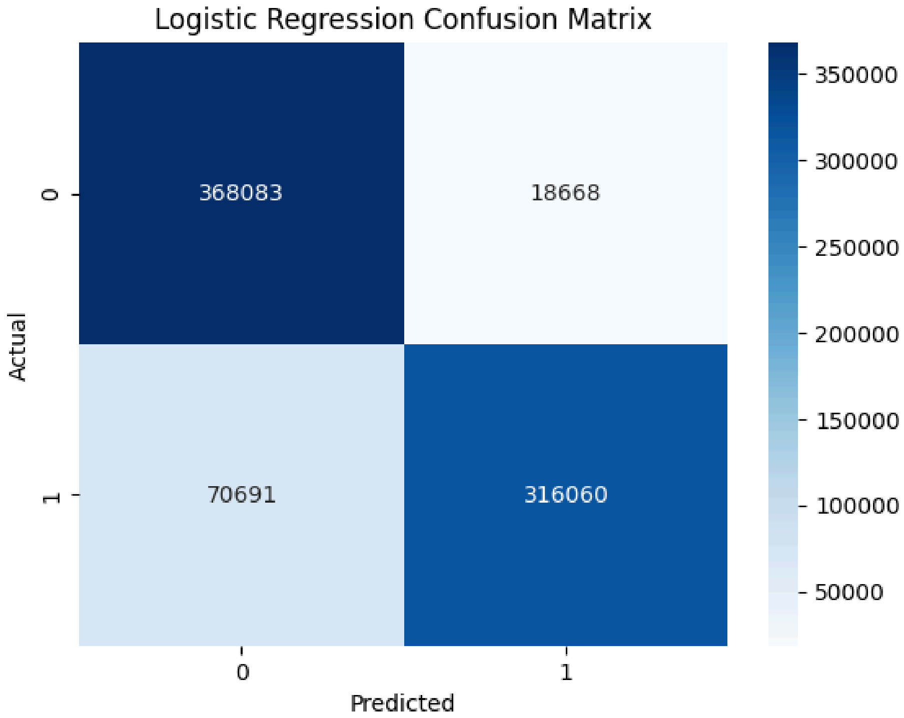
```
(1804836, 11) (773502, 11)
```

Model Training and Evaluation

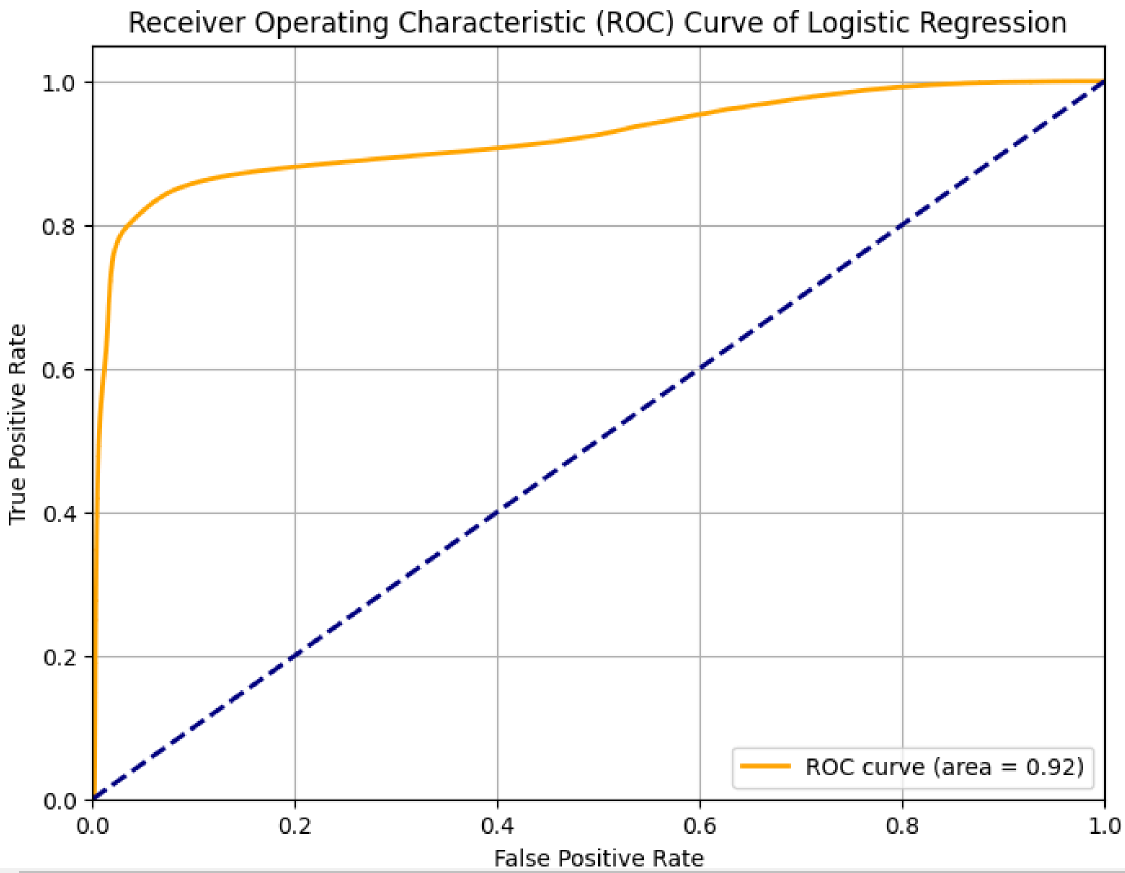
Logistic Regression

```
# Initialize the model
lr = LogisticRegression()

# Train the model
lr.fit(X_train, y_train)

# Predict
y_pred_lr = lr.predict(X_test)
y_prob_lr = lr.predict_proba(X_test)[:,1]

# Evaluation
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr))

print("ROC AUC Score:", roc_auc_score(y_test, y_prob_lr))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.95      0.89    386751
           1       0.94      0.82      0.88    386751

    accuracy                           0.88    773502
   macro avg       0.89      0.88      0.88    773502
weighted avg       0.89      0.88      0.88    773502

ROC AUC Score: 0.922022955730631
```




```
# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob_lr)

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Logistic Regression')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

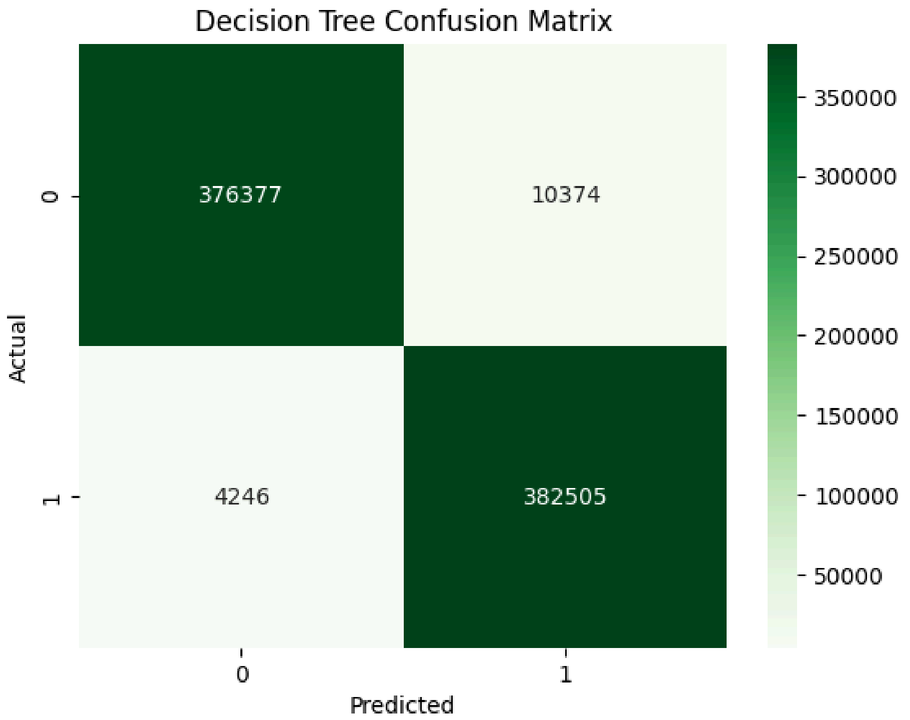


Decision Tree

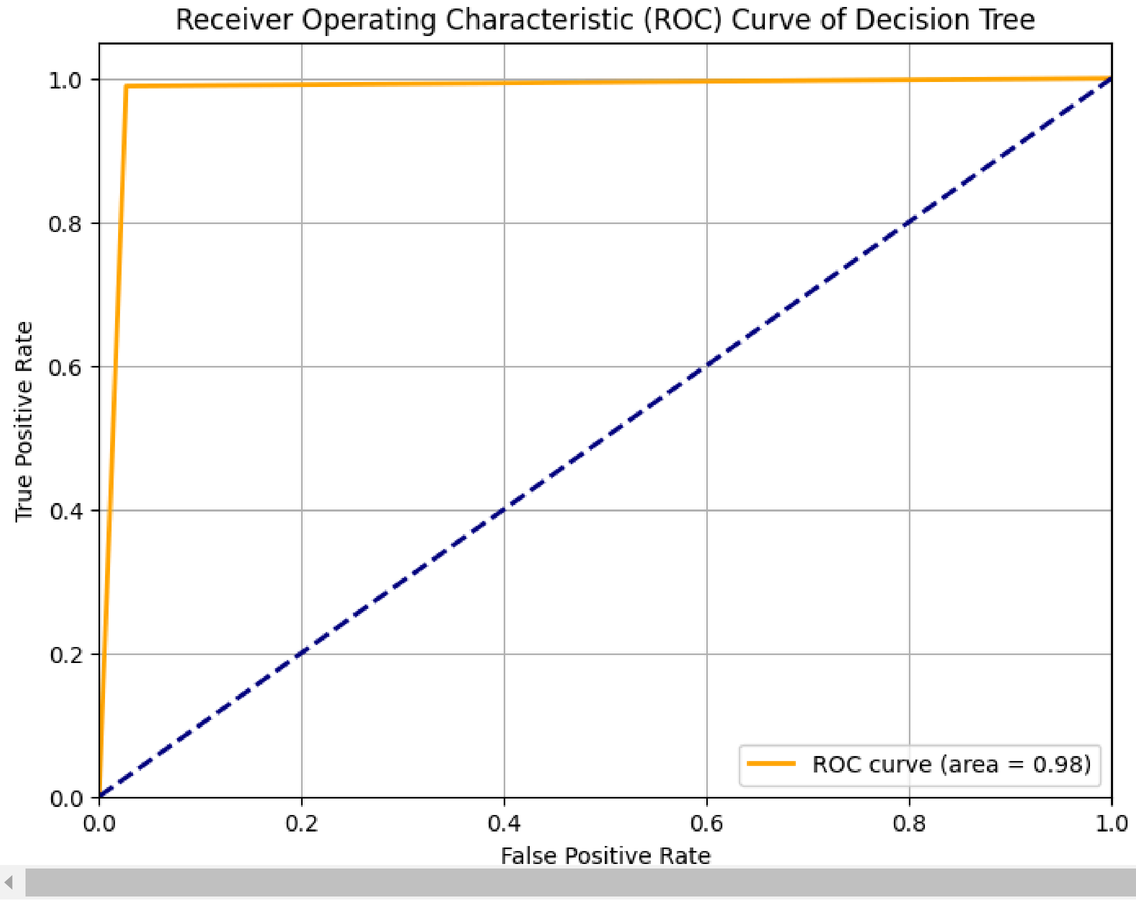
```
# Initialize the model
dt = DecisionTreeClassifier(random_state=42)

# Train the model
```

```
dt.fit(X_train, y_train)

# Predict
y_pred_dt = dt.predict(X_test)
y_prob_dt = dt.predict_proba(X_test)[:,1]

# Evaluation
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))

print("ROC AUC Score:", roc_auc_score(y_test, y_prob_dt))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98    386751
           1       0.97      0.99      0.98    386751

    accuracy                           0.98    773502
   macro avg       0.98      0.98      0.98    773502
weighted avg       0.98      0.98      0.98    773502

ROC AUC Score: 0.9810989499703945
```



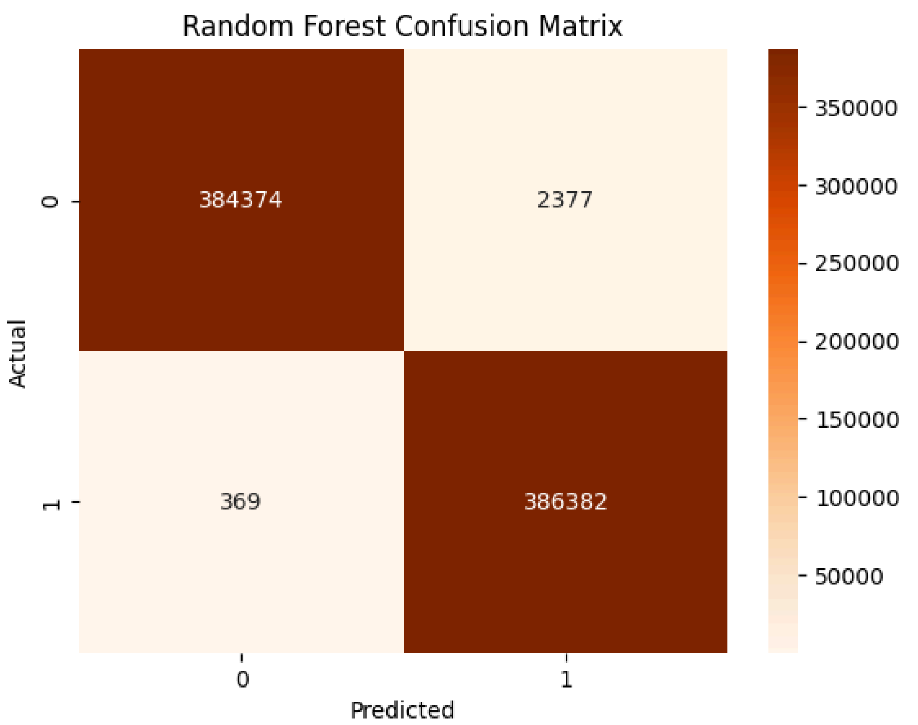

```
# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob_dt)

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Decision Tree')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



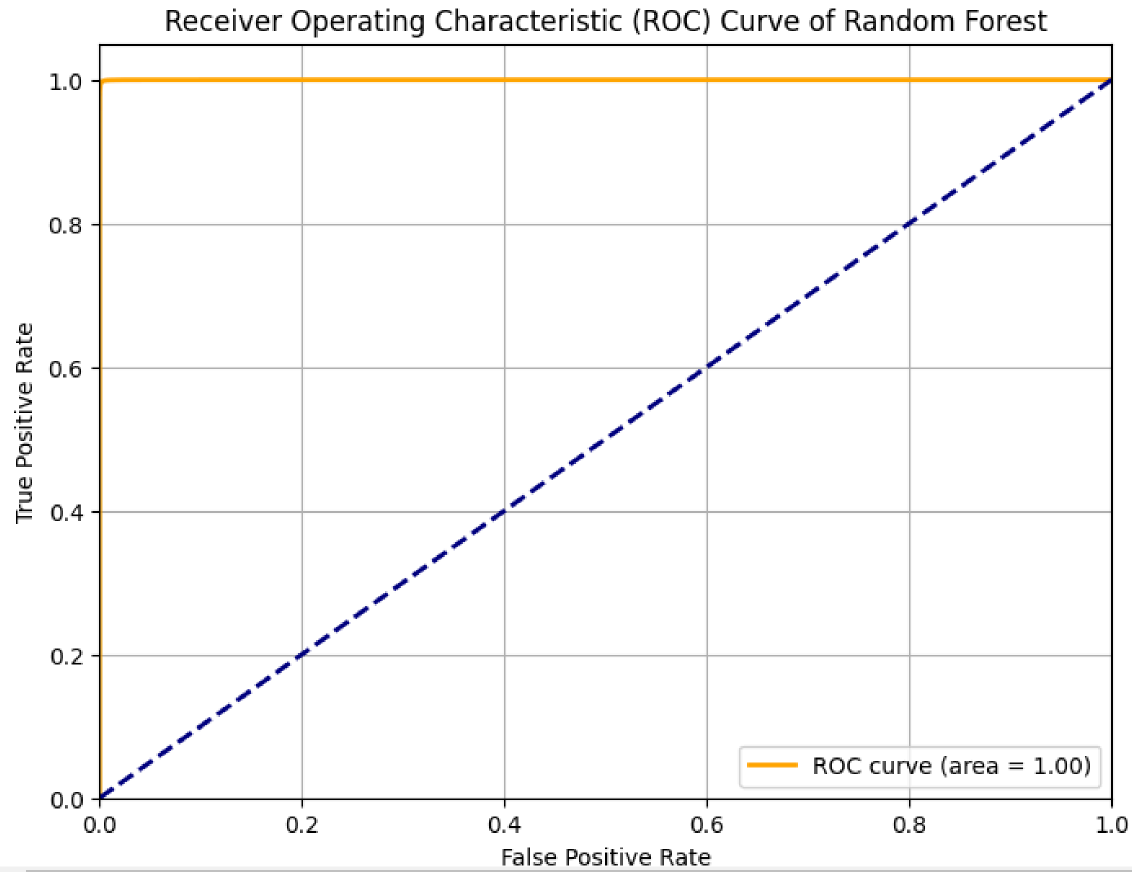
Random Forest

```
# Initialize the model
rf = RandomForestClassifier(random_state=42)

# Train the model
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:,1]

# Evaluation
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

print("ROC AUC Score:", roc_auc_score(y_test, y_prob_rf))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

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

           0       1.00      0.99      1.00    386751
           1       0.99      1.00      1.00    386751

    accuracy                           1.00    773502
   macro avg       1.00      1.00      1.00    773502
weighted avg       1.00      1.00      1.00    773502

ROC AUC Score: 0.9999636208831859
```



```
# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob_rf)

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Random Forest')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

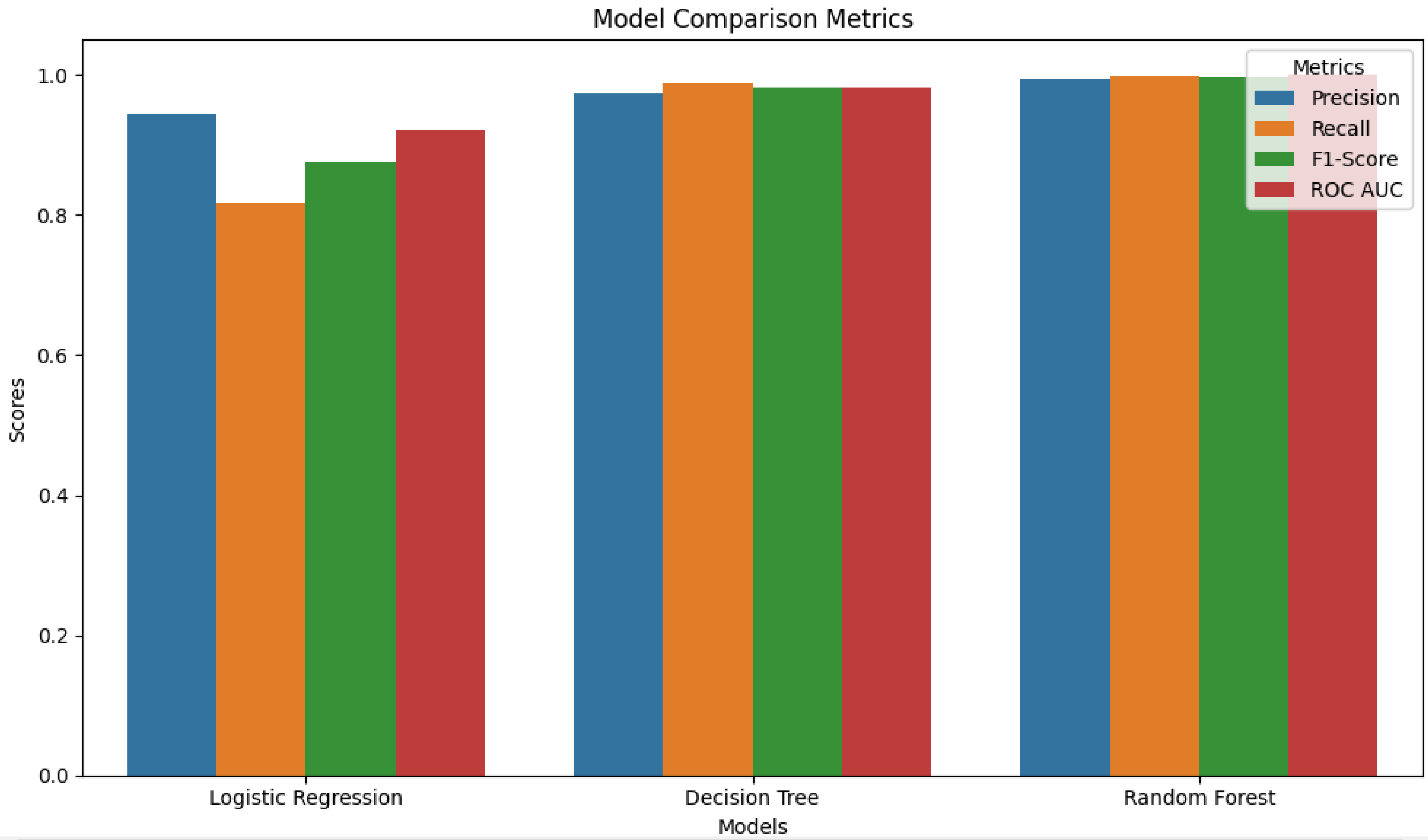


Comparing Models

```
# Create a dataframe to compare metrics
models = ['Logistic Regression', 'Decision Tree', 'Random Forest']
precision = [
    precision_score(y_test, y_pred_lr),
    precision_score(y_test, y_pred_dt),
    precision_score(y_test, y_pred_rf)
]
recall = [
    recall_score(y_test, y_pred_lr),
    recall_score(y_test, y_pred_dt),
    recall_score(y_test, y_pred_rf)
]
f1 = [
    f1_score(y_test, y_pred_lr),
    f1_score(y_test, y_pred_dt),
    f1_score(y_test, y_pred_rf)
]
roc_auc = [
    roc_auc_score(y_test, y_prob_lr),
    roc_auc_score(y_test, y_prob_dt),
    roc_auc_score(y_test, y_prob_rf)
]

comparison = pd.DataFrame({
    'Model': models,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1,
    'ROC AUC': roc_auc
})

print(comparison)
```

```
                 Model  Precision    Recall  F1-Score   ROC AUC
0  Logistic Regression   0.944229  0.817218  0.876145  0.922023
1        Decision Tree   0.973595  0.989021  0.981248  0.981099
2        Random Forest   0.993886  0.999046  0.996459  0.999964
```

```
import seaborn as sns

# Reshape the DataFrame to long format
comparison_melted = comparison.melt(id_vars='Model', var_name='Metric', value_name='Score')

# Create the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=comparison_melted, x='Model', y='Score', hue='Metric')

# Add labels and title
plt.title('Model Comparison Metrics')
plt.ylabel('Scores')
plt.xlabel('Models')
plt.legend(title='Metrics')
plt.tight_layout()

# Display the plot
plt.show()
```



---BELOW BLOCKS ARE EXPERIMENTATION---

-

Hyperparameter Tuning