CREDIT CARD FRAUD TRANSACTION DETECTION USING MACHINE LEARNING.

In [251...
```
#Importing the dataset
```

In [253...
```
import pandas as pd
```

In [255...
```
import numpy as np
```

In [257...
```
import matplotlib.pyplot as plt
```

In [259...
```
import seaborn as sns
```

In [261...
```
from sklearn.model_selection import train_test_split
```

In [263...
```
from sklearn.preprocessing import StandardScaler
```

In [265...
```
from sklearn.ensemble import RandomForestClassifier
```

In [267...
```
from sklearn.metrics import classification_report
```

In [269...
```
from sklearn.metrics import confusion_matrix
```

In [273...
```
from sklearn.metrics import roc_curve
```

In [279...
```
from sklearn.metrics import roc_auc_score
```

In [281...
```
from sklearn.metrics import precision_recall_curve, average_precision_score
```

In [283...
```
from imblearn.over_sampling import SMOTE
```

In [285...
```
from sklearn.decomposition import PCA
```

In [287...
```
import plotly.express as px
```

In [289...
```
#Loading the dataset
```

In [291...
```
data = pd.read_csv('/Users/udaykumar/Desktop/creditcard_2023.csv')
```

In [293...
```
print(data.head())
```

```
   id        V1        V2        V3        V4        V5        V6        V7  
\
0   0 -0.260648 -0.469648  2.496266 -0.083724  0.129681  0.732898  0.519014   
1   1  0.985100 -0.356045  0.558056 -0.429654  0.277140  0.428605  0.406466   
2   2 -0.260272 -0.949385  1.728538 -0.457986  0.074062  1.419481  0.743511   
3   3 -0.152152 -0.508959  1.746840 -1.090178  0.249486  1.143312  0.518269   
4   4 -0.206820 -0.165280  1.527053 -0.448293  0.106125  0.530549  0.658849   

         V8        V9  ...       V21       V22       V23       V24       V25  
\
0 -0.130006  0.727159  ... -0.110552  0.217606 -0.134794  0.165959  0.126280   
1 -0.133118  0.347452  ... -0.194936 -0.605761  0.079469 -0.577395  0.190090   
2 -0.095576 -0.261297  ... -0.005020  0.702906  0.945045 -1.154666 -0.605564   
3 -0.065130 -0.205698  ... -0.146927 -0.038212 -0.214048 -1.893131  1.003963   
4 -0.212660  1.049921  ... -0.106984  0.729727 -0.161666  0.312561 -0.414116   

        V26       V27       V28    Amount  Class  
0 -0.434824 -0.081230 -0.151045  17982.10      0  
1  0.296503 -0.248052 -0.064512   6531.37      0  
2 -0.312895 -0.300258 -0.244718   2513.54      0  
3 -0.515950 -0.165316  0.048424   5384.44      0  
4  1.071126  0.023712  0.419117  14278.97      0  

[5 rows x 31 columns]
```

In [295…

```
#Checking for the missing values
```

In [297…

```
print(data.isnull().sum())
```

```
id        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 [299...

```
#Scale Amount feature.
```

In [301...

```
scaler = StandardScaler()
```

In [303...

```
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
```

In [192...

```
#Dropping time column as it is not needed for the analysis
```

In [305...

```
data.drop(['id'], axis=1, inplace=True)
```

In [307...

```
print(data.columns)
```

```
Index(['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')
```

In [309...

```
x = data.drop('Class', axis=1)
```

In [311...

```
y = data['Class']
```

In [313…

```
#Splitting the data into training and testing.
```

In [315…

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, ran
```

In [317…

```
# Applying SMOTE to handle class imbalance
```

In [319…

```
smote = SMOTE(random_state=42)
```

In [323…

```
x_train_res, y_train_res = smote.fit_resample(x_train, y_train)
```

In [325…

```
#Train a RandomForestClassifier
```

In [327…

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [329…

```
model.fit(x_train, y_train)
```

Out[329]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [331…

```
#Predicting on the test set
```

In [333…

```
y_pred = model.predict(x_test)
```

In [335…

```
#Evaluating the model
```

In [337…

```
print(confusion_matrix(y_test, y_pred))
```

```
[[85261    34]
 [    0 85294]]
```

In [339…

```
print(classification_report(y_test, y_pred))
```

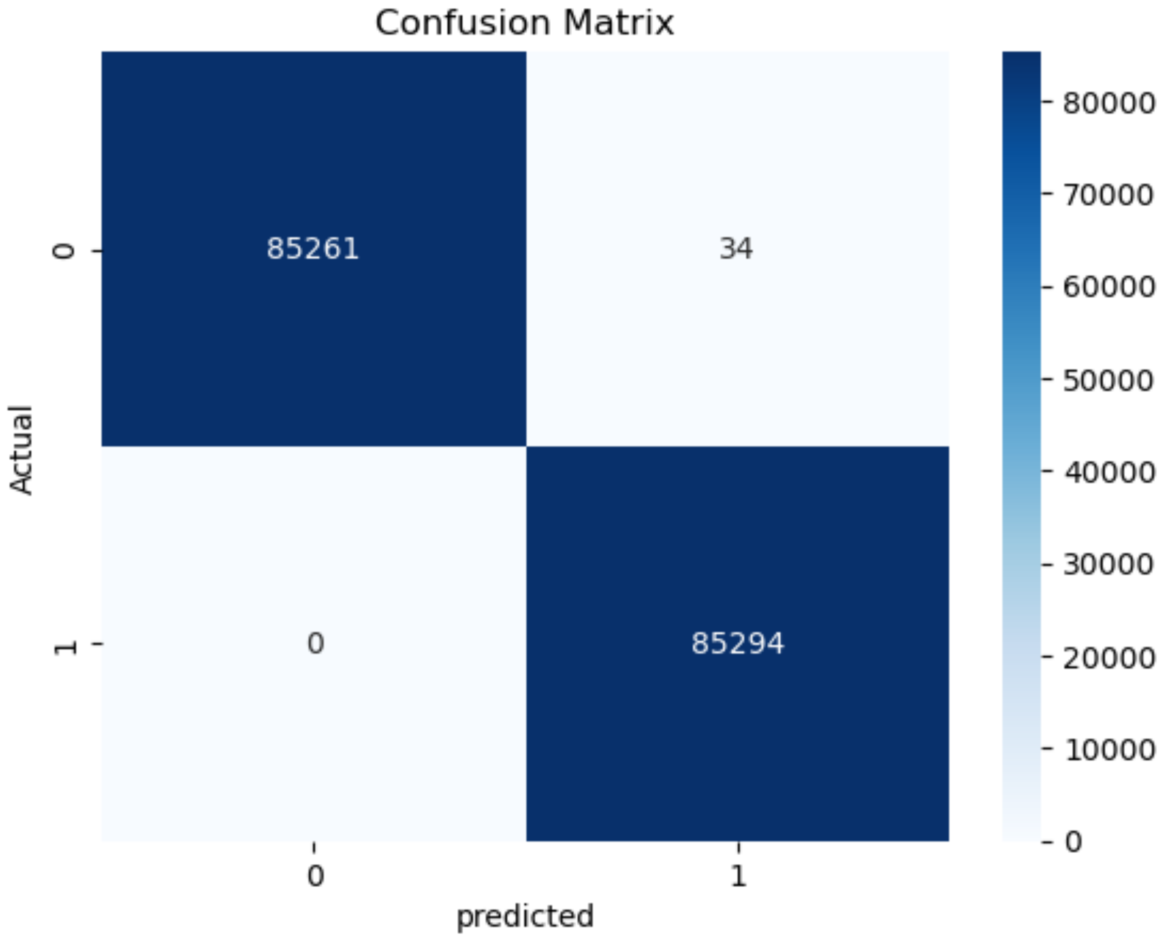
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       1.00      1.00      1.00     85294

    accuracy                           1.00    170589
   macro avg       1.00      1.00      1.00    170589
weighted avg       1.00      1.00      1.00    170589
```

In [239…

```
# Plotting confusion matrix
```

In [345…

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blu
plt.xlabel('predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

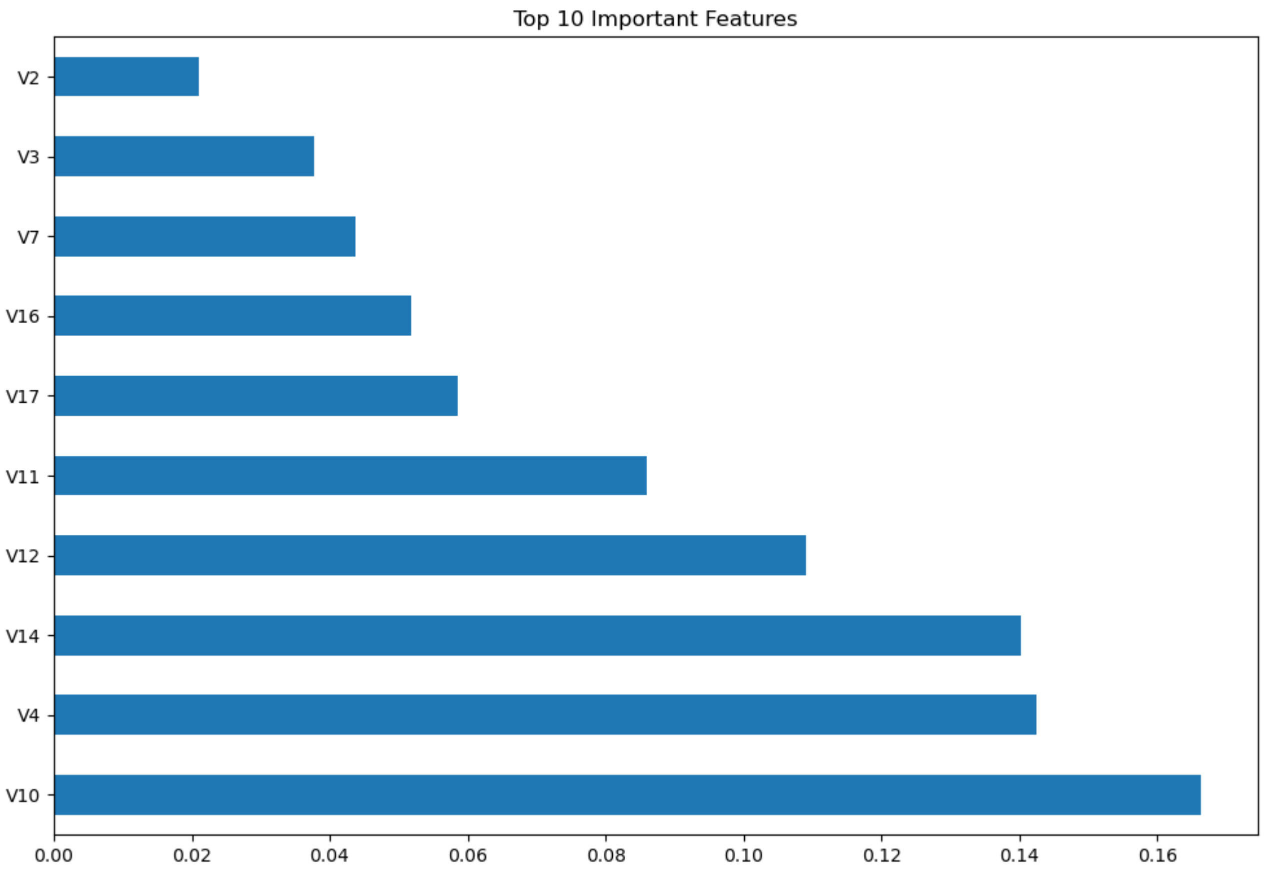


In [347…

```
# Feature Importance Plot
```

In [353…

```
feature_importances = pd.Series(model.feature_importances_, index=x.columns)
plt.figure(figsize=(12,8))
feature_importances.nlargest(10).plot(kind='barh')
plt.title('Top 10 Important Features')
plt.show()
```

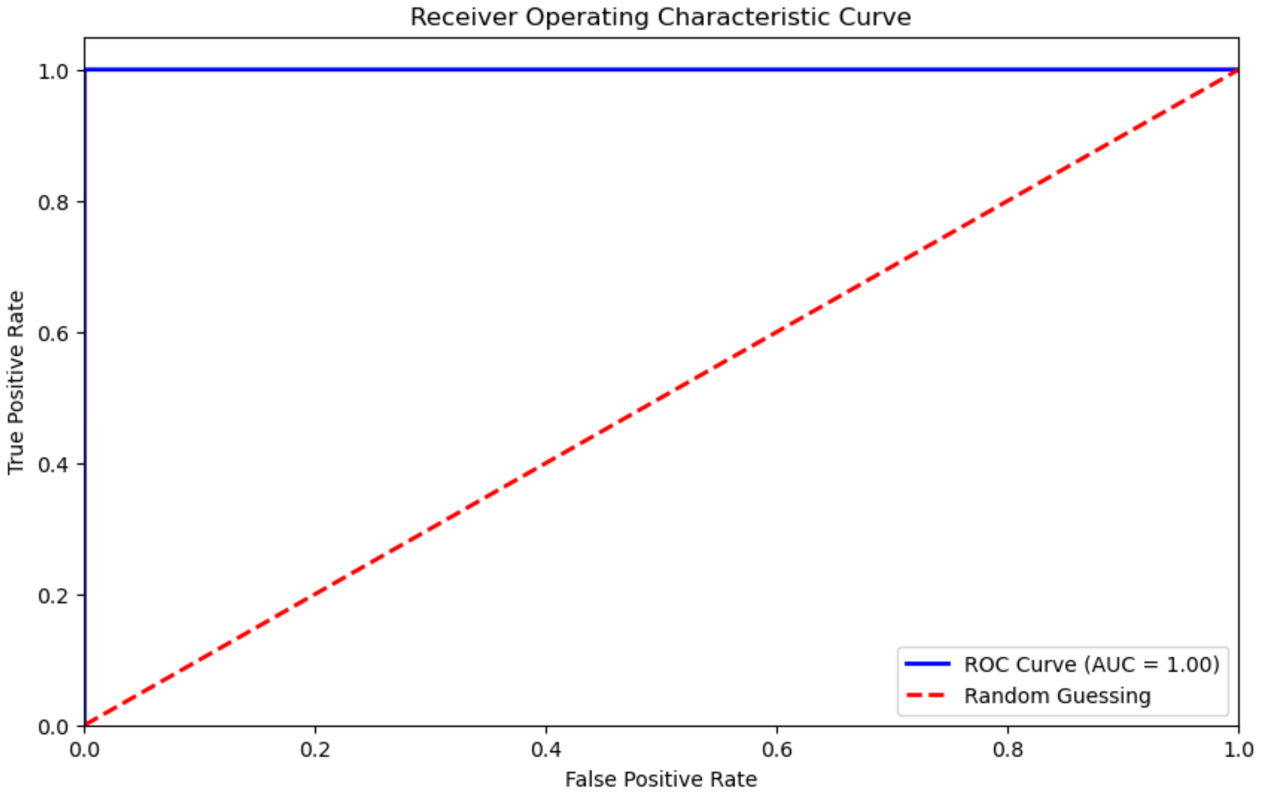


In [355…

```
# ROC CURVE AND AUC
```

In [365…

```
y_prob = model.predict_proba(x_test) [:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
plt.figure(figsize=(10,6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC Curve (AUC = %0.2F)' % roc
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random Gu
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 Curve')
plt.legend(loc="lower right")
plt.show()
```

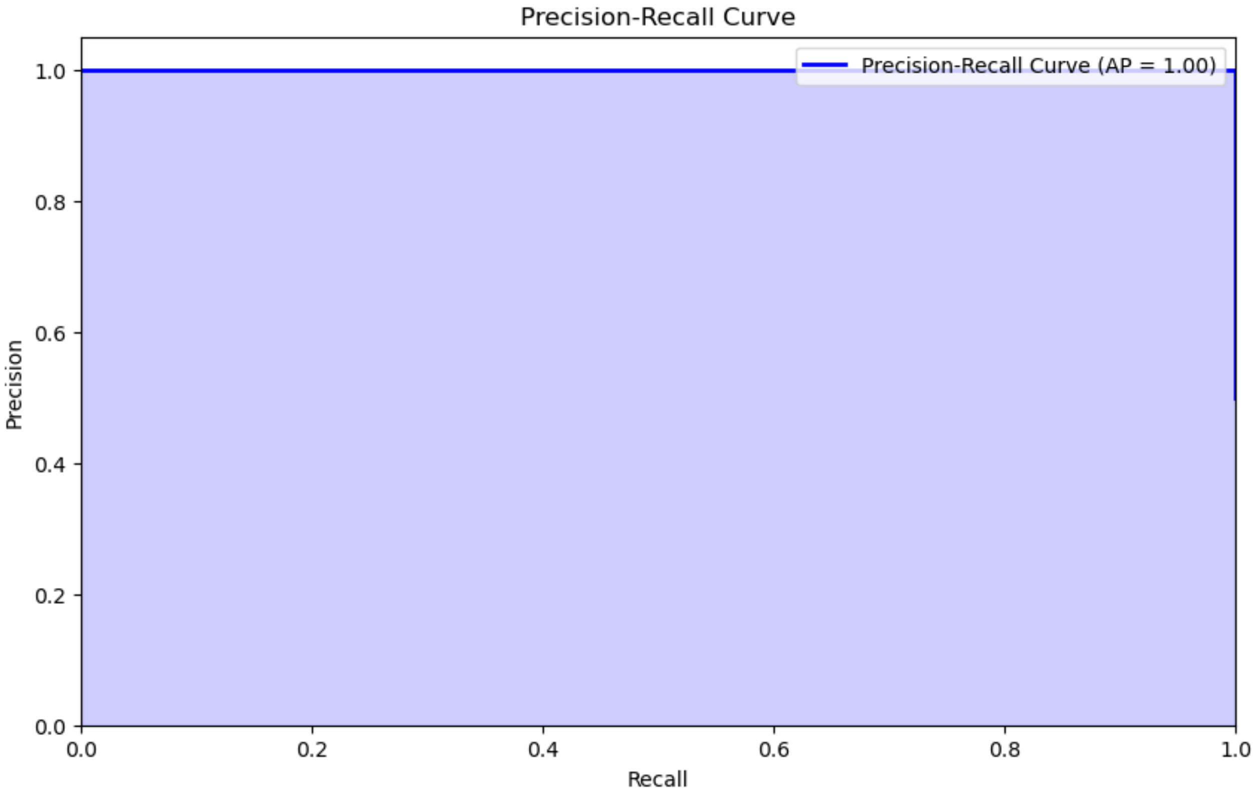


In [ ]:

```
#Precision Recall Curve
```

In [367…

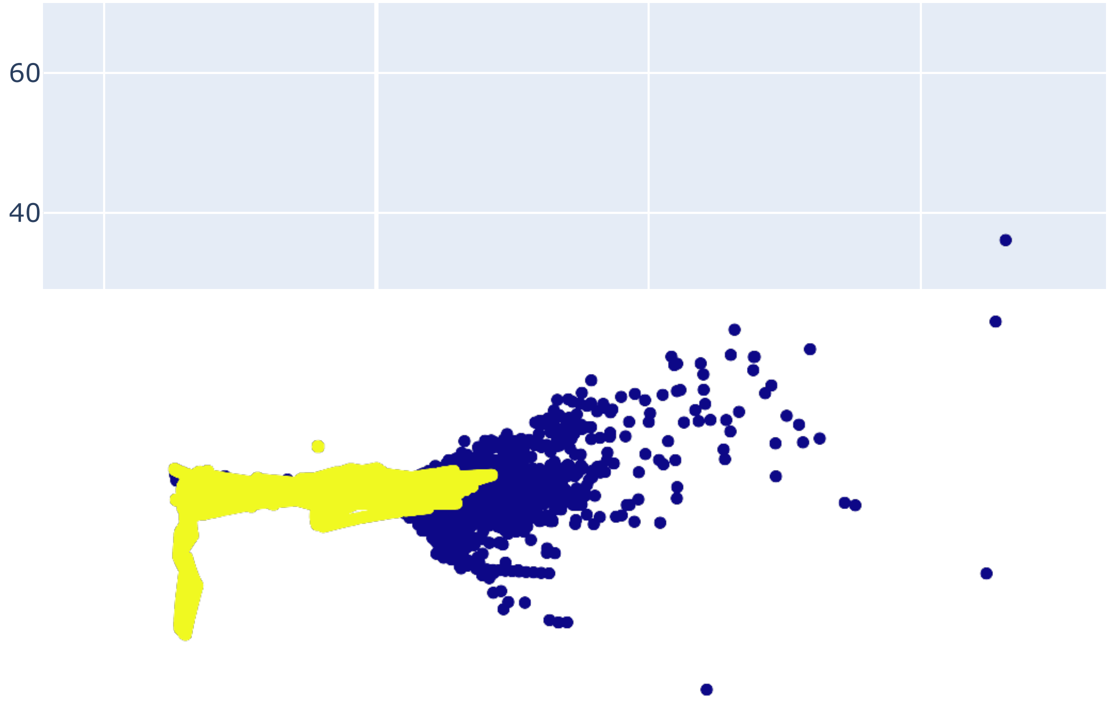
```
precision, recall, _ = precision_recall_curve(y_test, y_prob)
avg_precision = average_precision_score(y_test, y_prob)
plt.figure(figsize=(10, 6))
plt.step(recall, precision, color='blue', where='post', lw=2, label='Precisi
plt.fill_between(recall, precision, step='post', alpha=0.2, color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.show()
```



In [369…

```
pca = PCA(n_components=2)
x_pca = pca.fit_transform(x)
x_pca_df = pd.DataFrame(x_pca, columns=['PC1', 'PC2'])
x_pca_df['Class'] = y
fig = px.scatter(x_pca_df, x='PC1', y='PC2', color='Class', title='PCA Visua
fig.show()
```

PCA Visualization of Credit Card Transactions



```
In [ ]:
```