MACHINE LEARNING

Submitted by,

- Name:Jagannath v v

- Reg.No.-21122024

- class:2MSCDS

Overview

Objectives

make and fine-tune a model, which is ready for deployment.

• Find out a Dataset, and compare at least two different algorithms and choose the best one

• Use suitable Data Preprocessing and Feature Selection/Engineering Methods

• Fine tune the model and hyper parameters and Finalise the Model

• Make the model deployment-ready by giving User-Input provision

Problem Definition

For above mentioned objectives, detailed analysis are being performed in this lab. And the purpose of the lab is to have proper understanding of the classifier and tried to figure out which one of the classifier is best.

Approach

Here i reffered internet to find some more about the classification and try to find which classifier is better

IMPORTING LIBRARIES

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import roc_curve,roc_auc_score, precision_recall_curve, average_pr
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

LOADING DATASET

In [2]:
```
df=pd.read_csv(r"C:\Users\jagan\OneDrive\Desktop\creditcard.csv")
df.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

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.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

In [6]:
```
df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.0
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.33
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.61
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.6
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.7
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.9
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.33

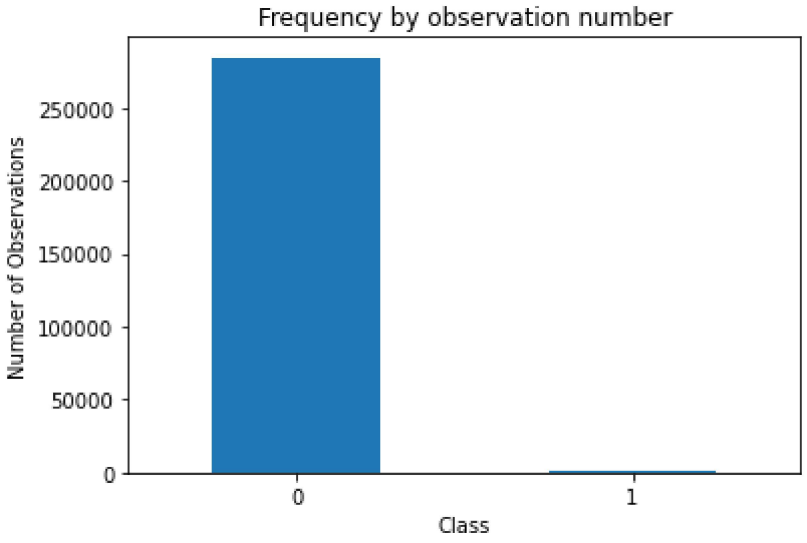
8 rows × 31 columns

In [7]:
```
print(df.Amount.describe())

# Count the occurrences of fraud and no fraud cases
fnf = df["Class"].value_counts()

# Print the ratio of fraud cases
print(fnf/len(df))

# Plottingg your data
plt.xlabel("Class")
```

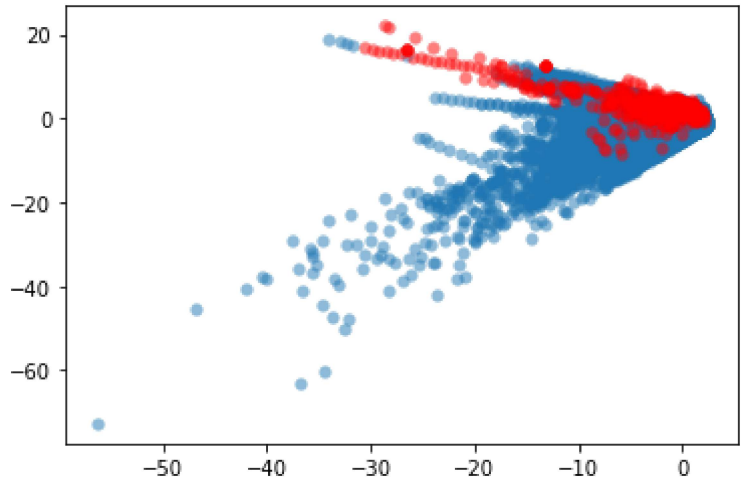
```
plt.ylabel("Number of Observations")
fnf.plot(kind = 'bar',title = 'Frequency by observation number',rot=0)
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Out[7]:
```
<AxesSubplot:title={'center':'Frequency by observation number'}, xlabel='Class', ylabel
='Number of Observations'>
```



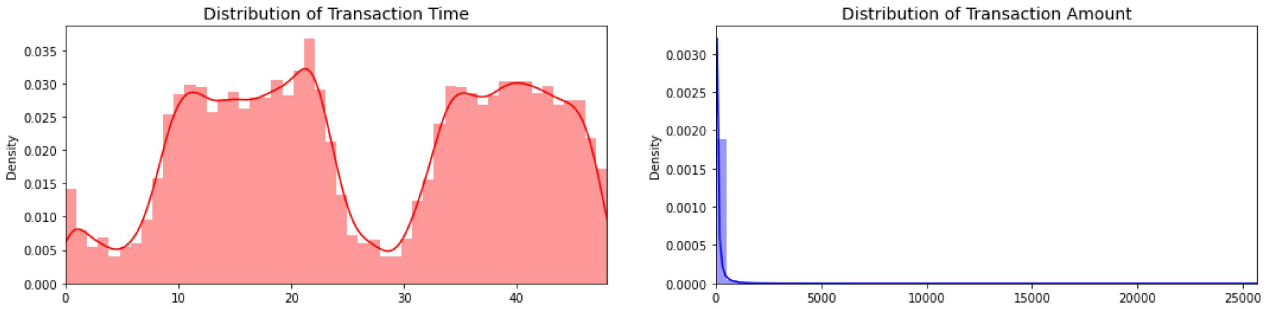
In [8]:
```
# Plot how fraud and non-fraud cases are scattered
plt.scatter(df.loc[df['Class'] == 0]['V1'], df.loc[df['Class'] == 0]['V2'], label="Clas
plt.scatter(df.loc[df['Class'] == 1]['V1'], df.loc[df['Class'] == 1]['V2'], label="Clas
plt.show()
```



In [9]:
```
import seaborn as sns
```

```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

# Plot the distribution of 'Time' feature
sns.distplot(df['Time'].values/(60*60), ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Time', fontsize=14)
ax[0].set_xlim([min(df['Time'].values/(60*60)), max(df['Time'].values/(60*60))])

sns.distplot(df['Amount'].values, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Amount', fontsize=14)
ax[1].set_xlim([min(df['Amount'].values), max(df['Amount'].values)])

plt.show()
```

```
C:\Users\jagan\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\jagan\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [10]:

```
# Seperate total data into non-fraud and fraud cases
df_nonfraud = df[df.Class == 0] #save non-fraud df observations into a separate df
df_fraud = df[df.Class == 1] #do the same for frauds
```

Compare the Amount of transactions in two separate datasets

In [11]:

```
# Summarize statistics and see differences between fraud and normal transactions
print(df_nonfraud.Amount.describe())
print('_'*25)
print(df_fraud.Amount.describe())

# Import the module
from scipy import stats
F, p = stats.f_oneway(df['Amount'][df['Class'] == 0], df['Amount'][df['Class'] == 1])
print("F:", F)
print("p:",p)
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
```

```
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
------------------------------
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
F: 9.033344712018891
p: 0.0026512206498171095
```

Summary: The mean transaction amout among fraud cases is 122 USD, and is 88 among non-fraud cases. And the difference is statistically significant.

Transaction Amount Visualization¶

Summary: In the long tail, fraud transaction happened more frequently. It seems It would be hard to differentiate fraud from normal transactions by transaction amount alone.

Feature Scaling

As we know before, features V1-V28 have been transformed by PCA and scaled already. Whereas feature "Time" and "Amount" have not. And considering that we will analyze these two features with other V1-V28, they should better be scaled before we train our model using various algorithms. Here is why and how.

Which scaling method should we use?

The Standard Scaler is not recommended as "Time" and "Amount" features are not normally distributed.

The Min-Max Scaler is also not recommende as there are noticeable outliers in feature "Amount".

The Robust Scaler are robust to outliers: (xi–Q1(x))/(Q3(x)–Q1(x)) (Q1 and Q3 represent 25% and 75% quartiles).

So we choose Robust Scaler to scale these two features.

In [12]:
```
# Scale "Time" and "Amount"
from sklearn.preprocessing import StandardScaler, RobustScaler
df['scaled_amount'] = RobustScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = RobustScaler().fit_transform(df['Time'].values.reshape(-1,1))

# Make a new dataset named "df_scaled" dropping out original "Time" and "Amount"
df_scaled = df.drop(['Time','Amount'],axis = 1,inplace=False)
df_scaled.head()
```

Out[12]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.09(
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.16(
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.20
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.05
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.75

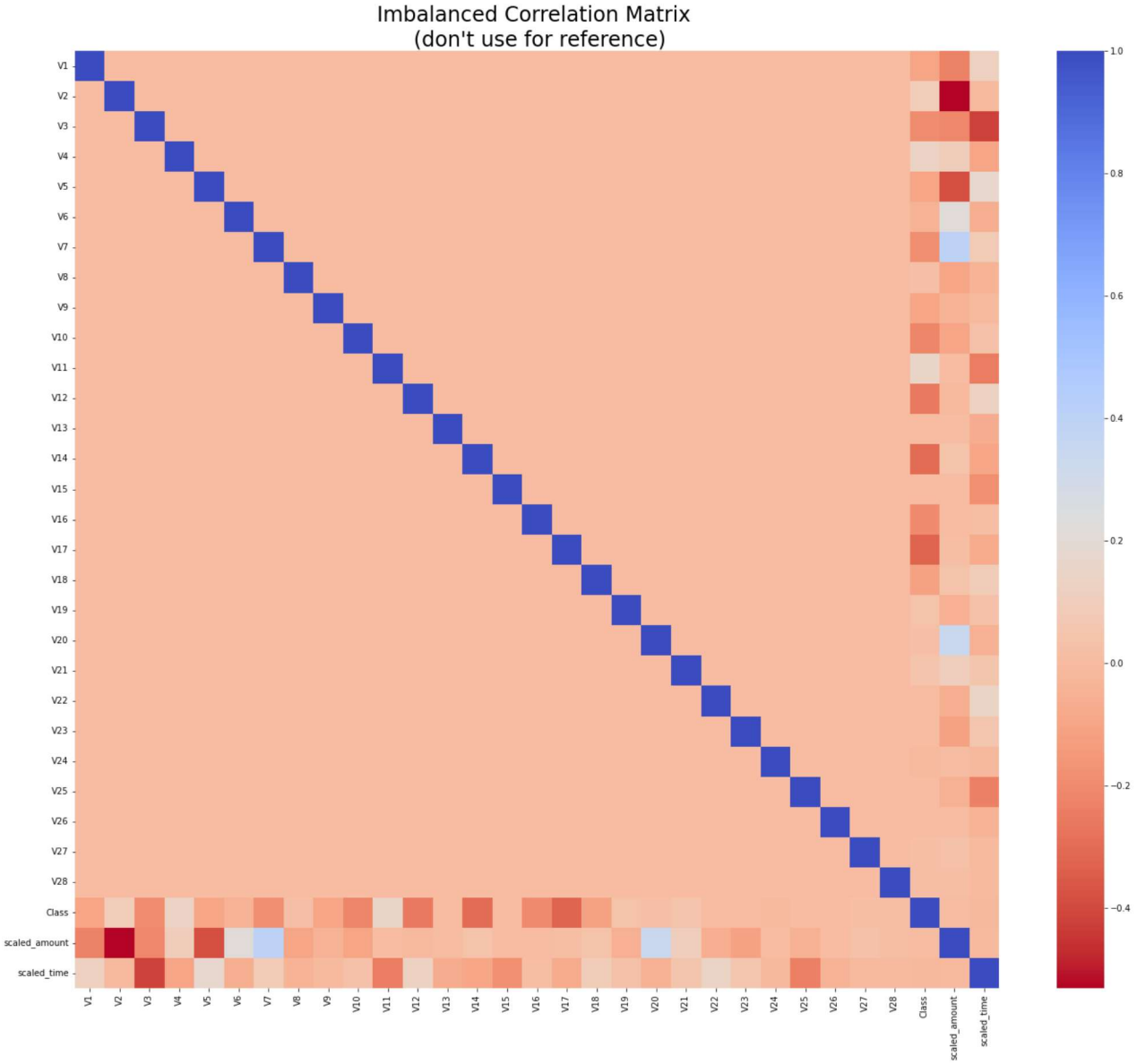
5 rows × 31 columns

Correlation Matrices

Correlation matrices are the essence of understanding our data. We want to know if there are features that influence heavily in whether a specific transaction is a fraud.

In [13]:
```
# Calculate pearson correlation coefficience
corr = df_scaled.corr()

# Plot heatmap of correlation
f, ax = plt.subplots(1, 1, figsize=(24,20))
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20})
ax.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fontsize=24)
```

Out[13]:
```
Text(0.5, 1.0, "Imbalanced Correlation Matrix \n (don't use for reference)")
```



In [14]:
```
# Resampling for Imbalanced Data
```

In [15]:
```
# Define the prep_data function to extrac features
def prep_data(df):
    X = df.drop(['Class'],axis=1, inplace=False) #
    X = np.array(X).astype(np.float)
    y = df[['Class']]
    y = np.array(y).astype(np.float)
    return X,y

# Create X and y from the prep_data function
X, y = prep_data(df_scaled)
```

```
C:\Users\jagan\AppData\Local\Temp/ipykernel_29224/3667396679.py:4: DeprecationWarning: `
np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f
loat` by itself. Doing this will not modify any behavior and is safe. If you specificall
y wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  X = np.array(X).astype(np.float)
```

```
C:\Users\jagan\AppData\Local\Temp/ipykernel_29224/3667396679.py:6: DeprecationWarning: `
np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f
loat` by itself. Doing this will not modify any behavior and is safe. If you specificall
y wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  y = np.array(y).astype(np.float)
```

In [16]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Create the training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)

# Fit a logistic regression model to our data
model = LogisticRegression()
model.fit(X_train, y_train)

# Obtain model predictions
y_predicted = model.predict(X_test)
y_predicted
```

```
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please change the s
hape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[16]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [ ]:

In [17]:

```
# Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, y_predicted)

# Calculate Area Under the Receiver Operating Characteristic Curve
probs = model.predict_proba(X_test)
roc_auc = roc_auc_score(y_test, probs[:, 1])
print('ROC AUC Score:',roc_auc)

# Obtain precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_predicted)

# Calculate average precision
average_precision = average_precision_score(y_test, y_predicted)
print("average_precision is : ",average_precision)
```

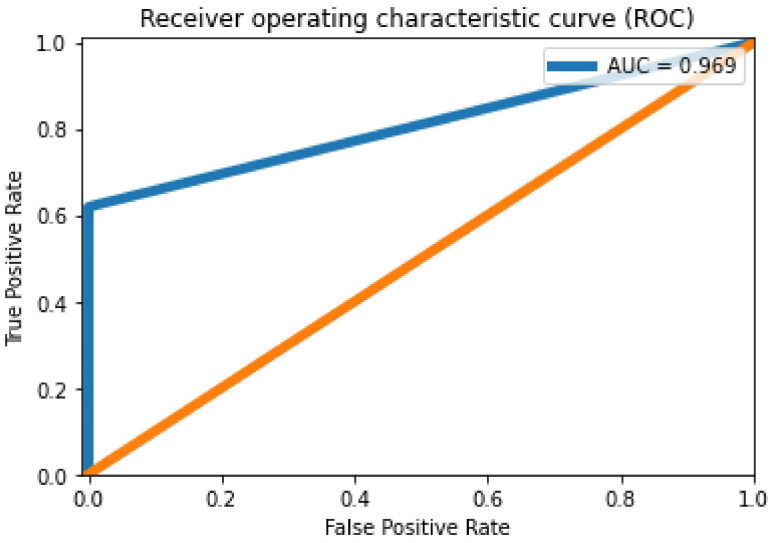
```
ROC AUC Score: 0.9693476386990738
average_precision is :  0.5475809740417419
```

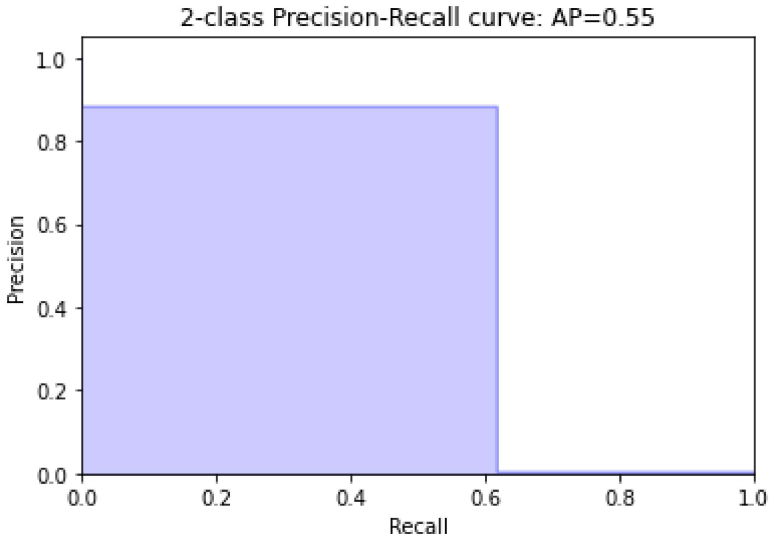
In [18]:
```
# Define a roc_curve function
def plot_roc_curve(false_positive_rate,true_positive_rate,roc_auc):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=5, label='AUC = %0.3f'%
    plt.plot([0,1],[0,1], linewidth=5)
    plt.xlim([-0.01, 1])
    plt.ylim([0, 1.01])
    plt.legend(loc='upper right')
    plt.title('Receiver operating characteristic curve (ROC)')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

In [19]:
```
# Define a precision_recall_curve function
def plot_pr_curve(recall, precision, average_precision):
    plt.step(recall, precision, color='b', alpha=0.2, where='post')
    plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
    plt.show()
```

In [20]:
```
# Print the classifcation report and confusion matrix
print('Classification report:\n', classification_report(y_test, y_predicted))
print('Confusion matrix:\n',confusion_matrix(y_true = y_test, y_pred = y_predicted))

# Plot the roc curve
plot_roc_curve(false_positive_rate,true_positive_rate,roc_auc)

# Plot recall precision curve
plot_pr_curve(recall, precision, average_precision)
print("**********************************************************************")
print("Accuracy is : " ,accuracy_score(y_test, y_predicted))
```

```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.88      0.62      0.73       147

    accuracy                           1.00     85443
   macro avg       0.94      0.81      0.86     85443
weighted avg       1.00      1.00      1.00     85443

Confusion matrix:
 [[85284    12]
 [   56    91]]
```





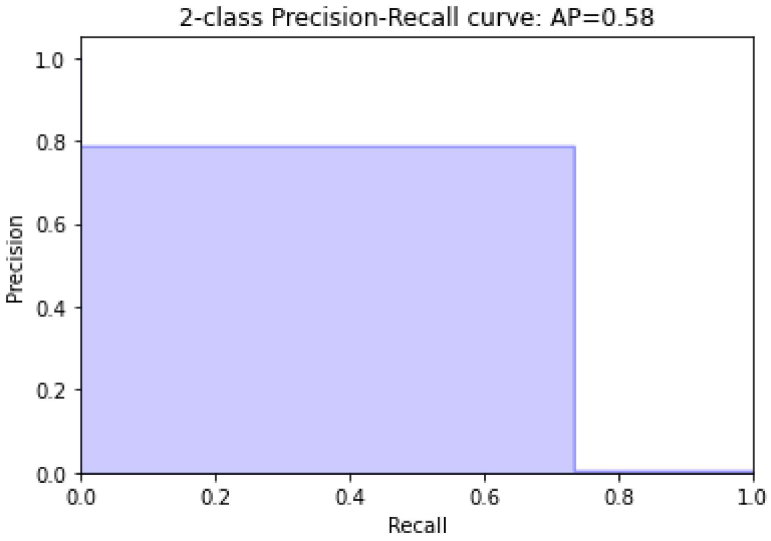

```
**********************************************************************
Accuracy is :  0.999204147794436
```

Decision tree

In [21]:

```
# Import the decision tree model from sklearn
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Create the training and testing sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=.3, random_stat

# Fit a logistic regression model to our data
model = DecisionTreeClassifier()
model.fit(X_train1, y_train1)

# Obtain model predictions
y_predicted = model.predict(X_test1)


# Calculate average precision
```

```
average_precision = average_precision_score(y_test1, y_predicted)

# Obtain precision and recall
precision, recall, _ = precision_recall_curve(y_test1, y_predicted)

# Plot the recall precision tradeoff
plot_pr_curve(recall, precision, average_precision)

# Print the classifcation report and confusion matrix
print('Classification report:\n', classification_report(y_test1, y_predicted))
print("**********************************************************************")
print('Confusion matrix:\n',confusion_matrix(y_true = y_test1, y_pred = y_predicted))
print("**********************************************************************")
print("Accuracy is : " ,accuracy_score(y_test1, y_predicted))
print("**********************************************************************")
```




```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.79      0.73      0.76       147

    accuracy                           1.00     85443
   macro avg       0.89      0.87      0.88     85443
weighted avg       1.00      1.00      1.00     85443

**********************************************************************
Confusion matrix:
 [[85267    29]
 [   39   108]]
**********************************************************************
Accuracy is :  0.999204147794436
**********************************************************************
```

Please make and fine-tune a model, which is ready for deployment.

In [22]:

```
df_new=pd.DataFrame(columns=['Time','v1','v2','v3','v4','v5','v6','v7','v8','v9','v10',
df_new
```

Out[22]:

Time	v1	v2	v3	v4	v5	v6	v7	v8	v9	...	v22	v23	v24	v24	v25	v26	v27	v28	Amount

0 rows × 32 columns

In [24]:

```
for i in range(1):
    Time=float(input("Enter the value for time : "))
    df_new[Time]=Time
    print("****************************************")
    v1=float(input("Enter the value : "))
    df_new[v1]=v1
    print("****************************************")
    v2=float(input("Enter the value : "))
    df_new[v1]=v2
    print("****************************************")
    v3=float(input("Enter the value : "))
    df_new[v1]=v3
    print("****************************************")
    v4=float(input("Enter the value : "))
    df_new[v1]=v4
    print("****************************************")
    v5=float(input("Enter the value : "))
    df_new[v1]=v5
    print("****************************************")
    v6=float(input("Enter the value : "))
    df_new[v1]=v6
    print("****************************************")
    v7=float(input("Enter the value : "))
    df_new[v1]=v7
    print("****************************************")
    v8=float(input("Enter the value : "))
    df_new[v1]=v8
    print("****************************************")
    v9=float(input("Enter the value : "))
    df_new[v1]=v9
    print("****************************************")
    v10=float(input("Enter the value : "))
    df_new[v1]=v10
    print("****************************************")
    v11=float(input("Enter the value : "))
    df_new[v1]=v11
    print("****************************************")
    v12=float(input("Enter the value : "))
    df_new[v1]=v12
    print("****************************************")
    v13=float(input("Enter the value : "))
    df_new[v1]=v13
    print("****************************************")
    v14=float(input("Enter the value : "))
    df_new[v1]=v14
    print("****************************************")
    v15=float(input("Enter the value : "))
    df_new[v1]=v15
    print("****************************************")
    v16=float(input("Enter the value : "))
    df_new[v1]=v16
    print("****************************************")
```

```
    v17=float(input("Enter the value : "))
    df_new[v1]=v17
    print("****************************************")
    v18=float(input("Enter the value : "))
    df_new[v1]=v18
    print("****************************************")
    v19=float(input("Enter the value : "))
    df_new[v1]=v19
    print("****************************************")
    v20=float(input("Enter the value : "))
    df_new[v1]=v20
    print("****************************************")
    v21=float(input("Enter the value : "))
    df_new[v1]=v21
    print("****************************************")
    v22=float(input("Enter the value : "))
    df_new[v1]=v22
    print("****************************************")
    v23=float(input("Enter the value : "))
    df_new[v1]=v23
    print("****************************************")
    v24=float(input("Enter the value : "))
    df_new[v1]=v24
    print("****************************************")
    v25=float(input("Enter the value : "))
    df_new[v1]=v25
    print("****************************************")
    v26=float(input("Enter the value : "))
    df_new[v1]=v26
    print("****************************************")
    v27=float(input("Enter the value : "))
    df_new[v1]=v27
    print("****************************************")
    v28=float(input("Enter the value : "))
    df_new[v1]=v28
    print("****************************************")
    Amount=float(input("Enter the Amount : "))
    df_new[v1]=Amount
    print("****************************************")

    df_new=pd.DataFrame(data=[[Time,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15,


prediction = model.predict(df_new)
if prediction==0:
    print("FRAUD CASE")
if prediction==1:
    print("NOT FRAUD ")
```

```
Enter the value for time : 2.0
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
```

```
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 0.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 1.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the value : 2.060018
****************************************
Enter the Amount : 100.02
****************************************
FRAUD CASE
```

CONCLUSION:

Here i tried explore more about the classification and tried to compare the classifier like which classifier is the best and which classifier give more accurate value. so here i found that decision tree

is more easier compared to other classifier.And last i created a dataframe and store some values ,so from that i am able to predict my model.

REFERENCE

https://www.kaggle.com/

https://www.javatpoint.com/machine-learning

https://www.tutorialspoint.com/scikit_learn/index.htm#:~:text=Scikit%2Dlearn%20(Sklearn)%20is,a%20c

In [ ]: