In [1]:

```
#first we need to import required libraries\

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

# We want our plots to appear in the notebook
%matplotlib inline
```

In [3]:

```
#readind the test data

df=pd.read_csv("Testdata.csv")
df.head(15)
```

```
c:\users\reliance digital\appdata\local\programs\python\python38-32\lib\site-packages\IP
ython\core\interactiveshell.py:3172: DtypeWarning: Columns (15) have mixed types.Specify
dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[3]:

| | tradeId | regulator | version | assetClass | clDateTime | clStatus | cflag | eFlag | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | SEBI | 1 | FX | 2023-07-18T11:47:26.075000Z | True | FULLY | False | 10T |
| **1** | 2 | SEBI | 9 | FX | 2023-08-02T03:12:48.207000Z | False | FULLY | False | 03T |
| **2** | 3 | SEBI | 8 | FX | 2023-05-09T07:42:36.475000Z | False | FULLY | False | 09T |
| **3** | 4 | SEBI | 1 | FX | 2023-10-01T05:02:54.209000Z | False | FULLY | True | 21T |
| **4** | 5 | SEBI | 1 | FX | 2023-01-23T20:53:01.076000Z | True | ONEWAY | False | 05T |
| **5** | 6 | SEBI | 3 | FX | 2023-01-15T02:08:20.740000Z | False | F | True | 28T |
| **6** | 7 | SEBI | 2 | FX | 2023-02-07T04:56:16.002000Z | False | FULLY | True | 30T |
| **7** | 8 | SEBI | 3 | FX | 2023-02-28T16:20:09.730000Z | True | UNCOLLATERALIZED | False | 10T |
| **8** | 9 | SEBI | 8 | FX | 2023-07-05T05:12:36.800000Z | False | UNCOLLATERALIZED | False | 17T |
| **9** | 10 | SEBI | 6 | FX | 2023-07-06T11:46:01.150000Z | True | UNCOLLATERALIZED | True | 28T |
| **10** | 11 | SEBI | 8 | FX | 2023-05-21T06:33:22.779000Z | False | FULLY | False | 24T |
| **11** | 12 | SEBI | 6 | FX | 2023-04-04T13:00:37.821000Z | True | UN | True | 21T |
| **12** | 13 | SEBI | 1 | FX | 2023-11- | False | OW | True | |

| | tradeId | regulator | version | assetClass | clDateTime | clStatus | cflag | eFlag | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | 16T04:21:12.346000Z | | | | 06T2 |
| **13** | 14 | SEBI | 7 | FX | 2023-03-15T13:39:41.082000Z | False | FULLY | False | 01T0 |
| **14** | 15 | SEBI | 4 | FX | 2023-06-21T09:41:39.908000Z | False | UNCOLLATERALIZED | False | 16T2 |

15 rows × 26 columns

In [16]:

```
df['Reporting Status'].value_counts()
#since we are considering Reporting Status as target column
```

Out[16]:

```
Ignored                    12878
Failed Ack                  7289
Error                       7079
ACK                         7045
Processing Error            5354
Failed Acknowledgement      5209
Acknowledged                5146
Name: Reporting Status, dtype: int64
```

In [ ]:

```
#Considering ACK and Acknowlodged as same values
#Considering Remaining values as Failed Acknowledged for easy preprocessing
```

In [39]:

```
df["Reporting Status"]=df["Reporting Status"].replace({"Acknowledged":1,"Failed Acknowl
```

In [44]:

```
df.dtypes
```

Out[44]:

```
tradeId               int64
regulator            object
version               int64
assetClass           object
clDateTime           object
clStatus               bool
cflag                object
eFlag                  bool
cDateTime            object
method               object
rate                float64
expirationDate       object
eventT               object
mType                object
Timestamp            object
quantity             object
seller               object
endDate              object
sType                object
Product              object
price               float64
terminationDate      object
party                object
```

```
PartyId                int64
transactionType       object
Reporting Status       int64
dtype: object
```

In [42]:

```
#filling the missing price values with median and dropping terminationDate rows which ar
```

In [43]:

```
median_price=df["price"].median()
df["price"]=df["price"].fillna(median_price)
df.dropna(inplace=True)
df.isnull().sum()
```

Out[43]:

```
tradeId             0
regulator           0
version             0
assetClass          0
clDateTime          0
clStatus            0
cflag               0
eFlag               0
cDateTime           0
method              0
rate                0
expirationDate      0
eventT              0
mType               0
Timestamp           0
quantity            0
seller              0
endDate             0
sType               0
Product             0
price               0
terminationDate     0
party               0
PartyId             0
transactionType     0
Reporting Status    0
dtype: int64
```

In [104…

```
string_columns = df.select_dtypes(include=['object']).columns
string_columns
```

Out[104…

```
Index(['regulator', 'assetClass', 'clDateTime', 'cflag', 'cDateTime', 'method',
       'expirationDate', 'eventT', 'mType', 'Timestamp', 'quantity', 'seller',
       'endDate', 'sType', 'Product', 'terminationDate', 'party',
       'transactionType'],
      dtype='object')
```

In [108…

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
x=df.drop(["Reporting Status","clDateTime","cDateTime","expirationDate","Timestamp","qu
y=df["Reporting Status"]
# convert the categorical columns to one hot encoded
# Turn the categories into numbers
catagorical_features=["regulator","assetClass","cflag","method","eventT","mType","selle
one_hot=OneHotEncoder()
```

```
transformer1=ColumnTransformer([("one_hot",one_hot,catagorical_features)],remainder='pa
transformed_x=transformer1.fit_transform(x)
transformed_x
```

Out[108…

```
array([[1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        3.20462670e+03, 5.01724834e+05, 1.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        1.06169430e+03, 5.01724834e+05, 4.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        9.33894220e+03, 5.01724834e+05, 4.00000000e+00],
       ...,
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        7.75840550e+03, 5.01724834e+05, 4.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        5.61898800e+03, 5.01724834e+05, 1.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,
        3.51909030e+03, 5.01724834e+05, 3.00000000e+00]])
```

In [109…

```
#spliting data into training and testing data
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
x_train,x_test,y_train,y_test=train_test_split(transformed_x,y)
clf = RandomForestClassifier()
clf.fit(x_train,y_train)
```

Out[109…

```
RandomForestClassifier()
```

In [110…

```
clf.score(x_test,y_test)
```

Out[110…

```
0.7112714651077823
```

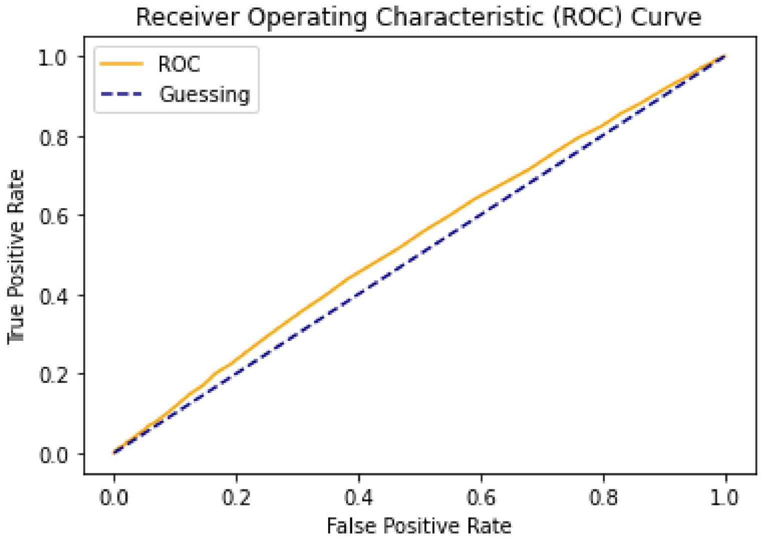
In [116…

```
#Metrics to evaluate a model


from sklearn.metrics import roc_curve
y_probs = clf.predict_proba(x_test)
y_probs = y_probs[:, 1]
# Calculate fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
```

In [114…

```
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='Guessing')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```



In [115…

```
#The maximum ROC AUC score you can achieve is 1.0 and generally, the closer to 1.0, the
from sklearn.metrics import roc_auc_score
roc_auc_score_value = roc_auc_score(y_test, y_probs)
roc_auc_score_value
```

Out[115…

```
0.5338642168721343
```

In [123…

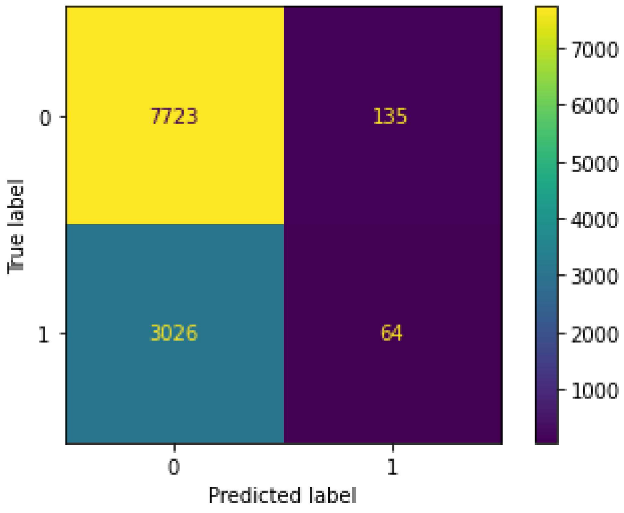
```
from sklearn.metrics import confusion_matrix
y_preds = clf.predict(x_test)
confusion_matrix(y_test, y_preds)
```

Out[123…

```
array([[7723,  135],
       [3026,   64]], dtype=int64)
```

In [124…

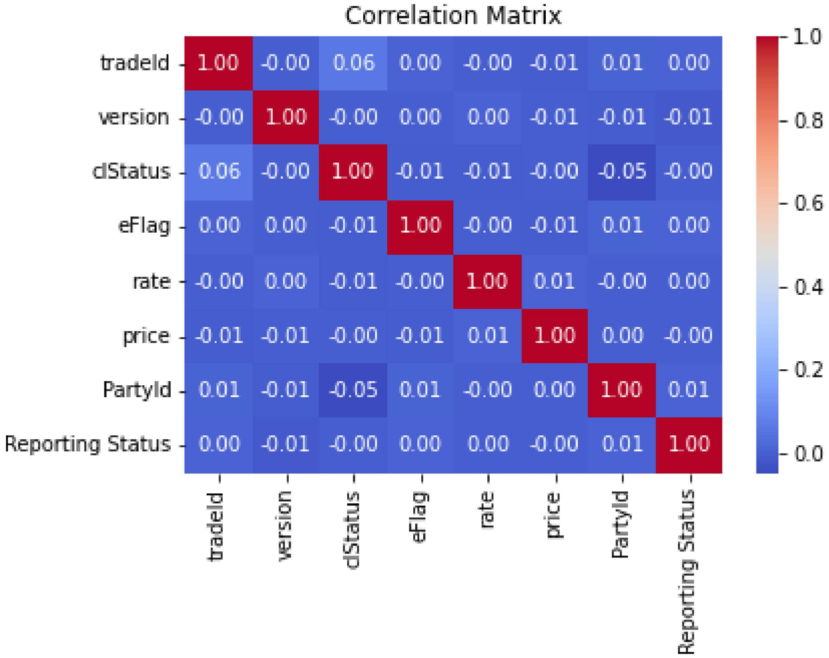
```
ConfusionMatrixDisplay.from_predictions(y_true=y_test,
                                        y_pred=y_preds);
```



In [128…

```
#corelation matrix
correlation_matrix=df.corr()
import seaborn as sns
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

```
plt.title('Correlation Matrix')
plt.show()
```



In [129…

```
correlation_matrix
```

Out[129…

| | tradeId | version | clStatus | eFlag | rate | price | PartyId | Reporting Status |
|---|---|---|---|---|---|---|---|---|
| **tradeId** | 1.000000 | -0.000376 | 0.063366 | 0.003022 | -0.002088 | -0.005872 | 0.007981 | 0.004449 |
| **version** | -0.000376 | 1.000000 | -0.001173 | 0.000336 | 0.004245 | -0.005340 | -0.009614 | -0.011364 |
| **clStatus** | 0.063366 | -0.001173 | 1.000000 | -0.005795 | -0.008206 | -0.002422 | -0.046435 | -0.001965 |
| **eFlag** | 0.003022 | 0.000336 | -0.005795 | 1.000000 | -0.002711 | -0.006410 | 0.009110 | 0.003076 |
| **rate** | -0.002088 | 0.004245 | -0.008206 | -0.002711 | 1.000000 | 0.006271 | -0.000908 | 0.002259 |
| **price** | -0.005872 | -0.005340 | -0.002422 | -0.006410 | 0.006271 | 1.000000 | 0.001381 | -0.001045 |
| **PartyId** | 0.007981 | -0.009614 | -0.046435 | 0.009110 | -0.000908 | 0.001381 | 1.000000 | 0.005163 |
| **Reporting Status** | 0.004449 | -0.011364 | -0.001965 | 0.003076 | 0.002259 | -0.001045 | 0.005163 | 1.000000 |

In [132…

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_preds)
report
```

Out[132…

```
'              precision    recall  f1-score   support\n\n           0       0.72
0.98      0.83      7858\n           1       0.32      0.02      0.04      3090\n\n    a
ccuracy                           0.71     10948\n   macro avg       0.52      0.50
0.43     10948\nweighted avg       0.61      0.71      0.61     10948\n'
```

In [ ]: