DOMAIN ORIENTED CASE STUDY

SCHUSTER MULTINATIONAL COMPANY

PROBLEM STATEMENT

- Schuster is a multinational retail company specializing in sports goods and accessories. They conduct significant business with multiple vendors, with whom they have credit arrangements. However, some vendors tend to make late payments, leading to financial impact and non-value-added activities for Schuster's employees who must chase payments.
- **Expected Results:**
- Conduct exploratory data analysis to extract useful insights.
- Segregate customers into different groups based on payment patterns.
- Build a classification model for predicting delayed payments.
- Apply the model on open invoice data and identify customers requiring precautionary measures.

DATA DICTIONARY

- Received Payment Data

RECEIPT_METHOD
CUSTOMER_NAME
CUSTOMER_NUMBER
RECEIPT_DOC_NO
RECEIPT_DATE
CLASS
CURRENCY_CODE
Local Amount
USD Amount
INVOICE_ALLOCATED
INVOICE_CREATION_DATE
DUE_DATE
PAYMENT_TERM
INVOICE_CLASS
INVOICE_CURRENCY_CODE
INVOICE_TYPE

- Open Invoice Data

AS_OF_DATE
Customer Type
Customer_Name
Customer Account No
Transaction Number
Transaction Date
Payment Term
Due Date
Transaction Currency
Local Amount
Transaction Class
AGE
USD Amount
INV_CREATION_DATE

RECEIVED PAYMENT DATASET OVER VIEW

```
df_received = pd.read_csv('Received_Payments_Data.csv')
df_received.head()                                    #Reading the dataset :Received_Payments_Data
```

	RECEIPT_METHOD	CUSTOMER_NAME	CUSTOMER_NUMBER	RECEIPT_DOC_NO	RECEIPT_DATE	CLASS	CURRENCY_CODE	Local Amount	USD Amount	INV
0	WIRE	C EA Corp	37403	1.421000e+10	20-Apr-21	PMT	USD	370990.92	101018.63040	
1	WIRE	RADW Corp	4003	9.921000e+10	31-Jan-21	PMT	SAR	183750.00	48990.21133	
2	WIRE	RADW Corp	4003	9.921000e+10	31-Jan-21	PMT	SAR	157500.00	41991.60971	
3	WIRE	FARO Corp	1409	9.921000e+10	31-Jan-21	PMT	SAR	157500.00	41991.60971	
4	WIRE	RADW Corp	4003	9.921000e+10	31-Jan-21	PMT	SAR	157500.00	41991.60971	

OPEN INVOICE DATASET OVERVIEW

```
df_open = pd.read_csv(r'C:\Users\91926\Desktop\Kaushal Shah\Open_Invoice_data.csv',encoding='ISO-8859-1')
df_open.head()                                                    #Reading the dataset: Open_Invoice_data
```

Customer Type	Customer_Name	Customer Account No	Transaction Number	Transaction Date	Payment Term	Due Date	Transaction Currency	Local Amount	Transaction Class	AGE	USD Amount	INV_CREATION_DATE
3rd Party	GIVE Corp	49144.0	100210000438	21/12/2021	Immediate	21/12/2021	AED	-3,088	CREDIT NOTE	105	-3,088	12/21/2021 12:53
Related Party	AL J Corp	23152.0	100220000052	01/02/2022	30 Days from Inv Date	03/03/2022	USD	2,000	INVOICE	33	2,000	2/1/2022 14:09
Related Party	AL J Corp	23152.0	100220000143	24/03/2022	30 Days from Inv Date	23/04/2022	USD	2,000	INVOICE	-18	2,000	3/24/2022 17:46
Related Party	AL R Corp	23312.0	100220000001	04/01/2022	15 Days from Inv Date	19/01/2022	AED	2,415	INVOICE	76	2,415	1/5/2022 11:49
Related Party	ALLI Corp	7530.0	100220000105	03/03/2022	30 Days from EOM	30/04/2022	AED	3,800	INVOICE	-25	3,800	3/3/2022 22:30

CUSTOMER SEGMENTATION BASED ON AVERAGE PAYMENT TIME AND STANDARD DEVIATION

```
Cust_Payment_stats = df_received.groupby('CUSTOMER_NUMBER').agg({'PAYMENT_TERM_DAYS':['mean','std']}).reset_index()
Cust_Payment_stats.columns = ['CUSTOMER_NUMBER','Avg Payment Term','Std Payment Term']
```

```
Cust_Payment_stats.head()
```

	CUSTOMER_NUMBER	Avg Payment Term	Std Payment Term
0	1044	32.766594	84.586212
1	1076	29.200000	4.417706
2	1146	73.316667	12.528600
3	1154	34.127660	41.741713
4	1192	37.851852	17.649807

```
Cust_Payment_stats.isnull().sum()
```

```
CUSTOMER_NUMBER       0
Avg Payment Term      0
Std Payment Term    156
dtype: int64
```

```
Cust_Payment_stats.fillna(0, inplace=True)
```

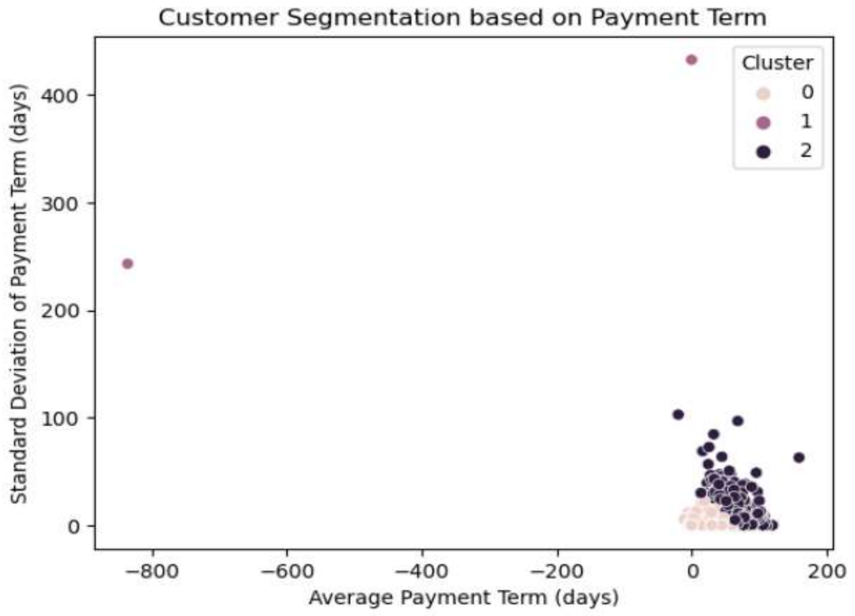
```
Cust_Payment_stats.isnull().sum()
```

```
CUSTOMER_NUMBER    0
Avg Payment Term   0
Std Payment Term   0
dtype: int64
```

```
scaler = StandardScaler()
X = Cust_Payment_stats[['Avg Payment Term', 'Std Payment Term']]
X_scaled = scaler.fit_transform(X)
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
Cust_Payment_stats['Cluster'] = kmeans.fit_predict(X_scaled)
```

```
sns.scatterplot(data=Cust_Payment_stats, x='Avg Payment Term', y='Std Payment Term', hue='Cluster')
plt.title('Customer Segmentation based on Payment Term')
plt.xlabel('Average Payment Term (days)')
plt.ylabel('Standard Deviation of Payment Term (days)')
plt.show()
```



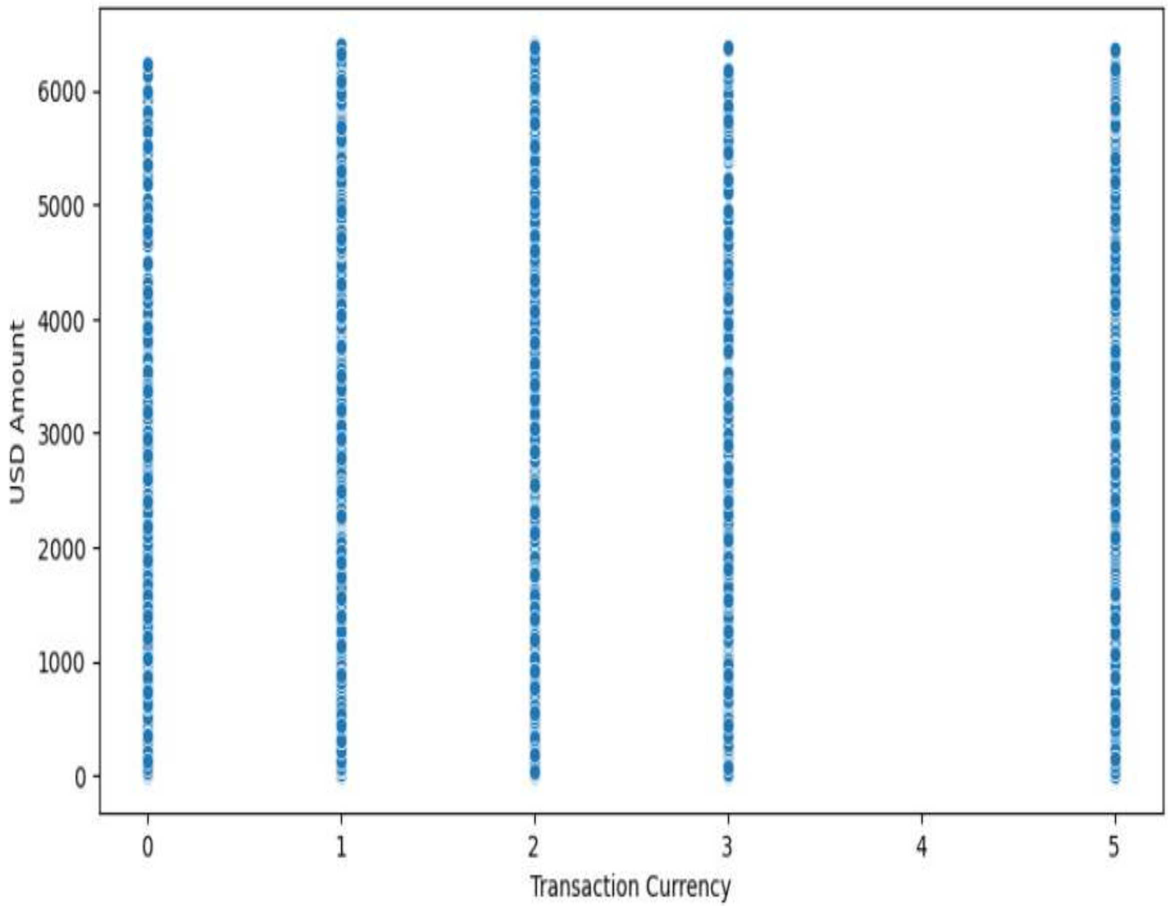
EXPLORATORY DATA ANALYSIS

Data Visualization and Outlier Treatments

```
#CORRELATION THROUGH HEATMAP
plt.figure(figsize=(15,10))
sns.heatmap(df_open.corr(),annot=True)
plt.show()
```

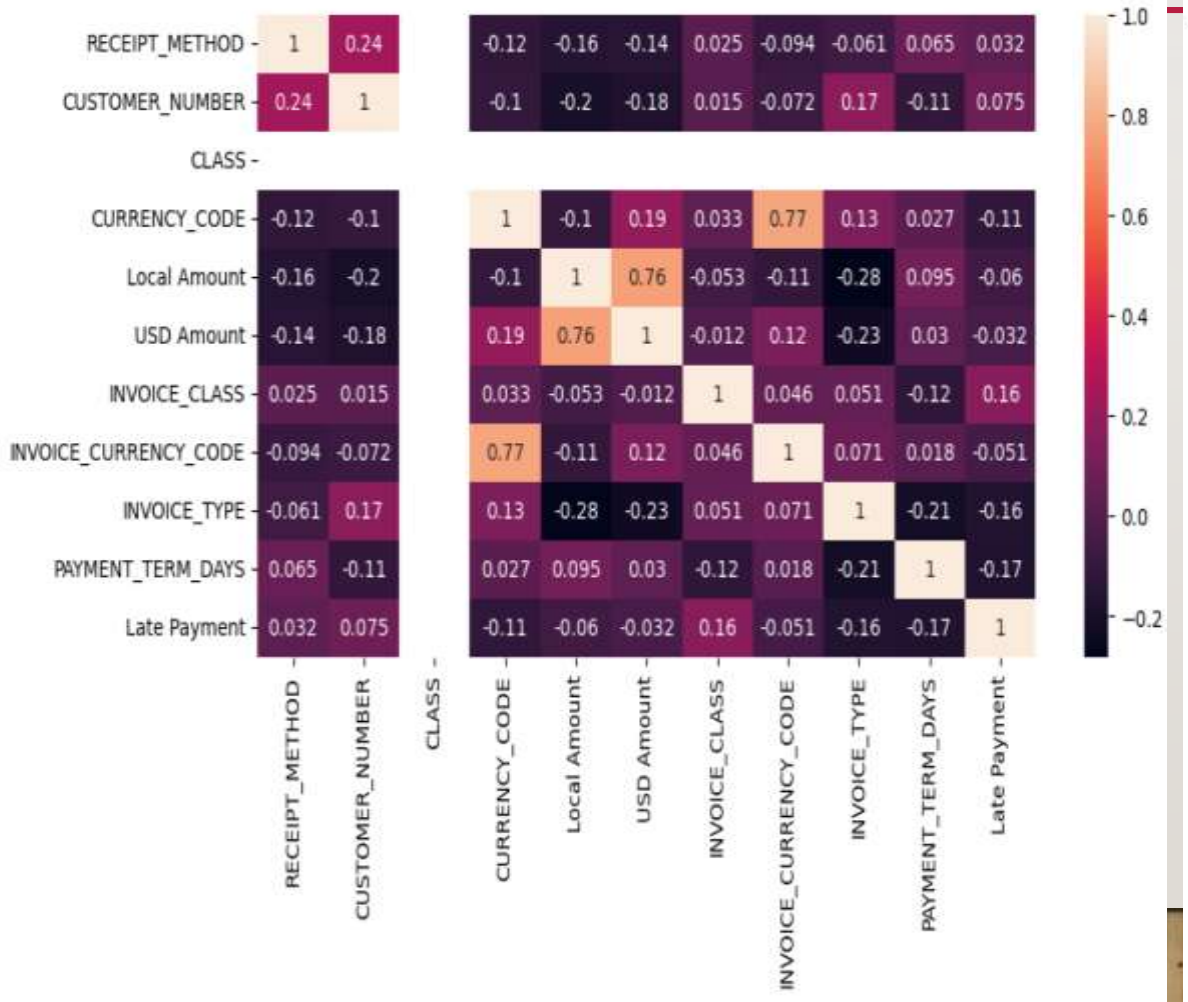

```
#Scatter plot
plt.figure(figsize=(10,5))
sns.scatterplot(data=df_open,x='Transaction Currency',y='USD Amount')
```

<AxesSubplot:xlabel='Transaction Currency', ylabel='USD Amount'>



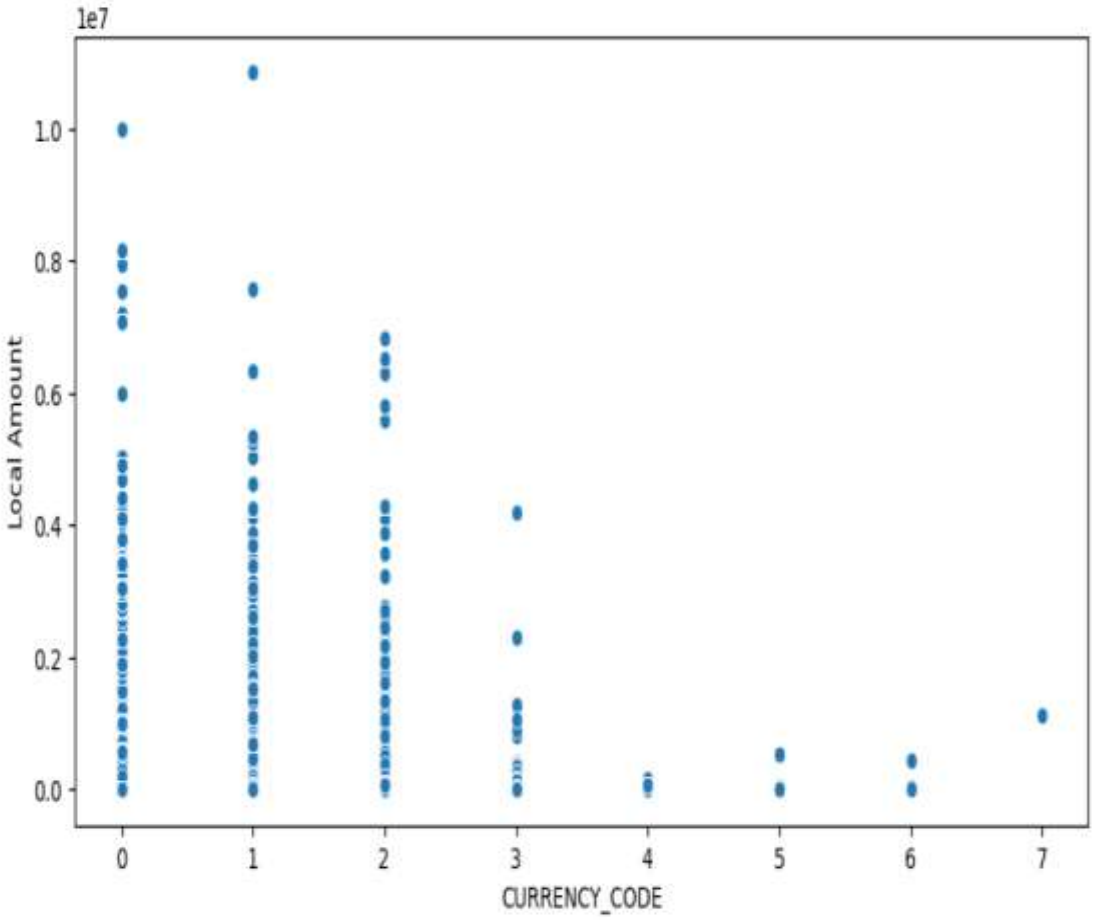
SCATTER PLOT AND CORRELATION PLOT OF RECEIVED PAYMENT DATA

```
plt.figure(figsize=(10,5))
sns.heatmap(df_received.corr(),annot=True)
plt.show()
```




```
#scatter plot
plt.figure(figsize=(10,5))
sns.scatterplot(data=df_received,x='CURRENCY_CODE',y='Local Amount')
```

<AxesSubplot:xlabel='CURRENCY_CODE', ylabel='Local Amount'>



MODEL BUILDING AND MODEL SELECTION

Model Building and Feature Selection

```
features = ['USD Amount', 'PAYMENT_TERM_DAYS', 'Cluster']
```

```
#Train Test Split of the dataset
```

```
X = df_received[features]
y = df_received['Late Payment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Random Forest Classifier

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

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

```
y_pred = rf_classifier.predict(X_test)
print("Testing Accuracy")
print(rf_classifier.score(X_test,y_test))
print("Training Accuracy")
print(rf_classifier.score(X_train,y_train))
```

```
Testing Accuracy
0.8786965376782078
Training Accuracy
0.9408529879017475
```

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

```
[[3343  854]
 [ 635 7443]]
              precision    recall  f1-score   support

           0       0.84      0.80      0.82      4197
           1       0.90      0.92      0.91      8078

    accuracy                           0.88     12275
   macro avg       0.87      0.86      0.86     12275
weighted avg       0.88      0.88      0.88     12275
```

Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
y_pred1 = lr.predict(X_test)
print("Testing Accuracy")
print(lr.score(X_test,y_test))
print("Training Accuracy")
print(lr.score(X_train,y_train))
```

```
Testing Accuracy
0.6580040733197556
Training Accuracy
0.6544462096215732
```

```
print(confusion_matrix(y_test,y_pred1))
print(classification_report(y_test,y_pred1))
```

```
[[    0 4197]
 [    1 8077]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4197
           1       0.66      1.00      0.79      8078

    accuracy                           0.66     12275
   macro avg       0.33      0.50      0.40     12275
weighted avg       0.43      0.66      0.52     12275
```

Gradient Boosting Model

```
gr_model = GradientBoostingClassifier(n_estimators=100,random_state=42)
gr_model.fit(X_train,y_train)
```

```
GradientBoostingClassifier(random_state=42)
```

```
ypred1 = gr_model.predict(X_test)
print("Testing Accuracy")
print(gr_model.score(X_test,y_test))
print("Training Accuracy")
print(gr_model.score(X_train,y_train))
```

```
Testing Accuracy
0.769775967413442
Training Accuracy
0.7735956658112346
```

```
print(confusion_matrix(y_test,ypred1))
print(classification_report(y_test,ypred1))
```

```
[[2252 1945]
 [ 881 7197]]
              precision    recall  f1-score   support

           0       0.72      0.54      0.61      4197
           1       0.79      0.89      0.84      8078

    accuracy                           0.77     12275
   macro avg       0.75      0.71      0.73     12275
weighted avg       0.76      0.77      0.76     12275
```

Model Building on Open Invoice Data

```
df_open.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54817 entries, 1 to 1
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Customer Account No   54817 non-null  float64
 1   Transaction Currency  54817 non-null  int64
 2   Local Amount          54817 non-null  float64
 3   Transaction Class     54817 non-null  int64
 4   USD Amount            54817 non-null  float64
 5   Payment Term days     54817 non-null  int64
dtypes: float64(3), int64(3)
memory usage: 2.9 MB
```

```
df_open.head()
```

	Customer Account No	Transaction Currency	Local Amount	Transaction Class	USD Amount	Payment Term days
Customer Type						
1	23152.0	2	2000.0	0	2000.0	60
1	23152.0	2	2000.0	0	2000.0	30
1	23312.0	1	2415.0	0	2415.0	-72
1	7530.0	1	3800.0	0	3800.0	58
1	7530.0	1	1264.0	0	1264.0	58

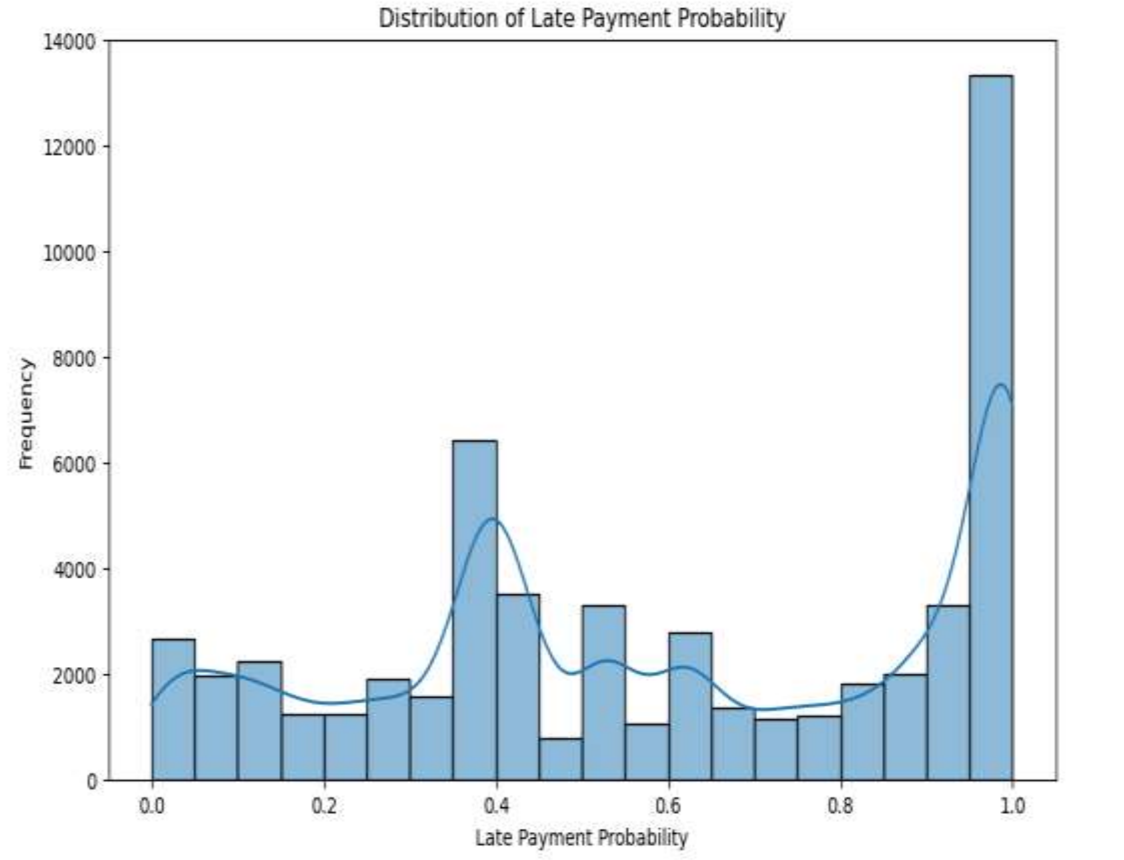
```
df_open.rename(columns={'Customer Account No':'CUSTOMER_NUMBER'},inplace=True)
```

DISTRIBUTION OF LATE PAYMENT PROBABILITY

```python
df_open.rename(columns={'Payment Term days': 'PAYMENT_TERM_DAYS'}, inplace=True)
```

```python
df_open['Late Payment Probability'] = rf_classifier.predict_proba(df_open[features])[:, 1]
```

```python
plt.figure(figsize=(10, 6))
sns.histplot(data=df_open, x='Late Payment Probability', bins=20, kde=True)
plt.title('Distribution of Late Payment Probability')
plt.xlabel('Late Payment Probability')
plt.ylabel('Frequency')
plt.show()
```




```python
plt.figure(figsize=(20,10))
high_risk_customers = df_open[df_open['Late Payment Probability'] > 0.5]
print("\nHigh Risk Customers for Precautionary Measures:",high_risk_customers.max().plot(kind='line'))
```

High Risk Customers for Precautionary Measures: AxesSubplot(0.125,0.11;0.775x0.77)